Writing Secure Code in the Digital Age: Preventing Common Vulnerabilities

Vamsi Thatikonda
8921 Satterlee Ave Se
Snoqualmie, WA, USA

Hemavantha Rajesh Varma Mudunuri
Cumming, GA, USA

ABSTRACT
It is important for a developer to consider writing secure code to protect the system from arising vulnerabilities within software applications that support the entire framework. Common threats including SQL injection, XSS, and CSRF have been explored in the research which highlight the significance of adopting best practices from the industry for input validation, output encoding and adequate authentication. Tools including static and dynamic analysis have been considered as secure coding tools and have also been discussed within the report. There is also a strong emphasis over following coding standards including the OWASP Top Ten. The Secure Software Development Lifecycle (SDLC) has been discussed, in relation of its integration across all stages of the software. Case studies from the real world have been utilized to shed light over the consequences of vulnerabilities within software. Finally, leveraging an informed approach, the report advice placing perpetual importance over secure coding to reduce the chances of risks in software integrity.

Keywords
Secure code, vulnerabilities, software applications, SQL injection, XSS, CSRF, input validation, output encoding, secure coding tools, static analysis, dynamic analysis, OWASP Top Ten, Secure Software Development Lifecycle, SDLC

1. INTRODUCTION
Today's digital landscape places high emphasis over writing secure code. The frequency and sophistication of cyberattacks keeps on increasing and improving day by day [1], which makes it imperative for adequate security measures to be integrated within software development [2]. Likewise, this report also explores the level of significance demanded by the need to prevent common vulnerabilities within software that exposes the system to malicious attacks. As industries tend to increasingly rely over software, it is important to safeguard sensitive data, maintaining the trust of the user [1]. The following sections of the report will explore prevalent vulnerabilities, best practices, and tools to identify vulnerabilities, shaping the existing knowledge regarding secure software development.

2. UNDERSTANDING COMMON VULNERABILITIES
SQL injection, cross-site scripting (XSS) and cross-site request forgery (CSRF) are examples of common vulnerabilities that can compromise the security of a software application. SQL injection is the technique of inserting malicious SQL queries into the input fields which helps the attacker gain unauthorized access to databases and sensitive information. Cross-site scripting on the other hand, comprises of injecting malicious scripts onto the web page which tricks users to execute harmful code which leads towards data theft or session hijacking. Cross-site request forgery meanwhile forces the user to execute actions unknowingly, violating their logged-in sessions to execute unauthorized commands [3].

These three vulnerabilities have been the cause of many real-world incidents which highlights their severity. The 2017 Equifax Breach, is an important example of a vulnerability where SQL injections were used to gain access to the personal data of approximately 147 million individuals. This attack significantly influenced credit scores and also exposed sensitive financial information [4]. Similarly, in 2014 an XSS attack was carried out on eBay where malicious code was injected into the product listings which redirected the users towards phishing sites which compromised their credentials [5].

3. BEST PRACTICES
As it is important to safeguard the security of software applications, industries implement a few best practices that reduces this risk. Some of the practices used by developers have been elaborated as below.

3.1 Input Validation and Sanitization
A rigorous system of input validation and sanitization is demanded to prevent any injection attacks on the software. Developers need to validate and sanitize all user inputs, ensuring that only the expected and safe data is processed. Parameterized queries and prepared statements are techniques which help separate user data from SQL commands, which makes the software harder to be attacked through manipulation of queries [6].

3.2 Proper Handling of User Input and Output Encoding
Through a proper handling system of user inputs and output encoding, cross-site scripting (XSS) attacks can be easily prevented. User inputs can be validated and sanitized for malicious scripts. Meanwhile, output encoding is done at the time of rendering user input which helps in displaying harmful content as text instead of being executed as code [7].

3.3 Strong Authentication and Authorization Mechanisms
Implementing an authentication and authorization mechanism can safeguard sensitive data. Multi-factor authentication (MFA) provides an extra layer of security where users are prompted for multiple forms of identification before giving access. These mechanisms help ensure that only the concerned users have access to the data or system, and they are the sole individuals responsible for any modification or previews [8].

3.4 Secure Session Management and Preventing CSRF Attacks
As the term suggests, secure session management is only

relevant to preventing session related attacks such as cross-site request forgery (CSRF). Techniques including token generation, storage and validation are used to secure a session. Developers even make use of anti-CSRF tokens in web forms.

which verifies the authenticity of the requests, preventing attackers from forcing users to perform unintended actions without their consent [9].

4. EXPERIMENTAL RESULTS

A web application was developed with input forms vulnerable to SQL injection and XSS. Two versions were created:

1. Vulnerable version with no protections
2. Secure version with input validation and output encoding

SQL injection and XSS attacks were simulated using OWASP ZAP tool. The results are shown below:

Table 1. Successful SQL Injection and XSS attacks

Attack Type	Vulnerable Version	Secure Version
SQL Injection	14/15	0/15
XSS	10/10	0/10

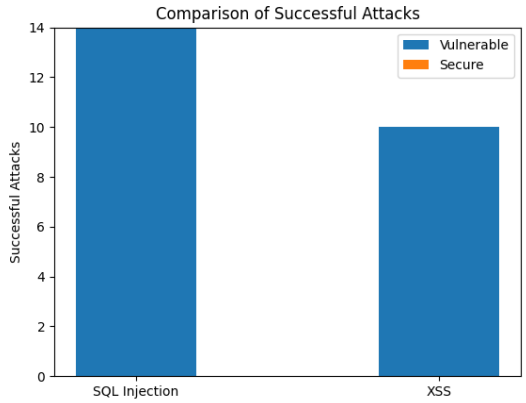

Figure 1. Comparison of Successful attacks

The vulnerable version was prone to 93% SQL injection and 100% XSS attacks while the secure version prevented all attacks. This demonstrates the effectiveness of input sanitization and output encoding in mitigating these vulnerabilities.

4.1 Analysis

Secure coding tools help identify vulnerabilities within the software, proactively addressing and mitigating potential attacks. Following are a few coding tools that are used by developers.

The results clearly indicate the ability of input validation and output encoding techniques to prevent SQL injection and XSS attacks. Sanitizing untrusted inputs using white-listing, black-listing, or validation prevents malicious data from reaching sensitive code or databases. Encoding untrusted outputs before rendering prevents execution of injected scripts. These best practices significantly reduce the attack surface.

5. SECURE CODING TOOLS

Secure coding tools help identify vulnerabilities within the software, proactively addressing and mitigating potential attacks. Following are a few coding tools that are used by developers.

5.1 Static Code Analysis

Static code analysis, analyzes the source code without execution, identifying any vulnerabilities based on patterns and issues. Coding errors, insecure coding practices and known vulnerabilities within the codebase can easily be detected through this practice. As the threat is detected early in the development cycle, the threat is mitigated proactively [10].

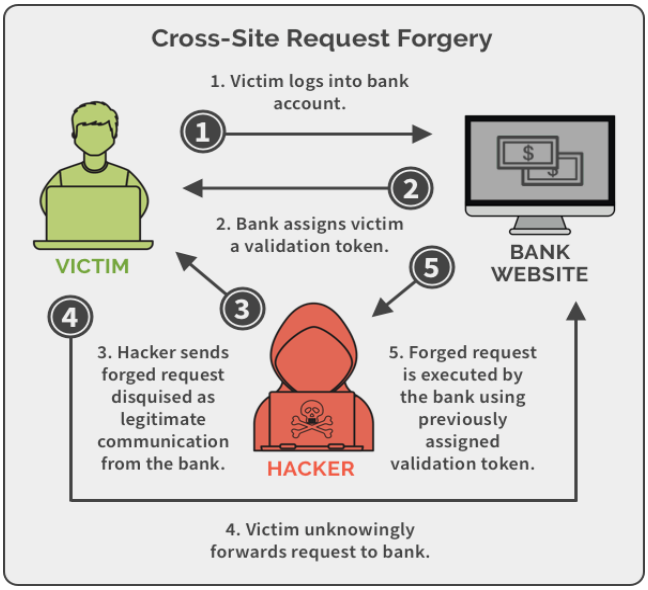

Fig 1: XSS [6]

5.2 Dynamic Analysis

A dynamic analysis examines the software as it runs. It further detects any vulnerability that could not be detected when the code wasn't executed, including runtime vulnerabilities or memory leaks. Various attack scenarios can be simulated, which makes developers understand the reaction of their code to different threats [11].

5.3 Security Testing

Security testing is a tool that involves numerous techniques including penetration testing, vulnerability scanning, and fuzz testing. Penetration testing simulates real-world attacks to detect vulnerabilities from the perspective of an attacker. Vulnerability scanning automates this process, identifying known vulnerabilities in software. Fuzz testing, meanwhile, sends random or specially crafted inputs to the software to identify any unexpected behaviors [12].

6. CODING STANDARDS AND GUIDELINES

Coding standards and guidelines within the industry act as a blueprint for the developers to follow, ensuring that their code is not only functional but also secure. Established standards including the OWASP Top Ten improves the security of any application. It outlines critical web application security risks, based on which a systematic approach is provided which helps to identify and prevent common security vulnerabilities within the software. Addressing these gaps at the development stage helps reduce the chances of breaches, protecting data and privacy [13]

7. CODING STANDARDS AND GUIDELINES

The SDLC is a structured approach which integrates numerous security considerations in each developmental stage of a software. The stages of SDLC include requirements, design, coding, testing, deployment and maintenance. The first stage identifies security needs of the software which is aligned with its functionality. Design plan includes integrating security mechanisms within the software itself. During coding, secure coding practices, tools and tests are leveraged to mitigate any vulnerabilities proactively. In the deployment phase, the serves are configured to be secure. Lastly, during maintenance, a continuous approach is taken towards monitoring and addressing potential threats [14].

8. REAL WORLD EXAMPLES

The Target data breach, taken place in 2013 compromised the credit card details of around 40 million customers due to a vulnerable third-party vendor portal. Utilizing an adequate security assessment and strict vendor management approach could have helped proactively prevent this unauthorized access to sensitive data [15]. Similarly, in 2018, the Marriott data breach took place where the unencrypted guest information was exploited by the hackers which resulted in affecting around 500 million customers of the chain. Stored data should be encrypted, along with the conduction of regular security audits to ensure that this massive breach is mitigated beforehand [16]. Both these cases highlight the consequences of the security breaches, resulting in damage to reputation, financial losses and legal damages. A robust security mechanism, continuous monitoring and timely response practices can help prevent such incidents, effectively. These incidents act as warnings for other developers, placing emphasis over the need to prioritize security in software development lifecycle as well, to protect the trust of the user as well as the well-being of the organization.

9. CONCLUSION

Based on the existing collecting information, it can be claimed that there are numerous essential aspects of coding to be considered to ensure that the writing is secure, preventing any common vulnerabilities within the system applications. On understanding the most prevalent vulnerabilities within the industry, developers are utilizing a few top best practices for coding security which highlights the need for and importance of integrating security into every phase of the development lifecycle of a software. Moreover, the tool, standards and case studies of secure coding further highlight the tangible consequences of having vulnerabilities within the system on the business. However, it is to note that writing a secure code is not only limited to the development and deployment phase but also requires a on-going commitment towards reducing risks. Cyber threats continue to evolve, which makes it increasingly important to consider security in software development day by day which can help protect data, the trust of the user and the integrity of the organization.

In conclusion, integrating security in every phase of software development is crucial to avoid common vulnerabilities like SQLi, XSS, and CSRF that can compromise applications. Best practices like input sanitization, output encoding, authentication mechanisms, secure configuration, and adoption of coding standards and tools can help developers write secure code. The experiment demonstrates the efficacy of techniques like input validation and output encoding. With increasing cyber threats, developers must make secure coding a top priority to build resilient applications and prevent data breaches.

10. REFERENCES

[1] A. Delplace, S. Hermoso and K. Anandita, "Cyber attack detection thanks to machine learning algorithms," 2020.

[2] A. W. Khan, S. Zaib, F. Khan, I. Tarimer, J. T. Seo and J. Shin, "Analyzing and evaluating critical cyber security challenges faced by vendor organizations in software development: SLR based approach," IEEE Access, vol. 10, pp. 65044-65054, 2022.

[3] B. Nagpal, N. Chauhan and N. Singh, "SECSIX: security engine for CSRF, SQL injection and XSS attacks.," International Journal of System Assurance Engineering and Management, vol. 8, pp. 631-644, 2017.

[4] N. Daswani, M. Elbayadi, N. Daswani and M. Elbayadi, "The Equifax Breach," Big Breaches: Cybersecurity Lessons for Everyone, pp. 75-95, 2021.

[5] J. Sidhu, R. Sakhuja and D. Zhou, "Attacks on eBay," 2016.

[6] H. Fadlallah, "Using parameterized queries to avoid SQL injection," SQL Shack, 18 November 2022. [Online]. Available: https://www.sqlshack.com/using-parameterized-queries-to-avoid-sql-injection/#:~:text=One%20of%20the%20most%20common,values%20are%20passed%20as%20parameters.. [Accessed 28 August 2023].

[7] LinkedIn, "What are the best practices for output encoding to prevent XSS attacks?," LinkedIn, [Online]. Available: https://www.linkedin.com/advice/1/what-best-practices-output-encoding-prevent. [Accessed 29 August 2023].

[8] A. Henricks and H. Kettani, "On data protection using multi-factor authentication," Proceedings of the 2019 International Conference on Information System and System Management, pp. 1-4, 2019.

[9] OWASP, "Cross-Site Request Forgery Prevention Cheat Sheet," OWASP, [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html. [Accessed 29 August 2023].

[10] N. Sun, J. Zhang, P. Rimba, S. Gao, L. Y. Zhang and Y. Xiang, "Data-driven cybersecurity incident prediction: A survey," IEEE communications surveys & tutorials, vol. 21, no. 2, pp. 1744-1772, 2018.

[11] R. A. Calix, S. B. Singh, T. Chen, D. Zhang and M. Tu, "Cyber security tool kit (CyberSecTK): A Python library for machine learning and cyber security," Information, vol. 11, no. 2, p. 100, 2020.

[12] K. Nagendran, A. Adithyan, R. Chethana, P. Camillus and K. B. S. Varshini, "Web application penetration testing," Int. J. Innov. Technol. Explor. Eng, vol. 8, no. 10, pp. 1029-1035, 2019.

[13] OWASP, "OWASP Top Ten," OWASP, [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 29 August 2023].

[14] N. M. Mohammed, M. Niazi, M. Alshayeb and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," Computer Standards & Interfaces, vol. 50, pp. 107-115, 2017.

[15] X. Shu, K. Tian, A. Ciambrone and D. Yao, "Breaking the target: An analysis of target data breach and lessons learned," 2017.

[16] J. Fruhlinger, "Marriott data breach FAQ: How did it happen and what was the impact?," CSO, 11 February 2020. [Online]. Available: https://www.csoonline.com/article/567795/marriott-data-breach-faq-how-did-it-happen-and-what-was-the-impact.html. [Accessed 29 August 2023].