SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

Vulnerability Analysis and Mitigation of Web Applications Using Static Code Analysis and SSDLC Approach

Evan Samuel Reinheart Mamesah[1], Jimmy Herawan Moedjahedy[2*]

Faculty of Computer Science, Universitas Klabat, Airmadidi, Indonesia

Author Email: s22110392@student.unklab.ac.id[1], jimmy@unklab.ac.id[2]

Abstract. In today's age of technology, web applications have become essential parts of the environment. Due to easy accessibility of the internet, the user can engage in actions without considering the possible impact, leading individuals and organizations to commit such actions freely. As a result, there has been an upsurge in cyber-attacks against web applications that are prone to attacks. Mitigation is achieved using static code analysis with SonarQube in order to detect vulnerabilities within the web application code. In this case, the aim is to contribute to developers through the recommendation on how to develop their web applications while considering security aspects. SSDLC is utilized to manage the web application used during the simulation of the cyber-attacks and mitigation of the impacts thereof. Results are reported as comparisons made before and after mitigation. Before implementation of mitigation measures, the web application was vulnerable to all simulated cyber-attacks. After implementation of mitigation measures, it became clear from analyses that the attack had no way of exploiting the secured vulnerabilities.

Keywords: Web Application, Static Code Analysis, SonarQube, Vulnerabilities, Cyberattack

1 Introduction

In today's digitalized world, cybersecurity is a significant problem considering the use of advanced technology in all actions. It helps keep safety and connectivity through firewalls, encryption, access control, and antivirus programs. Nonetheless, modern-day cybersecurity is still prone to cyber-attacks, such as Distributed Denial of Service (DDoS), Brute Force, SQL Injection, and Cross-Site Scripting because of weaknesses in the systems, which remain concealed from developers [1]. The most recent report published by ISACA (Information Systems Audit and Control Association) about attacks in 2023 was that Social Engineering accounted for 15%, Advanced Persistent Threats (APTs) – for 11%, Security Misconfiguration, Ransomware, and Unpatched Systems each – for 10%, Sensitive Data Exposure, Denial of Service – for 9%, Broken Authentication and Third-Party Risks each – for 8%, and Injection Flaws – for 7%. One of the most complex cyber-attacks, being rather invisible and hard to detect, is known as the Zero-Day Attack. This attack exploits vulnerabilities of a system for which there is no patch available at the time of the attack, and to hide the existence of the threat requires thorough examination to find out the truth [3]. Researchers have noted that to detect those vulnerabilities, debugging, particularly through static code analysis, may prove useful, as it helps to identify possible weaknesses in the code of a system and generate mitigation to prevent further similar attacks [4]. Therefore, the utilization of a tool specifically designed for remedial static code analysis, such as SonarQube Community Edition, will help detect and mitigate vulnerabilities without executing the source code [5].

There is ample information available on the topic of cyber-attacks targeted at individuals, along with available detection measures. However, the research problem lies in the fact that there is a significant knowledge gap in the literature. While most research focus on vulnerability detection or theoretical mitigation strategy development, few of them offer a simulation platform to evaluate the pre- and post-mitigation results of such attacks, within the SSDLC model. Therefore, this research paper aims to fill this research gap through a thorough phase-to-phase examination of web application security improvement. In this regard, the research uses a simulated laboratory with Windows 10 and Kali Linux operating systems to show how Secure Software Development Life Cycle (SSDLC) model implementation can help move a web application from the vulnerable phase to the protected phase.

This research adopts a web application that has thow phases (Vulnerable phase and Protected phase) which intended to be used for simulation purpose of Cyberattacks. It will show the before and after of the implementation of the mitigation once it analyzed the vulnerabilities in it and see how effective it is for the web application. The approach in this research is using SSDLC to ensure each phase of the web application was done in the right order for the vulnerabilities to be located and finding the possible fixes [6]. When considering the complexity involved

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

in cyberattacks, such attacks typically involve exploiting a vulnerability in the targeted system through a motive for which they may be instigated by various reasons either as individual or group entities [7]. The following section discusses several common types of attack vectors. Distributed Denial of Service (DDoS) refers to a disruption technique that involves overwhelming the targeted system using a large amount of malicious traffic and subsequently causing a service outage [8]. Botnets are considered part of the DDoS attacks as they are comprised of multiple hijacked computers that are used to send a huge number of requests to the target system [9]. Another prominent threat is known as SQL Injection attacks where the hacker uses the malicious scripts to infiltrate the database present in the targeted system [10]. In brute force attacks, the objective of the attacker is to break into the network system through various login credentials by guessing them all [11]. Lastly, Remote Code Execution refers to an attack where the malicious code is executed in the targeted system from a remote location [12]. Cross-Site Scripting is a web-based attack method that has a similar attack signature to SQL Injection but targets web interfaces rather than database management systems [13]. Social Engineering refers to cyberattacks that involve manipulating people instead of computers, where personal and private data like login credentials or other confidential information is gathered from targeted users unknowingly [14]. A similar method used is called Spoofing, where an attacker masquerades as a reputable business or entity to acquire important information from a victim [15]. Phishing is a type of social engineering and spoofing-based attack where an attacker pretends to be a credible website, application, or user profile to gather personal information from the user by input fields [16]. Identity Theft involves using someone else's authentic information, like a national identification card, to misuse confidential information and assets of that individual [17]. Man-in-the-Middle attacks occur when an attacker intercepts a communication session between two individuals without their knowledge [18]. Malware describes software that is created specifically for surveillance, data theft, disruption, or destruction of the target computer system. The most common types of malwares include viruses, worms, Trojan horses, ransomware, spyware, rootkits, keyloggers, and magic bomb software [19].

The Web application environment was set up using JavaScript as the backend programming language, MySQL to store the data being entered, Node.js to run the server side, and React.js to run the frontend side. Two different operating systems had to be used to run both the Vulnerability Phase and Protected Phase; specifically, Windows 10 was used as the Host OS, while Kali Linux was used as the Guest OS [20].

2 Research Method

The chronological methods of this research will be explained in this section, including the SSDLC, Research Procedure, Simulation Flow and Research Environment.

2.1 Secure System Development Life Cycle

Secure Software Development Life Cycle acts as the methodology adopted by the current study for the web application used as the simulation object in implementing the cyber-attack as shown on Figure 1. SSDLC is aimed at minimizing risks of vulnerability in every development phase. The phases involved include:

a. Requirements: Identification of key features needed from the web application in terms of functionality and security.
b. Design: Creation of the web application according to its requirements.
c. Development: Improvements regarding usability and security are emphasized in this web application.
d. Testing: Testing the design created in the previous phase for identification of the web application vulnerabilities.
e. Deployment: The development process is repeated until there is a reduced number of vulnerabilities before deploying the web application

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

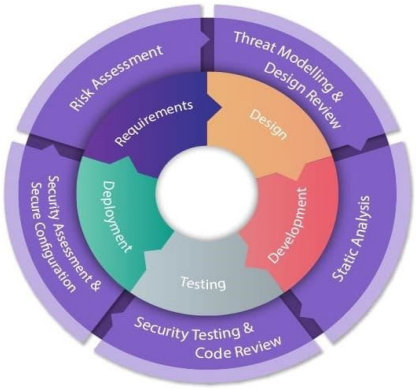

Figure 1. Architecture of a typical wireless sensor node.

2.2 Research Procedure

The simulation process starts with the selection of the web application which is to be used as the target for the simulation process. After deciding on the web application, the next process is to select a suitable simulation tool that would undertake the simulation process via the static code analysis using the SonarQube. In following the set rules by the selected simulation tool, the simulation process starts with identifying the vulnerability state of the web application, and then mitigation is performed in order to gauge the level of compatibility between the two states of the web application.

2.3 Simulation Flow

As illustrated in Figure 2 below, each of the stages involved in the simulation process has been captured. For purposes of enhancing clarity in understanding how the simulation process works, the following key points can be highlighted:

a. Determining the web application that will be the core of simulation.
b. Selecting SonarQube as the tool of static code analysis.
c. Applying the rules inside SonarQube for vulnerability testing.
d. Running the analysis process based on the applied rules.
e. Analyzing the output.
f. Applying mitigation measure based on the output from vulnerability analysis.

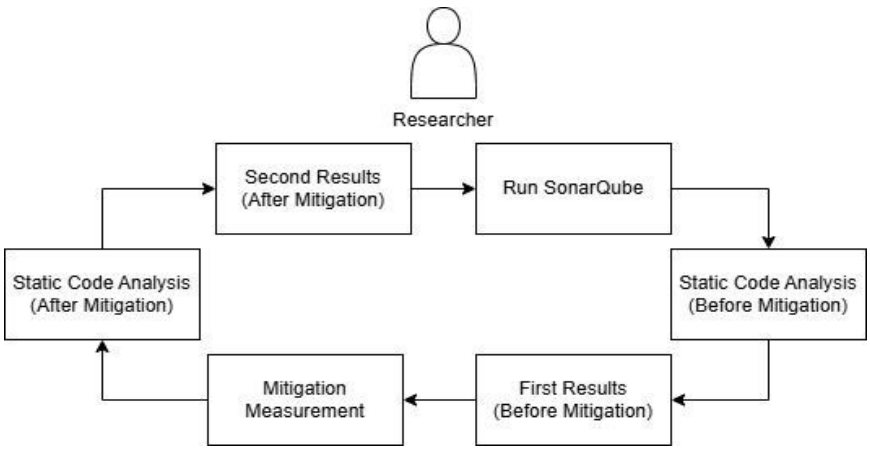

Figure 2. Simulation Flow

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

2.4 Research Environment

Figure 3 demonstrates the execution of this activity in a virtualized environment, in line with the experiment's two operating systems. This system involves two operating systems, whereby Windows 10 acts as the host operating system and controls the web applications, whereas Kali Linux runs concurrently with the former, with the main task of exploiting the web applications.

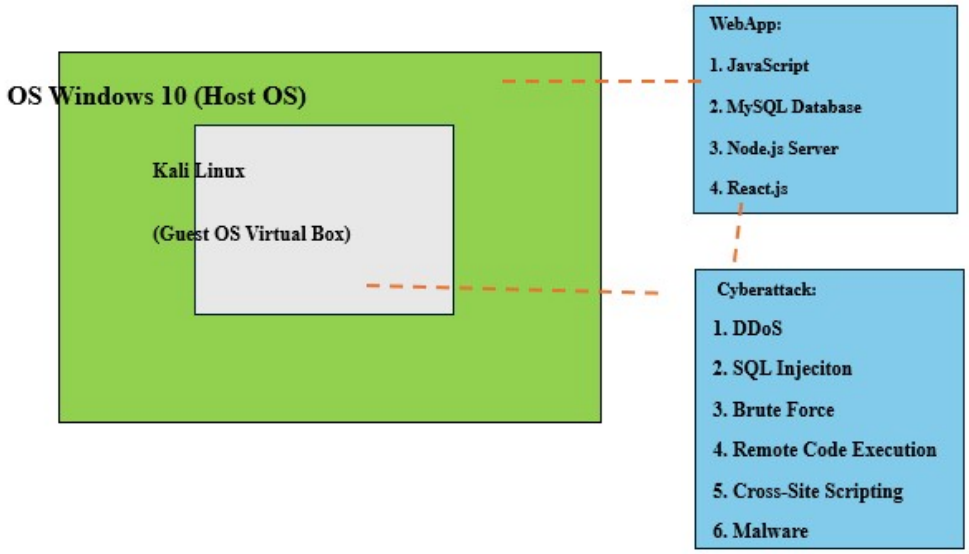

Figure 3. Research Environment Simulation

3 Result and Discussion

3.1 Web Application for Simulation

The SignUp page, shown in Figure 4, is utilized for the purpose of generating a new user account. The SignUp page features an XSS vulnerability, which exists in the vulnerable version of the web application that may be exploited to break into or breach the system. Following the creation of the account, users are redirected to the Login page of the web application. Sadly, the Login page suffers from SQL Injection vulnerabilities in the vulnerable version of the web application.

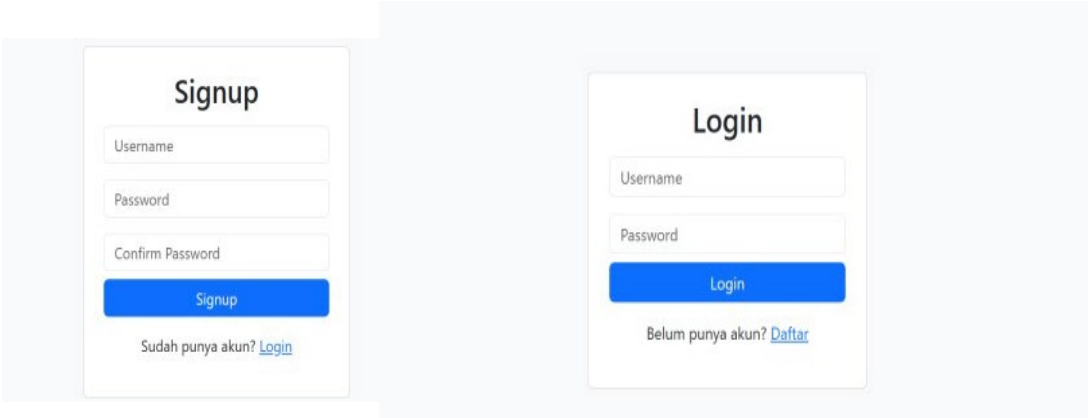

Figure 4. Sign Up and Login Page

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

After successful verification, a web application interface appropriate for the simulation test is displayed. The web application consists of five pages, namely, Home, About, Service, and Contact, as shown in Figure 5. The About page contains clear information on the purpose of the selected web application, which is specifically for simulation to detect hidden vulnerabilities when under cyber-attacks. On the other hand, the Service page lists down the possible choices, but for now, it mainly serves the purpose of displaying the choices. The Contact page is meant to store the input data collected; however, it can be prone to cyber-attacks such as SQL Injection and XSS, as demonstrated in Figure 6.

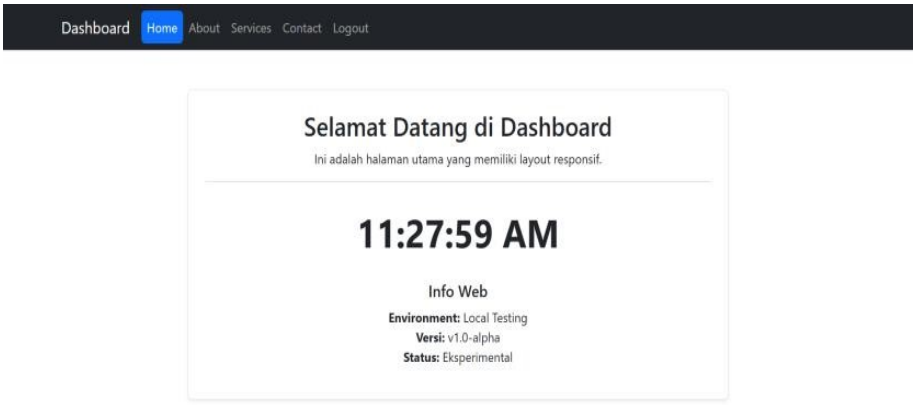

Figure 5. Homepage

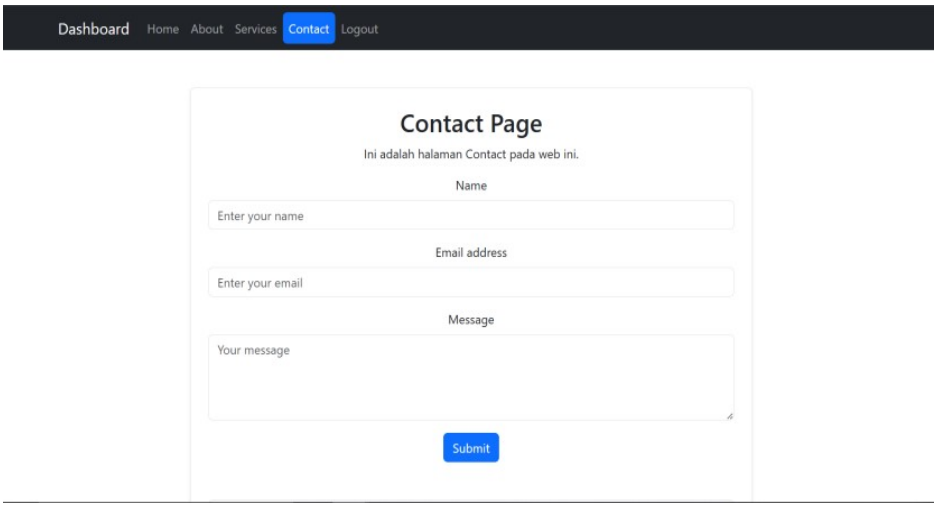

Figure 6. Contact Page that contain form

Figure 7 shows the results of the analysis, showing that the web application is unable to meet the standards necessary for obtaining the Pass status at the Quality Gate. The reason behind this lies in the web application being wholly susceptible to any form of attack by the hacker through the web application itself. There are five Security Hotspots identified in the assessment, which may be susceptible to cyber-attacks such as SQL Injection, XSS, or even hardcoded passwords.

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

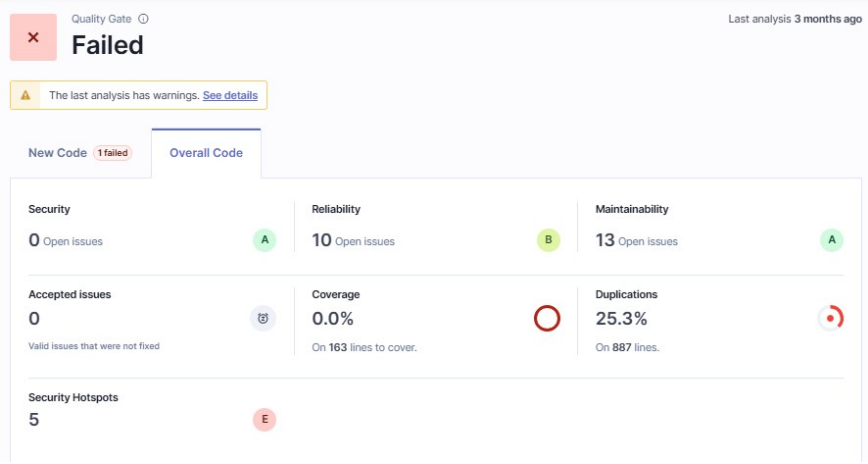

Figure 7. Analysis Before Mitigation

After the implementation of the mitigation strategy, the output produced is different, as shown in Figure 8 below. In this secure stage of the web application, the risks within the Security Hotspots are no longer present. This implies that the vulnerabilities associated with these hotspots have been resolved, thanks to the mitigation approach introduced previously through analysis. Thus, it facilitates the detection and resolution of any remaining weaknesses in the existing web application.

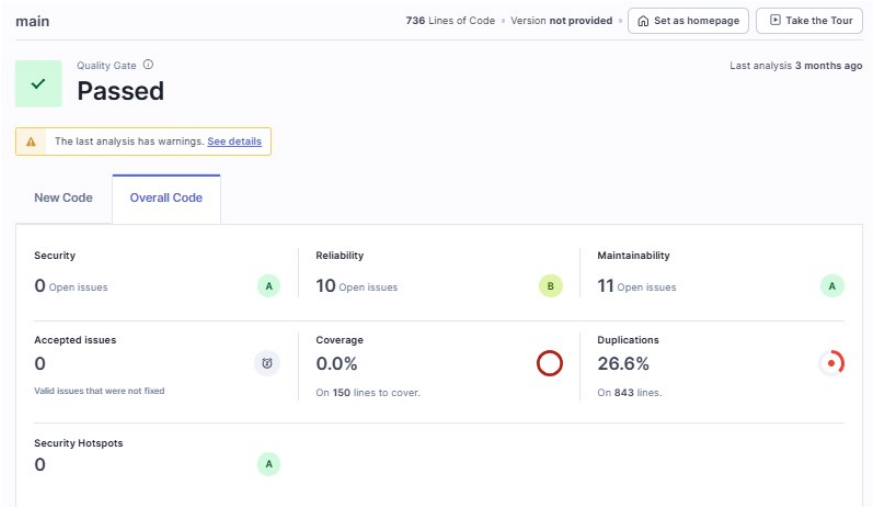

Figure 8. Analysis After Mitigation

Following the identification of the weaknesses that have been exposed in the web application, this study also sets out some rules that would ensure that similar mistakes do not occur again in the vulnerable conditions of the web application. Following the rules set out in Table 1 below can significantly reduce the probability of any hidden problems that could emerge in the developed web application.

Table 1. Recommended Actions

Analysis Results	Recommended Actions	Results
There are function dangerouslySetInnerHTML usage within web application.	Make sure not to use dangerouslySetInnerHTML.	Protected from Cross-Site Scripting.
The source code doesn't have Input Validation.	Implement validation based on data types (int, string, etc).	SQL injection cannot penetrate the web application.

SAGA: Journal of Technology and Information Systems
Vol 4, Issue 1, February 2026, Page 591-598
ISSN: 2985-8933 (Media Online)
DOI: 10.58905/SAGA.v4i1.630

There were found several dangerous function in the source code that could lead to unauthorized access.	If not necessary, make sure not to use the following commands: exec(), system(), or eval().	Reducing the occurrence risk of Remote Code Execution.
Password quality appears to be simple.	Make sure to use hashing when creating password.	If a data leak were occured, no data will be easily stolen.
Unlimited login attempt from the client side.	Add rate limiting on the source code.	Preventing Brute Force from trying to break in.
Sensitive informations such as databases are not properly protected.	Use .env command to store web application's confidental informations.	It will no longer be exposed (hardcoded).
No validation between frontend and backend.	Apply validations towards backend and frontend.	Mitigates the next possible cyberattacks on the backend side or frontend side.
Low complexity password, high potential to be easily hacked.	Make sure to create high complexity password to prevent further unauthorized login attempt.	It won't be easily penetrated, especially with the tool such as Brute Force.

4 Conclusion

In this research, it has been proven that a particular vulnerability will happen regardless of what actions have been taken before to prevent this from happening. As a result, the necessity for implementation of preventive actions to prevent a threat is crucial through static code analysis using appropriate software like SonarQube. In the results, after the mitigation, the web application security increases significantly. Even though this process cannot guarantee the absence of any attacks, this strategy may help increase the security level of the application in question. However, since vulnerability is found in the code, it may be useful to conduct some other actions as well.

References

[1] W. S. Admass, Y. Y. Munaye, and A. A. Diro, "Cyber security: State of the art, challenges and future directions," *Cyber Security and Applications*, vol. 2, Jan. 2024, doi: 10.1016/j.csa.2023.100031.

[2] ISACA and Adobe, "State of Cybersecurity 2023: Global Update on Workforce Efforts, Resources and Cyberoperations," Schaumburg, 2023.

[3] S. Ali, S. U. Rehman, A. Imran, G. Adeem, Z. Iqbal, and K. Il Kim, "Comparative Evaluation of AI-Based Techniques for Zero-Day Attacks Detection," *Electronics (Switzerland)*, vol. 11, no. 23, Dec. 2022, doi: 10.3390/electronics11233934.

[4] R. Haas, R. Niedermayr, T. Roehm, and S. Apel, "Is static analysis able to identify unnecessary source code?," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 1, Jan. 2020, doi: 10.1145/3368267.

[5] SonarQube, "SonarQube System Requirements," SonarQubeDocumentation. Accessed: Oct. 17, 2024. [Online]. Available: https://docs.sonarsource.com/sonarqube/latest/setup-andupgrade/installation-requirements/server-host/

[6] W. Umeugo Candidate, "SECURE SOFTWARE DEVELOPMENT LIFECYCLE: A CASE FOR ADOPTION IN SOFTWARE SMES," *International Journal of Advanced Research in Computer Science*, vol. 14, no. 1, doi: 10.26483/ijarcs.v14i1.6949.

[7] Y. Li and Q. Liu, "A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments," *Energy Reports*, vol. 7, pp. 8176–8186, Nov. 2021, doi: 10.1016/j.egyr.2021.08.126.

[8] A. Singh and B. B. Gupta, "Distributed Denial-of-Service (DDoS) Attacks and Defense Mechanisms in Various Web- Enabled Computing Platforms: Issues, Challenges, and Future Research Directions," *Int J Semant Web Inf Syst*, vol. 18, no. 1, 2022, doi: 10.4018/IJSWIS.297143.

[9] I. Ali *et al.*, "Systematic Literature Review on IoT-Based Botnet Attack," Nov. 24, 2020, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2020.3039985.

[10] I. Tasevski and K. Jakimoski, "Overview of SQL injection defense mechanisms," in *2020 28th Telecommunications Forum, TELFOR 2020 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020. doi: 10.1109/TELFOR51502.2020.9306676.

[11] S. Zhang, X. Xie, and Y. Xu, "A Brute-Force Black-Box Method to Attack Machine LearningBased Systems in Cybersecurity," *IEEE Access*, vol. 8, pp. 128250–128263, 2020, doi: 10.1109/ACCESS.2020.3008433.

[12] H. Holm, T. Sommestad, U. Franke, and M. Ekstedt, "Success Rate of Remote Code Execution Attacks Expert Assessments and Observations."

[13] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides, "Cross-site scripting (XSS) attacks and mitigation: A survey," *Computer Networks*, vol. 166, Jan. 2020, doi: 10.1016/j.comnet.2019.106960.

[14] H. Aldawood and G. Skinner, "An Advanced Taxonomy for Social Engineering Attacks," *Int J Comput Appl*, vol. 177, no. 30, pp. 1–11, Jan. 2020, doi: 10.5120/ijca2020919744.

[15] H. Tak, J. Patino, M. Todisco, A. Nautsch, N. Evans, and A. Larcher, "End-to-end anti-spoofing with rawnet2," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 6369–6373. doi: 10.1109/ICASSP39728.2021.9414234.

[16] R. Alabdan, "Phishing attacks survey: Types, vectors, and technical approaches," Oct. 01, 2020, *MDPI AG*. doi: 10.3390/fi12100168.

[17] C. M. Gupta and D. Kumar, "Identity theft: a small step towards big financial crimes," Oct. 25, 2020, *Emerald Group Holdings Ltd.* doi: 10.1108/JFC-01-2020-0014.

[18] M. A. Al-shareeda, M. Anbar, S. Manickam, and I. H. Hasbullah, "Review of Prevention Schemes for Man-In-The-Middle (MITM) Attack in Vehicular Ad hoc Networks," *International Journal of Engineering and Management Research*, vol. 10, no. 3, pp. 153–158, Jun. 2020, doi: 10.31033/ijemr.10.3.23.

[19] O. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," 2020, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2019.2963724.

[20] Z. Bin Akhtar and A. T. Rawol, "Uncovering Cybersecurity Vulnerabilities: A Kali Linux Investigative Exploration Perspective," *International Journal of Advanced Network, Monitoring and Controls*, vol. 9, no. 2, pp. 11–22, Jun. 2024, doi: 10.2478/ijanmc-2024-0012.