Demystifying security in the software lifecycle: From threats to best practices

Qasem Abu Al-Haija

Departmnet of Cybersecurity, Jordan University of Science and Technology, Jordan

Abstract

The quickly changing threat landscape has highlighted the necessity of integrating strong security measures into each stage of the software development lifecycle (SDLC). In this editorial, I reflect on exploring the best practices, obstacles, and techniques for developing secure software. This editorial provides a comprehensive framework for developers seeking to enhance the resilience of their software products against emerging threats by closely examining contemporary approaches and technologies. This editorial examines the role of security testing and code review in enhancing software resilience, identifying and assessing prevalent vulnerabilities, and providing practical mitigation strategies to address these issues. This research aims to develop a standardized method for producing secure software, enabling businesses to create and implement software confidently.

Keywords: cybersecurity, software development, threat modeling, secure coding

1. Introduction

Secure software development has become increasingly essential as cyber threats become more sophisticated and frequent. Conventional development approaches often prioritize functionality and performance over security, resulting in software flaws that can be exploited with severe consequences. These include threats to national security, financial loss, and damage to one's reputation. It is not enough to increase cybersecurity investment, which is expected to reach \$174.7 billion by 2024; security needs to be integrated into every step of the Software Development Lifecycle (SDLC) [1]. From the beginning of development, dangers are handled using a proactive, security-first approach. To facilitate this, the Secure Development Lifecycle (SDL) framework (shown in Figure 1) integrates security into every stage, from requirements and design to coding and testing [2]. To identify system vulnerabilities and create mitigation techniques, threat modeling—specifically, the STRIDE methodology—is essential [3]. Developers can visualize these dangers with the help of tools like the Microsoft Threat Modeling Tool.

Secure coding techniques are just as important. Developers can steer clear of typical security hazards by adhering to guidelines like the CERT Secure Coding guidelines [4]. Static code analysis is aided by tools such as SonarQube and Coverity, which help identify vulnerabilities before they are released. Automated security testing in modern CI/CD environments advances security deployment. Approaches such as Software Composition Analysis (SCA), Static Application Security

Testing (SAST), and Dynamic Application Security Testing (DAST) afford incessant vulnerability assessment across the SDLC. Best practices for software risk management comprise Security by Design, regular training, penetration testing, and secure development environments. In this editorial, I reflect on presenting an inclusive approach to integrating security into modern software engineering, providing programmers with the necessary tools and resources to develop robust and secure systems.

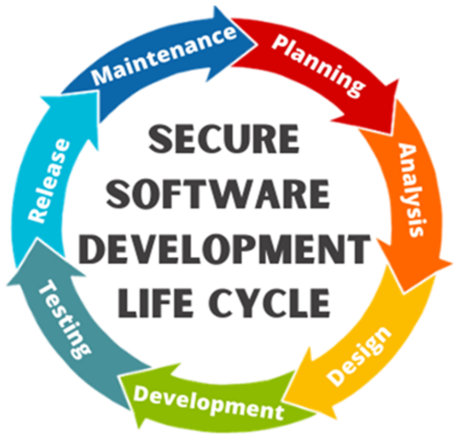

Fig. 1. Secure software development life cycle

To provide context for the recent state of secure software development, Table 1 summarizes the key research findings and best practices in several critical areas, such as automated security testing, risk management, secure coding, threat modeling, and secure SDLC frameworks. The perceptions and methodologies presented in this editorial are grounded in this integrated perspective, which also highlights the interdisciplinary efforts that encompass the field.

Table 1. Summary of recent research in secure software development, highlighting key areas, tools, and representative studies

Topic	Key Contributions	Tools/Frameworks	References
Secure SDLC Frameworks	Integration of Security into all development phases	Microsoft SDL, OWASP ASV5	[2, 3]
Threat Modeling	Structured methodologies for identifying threats; accessibility through automation	STRIDE, Microsoft Threat Modeling Tool	[4, 5]
Secure Coding Practices	Guidelines to prevent coding vulnerabilities across languages	CERT Secure Coding Standards, SonarQube, Coverity	[6, 7]
Automated Security Testing	Continuous testing during CI/CD to detect vulnerabilities in code and dependencies	SAST, DAST, SCA	[8]
Common Vulnerabilities	Identification and mitigation of critical security flaws	OWASP Top Ten	[9, 10]
Risk Management	Quantitative risk analysis and integration in Agile/DevOps environments	FAIR model	[11, 12]
Security by Design	Embedding security awareness and strategies from the beginning of development	Industry case studies	[13, 14]
Developer Training	Continuous education reduces security incidents; gamified modules improve retention.	Interactive training platforms	[15, 16]
Secure Development Environments	Enhancing security through isolation, access controls, and modern container technologies	Containerization, virtualization	[17, 18]

2. Secure software development strategies

Integrating security into each phase of the SDLC involves using secure software development strategies. Here, we emphasize three essential techniques that can be integrated to reduce vulnerabilities and improve software robustness:

• *Threat modeling:* One proactive technique for spotting and reducing possible risks early in the process is threat modeling. Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege are the categories into which the commonly used STRIDE model divides threats [19]. System architecture definition, STRIDE threat identification, mitigation strategy implementation, and model validation are all steps in the process. This process is aided by the Microsoft Threat Modeling Tool, which identifies risks and suggests defenses, thereby significantly reducing vulnerabilities.

• *Secure coding practices:* Secure coding techniques aim to avoid security vulnerabilities when writing code. The OWASP and CERT guidelines are well known for encouraging secure development. Important procedures include secure error handling, strong authentication and access control, data encryption, secure communications, and input validation to stop injection attacks. By checking code for vulnerabilities and offering feedback as it is being developed, tools such as SonarQube and Coverity help enforce these standards [20]. These tools support proactive risk reduction and maintenance of coding standards when incorporated into CI/CD pipelines.

• *Automated security testing:* This is essentially used to help ensure unceasing reassurance, especially in agile and DevOps environments. It comprises Software Composition Analysis (SCA) for recognizing risks in third-party libraries, Static Application Security Testing (SAST) for inspecting source code, and Dynamic Application Security Testing (DAST) for discovering runtime vulnerabilities. Tools like Snyk, OWASP ZAP, and Checkmarx are commonly used for these tasks. Placing these into CI/CD pipelines empowers real-time discovery and remediation, advancing security. According to a NIST investigation, automated security testing methodologies can decrease vulnerabilities by up to 60% [8].

Integrating the tactics above can provide a comprehensive and proactive framework for producing secure software conforming to regulatory compliance requirements and contemporary development approaches.

3. Software development risks

The software development and deployment lifecycle has risks, including vulnerabilities, risk assessment techniques, and mitigation tactics. This part examines these components to provide robust and secure software systems.

Common vulnerabilities are persistent threats exploited by attackers to compromise systems. The OWASP Top Ten (2021) identifies the most critical web application security risks [12]. Notable examples include:

• SQL Injection, where malicious input alters database queries, leading to data breaches or system compromise. It can be prevented with parameterized queries and input validation.

• Cross-site scripting (XSS) injects malicious scripts into web pages, enabling session hijacking or redirection. Countermeasures include output encoding and implementing Content Security Policies.

• Buffer Overflows occur when excessive data overwrites memory, potentially leading to crashes or code execution. Memory-safe language use and bounds checking are key defenses.

Risk assessment is essential for identifying threats and prioritizing responses. It typically follows five steps: asset identification, threat and vulnerability analysis, risk evaluation, prioritization, and mitigation planning. There are two main approaches:

• Qualitative assessments rely on subjective analysis using risk matrices to categorize threats based on likelihood and impact.

• Quantitative assessments, such as the FAIR (Factor Analysis of Information Risk) model, use numerical data to estimate risks, focusing on potential financial impact by evaluating threat frequency and loss magnitude.

Mitigation strategies aim to reduce risk exposure and improve system security. Key practices include:

• *Security Patches:* Regular updates from the vendor address known vulnerabilities. Automated patch management ensures timely deployment.

• *Code Reviews:* Manual or tool-assisted reviews (e.g., using SonarQube) help detect security flaws and enforce coding standards.

• *Penetration Testing:* Simulated attacks expose vulnerabilities from an attacker's perspective, offering insights beyond automated scans. Tests can be conducted internally or by third parties.

By identifying vulnerabilities, assessing risks methodically, and applying targeted mitigation strategies, organizations can enhance the security of their software systems against evolving threats.

4. Best practice

Best Software Development practices are critical to ensuring the security and resilience of software systems. Security by Design, Continuous Training and Awareness, and Secure Development Environments are key best practices for ensuring security.

• *Security by design:* The key readymade is embedding security into every phase of software development, from the initial concept to the final deployment. This ensures security isn't a tacked-on feature but an inherent element of the entire process. This involves crucial practices, such as architecting secure systems using established security principles, including least privilege and secure defaults [21], proactively conducting threat modeling and risk assessments throughout the design stages, and consistently adhering to secure coding guidelines, including input validation and secure authentication protocols. Furthermore, routine code inspections and applying static and dynamic analysis for security testing are crucial for detecting vulnerabilities early. Evidence from case studies demonstrates that this "Security by Design" approach, by identifying vulnerabilities early, can significantly minimize security incidents and overall development expenditures [22].

• *Continuous training and awareness:* To make sure developers are up-to-speed on the latest cybersecurity threats, incessant training and awareness are key. By training regularly, they can stay informed about new weaknesses and the recommended methods for addressing them. Activities such as taking online courses, obtaining certifications, attending industry conferences, and reading security blogs are all helpful for professional development and learning. Still, training programs should focus on practical aspects, such as writing secure code, understanding how to model threats, and becoming comfortable with security technology. There's an example where consistent training led to a significant 60% drop in security errors found during code review [23].

• *Secure development environments:* The integrity of the development process profoundly relies on secure environments. Vigorous access controls, such as role-based access and multi-factor authentication, are essential for restricting access to critical resources to authorized personnel alone [24]. Embedding automated vulnerability detection and security policy enforcement directly within coding environments is also paramount. Furthermore, establishing isolated testing environments serves a vital purpose: it ensures that security evaluations do not disrupt live systems, thereby preventing unintended data breaches and enabling the secure validation of new functionalities. Concrete examples illustrate the tangible benefits of secure development environments, with one organization documenting a 50% reduction in code tampering incidents and unauthorized access attempts following the implementation of these environments.

5. Conclusion and remarks

In this editorial, I reflect on the critical importance of a comprehensive framework for secure software development, which inherently integrates threat modeling, rigorous secure coding practices, & systematic automated security testing. These interconnected processes are principal in proactively identifying & mitigating exploitable vulnerabilities in production through early detection and resolution. Future research should prioritize the development & experiential validation of progressive automated security analysis tools, the iterative refinement of threat modeling methodologies to enhance their practical utility, & the fostering of stronger collaborative ties between security specialists & software development professionals.

6. Future outlook

Looking ahead, the convergence of artificial intelligence, secure software engineering, and data-centric computing will fundamentally reshape how digital systems are designed and trusted. Future secure development paradigms will increasingly rely on automation, intelligent threat modeling, and continuous security validation embedded directly into development pipelines. As systems become more autonomous and interconnected, especially with the rise of AI-native applications, cloud-native architectures, and cyber-physical systems, security can no longer remain reactive. Instead, proactive, adaptive, and intelligence-driven security frameworks must become the default. Emerging directions such as AI-assisted secure coding, autonomous vulnerability discovery, explainable security analytics, and quantum-resilient design principles are expected to define the next generation of secure software ecosystems.

7. Call to researchers

There is a growing need for interdisciplinary collaboration to address the evolving complexity of secure software development in the age of intelligent systems. Researchers are encouraged to move beyond isolated technical solutions and contribute toward holistic, trustworthy, and human-centered security frameworks. This includes advancing explainable and trustworthy AI for secure development, scalable automated security validation, secure-by-design engineering methodologies, and real-world validation through industry-driven datasets and benchmarks. By fostering stronger collaboration between academia, industry, and policymakers, the research community can play a decisive role in building resilient digital infrastructures that are not only innovative but also secure, ethical, and sustainable.

References

[1] International Data Corporation (IDC), "Worldwide Spending on Security Solutions Forecast to Reach $174.7 Billion in 2024," 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS46792520.

[2] Microsoft, "Security Development Lifecycle (SDL)," 2021. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/.

[3] Shostack, A. (2014). *Threat Modeling: Designing for Security*. John wiley & sons.

[4] CERT, "CERT Secure Coding Standards," 2020. [Online]. Available: https://www.securecoding.cert.org.

[5] OWASP, "OWASP Application Security Verification Standard," 2021. [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/.

[6] Microsoft, "Microsoft Threat Modeling Tool," 2020. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/mtmtool/.

[7] SonarSource, "SonarQube", 2021. [Online]. Available: https://www.sonarqube.org/.

[8] OWASP, "OWASP CI/CD Security Guidelines," 2021. [Online]. Available: https://www.owasp.org/index.php/CI/CD_Security.

[9] Veracode, "Static Application Security Testing (SAST)," 2021. [Online]. Available: https://www.veracode.com/products/static-analysis-sast.

[10] OWASP, "OWASP Top Ten," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/.

[11] FAIR Institute, "Factor Analysis of Information Risk (FAIR)," 2021. [Online]. Available: https://www.fairinstitute.org/fair.

[12] Hauck, Jean Carlo Rossa, and Marcel Vieira. "Towards a guide for risk management integration in agile software projects." European Conference on Software Process Improvement. Cham: Springer International Publishing, 2021.

[13] Schneier, B. (2015). *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*. WW Norton & Company.

[14] Baca, Dejan, and Kai Petersen. "Prioritizing countermeasures through the countermeasure method for software security (CM-Sec)." International Conference on Product Focused Software Process Improvement. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[15] van Steen, Tommy, and Julia RA Deeleman. "Successful gamification of cybersecurity training." Cyberpsychology, Behavior, and Social Networking 24.9 (2021): 593-598.

[16] IBM, "Setting Up Secure Development Environments," 2021. [Online]. Available: https://www.ibm.com/security/development-environments.

[17] Docker, "Containerization and Security," 2021. [Online]. Available: https://www.docker.com/products/containerization-security.

[18] Swiderski, F., & Snyder, W. (2004). *Threat Modeling*. Microsoft Press.

[19] Synopsys, "Coverity Static Analysis," 2021. [Online]. Available: https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html.

[20] National Institute of Standards and Technology (NIST), "The Role of Automated Security Testing in Reducing Vulnerabilities," NIST Technical Report, 2021.

[21] McGraw, G. (2012). Software security: Building security in. *Datenschutz Und Datensicherheit-Dud, 36*(9), 662-665.

[22] Varadam, Deepak, et al. "Evolution of Data Security in E-Commerce." Utilizing AI in Network and Mobile Security for Threat Detection and Prevention. IGI Global Scientific Publishing, 2025. 281-304.

[23] Thomas, Tyler W., et al. "Security during application development: An application security expert perspective." Proceedings of the 2018 CHI conference on human factors in computing systems. 2018.

[24] National Institute of Standards and Technology (NIST), "Access Control Practices for Secure Development," NIST Technical Report.

How to cite this article: Q. A.Al-Haija (2026). Demystifying security in the software lifecycle: From threats to best practices . *Bulletin of Computer and Data Sciences*, 7(1), 10-16. DOI: 10.71448/bcds2671-2

Published: 24/02/2026

Copyright: © 2026 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See https://creativecommons.org/licenses/by/4.0/.

AND DATA SCIENCES BULLETIN OF COMPUTER *Bulletin of Computer and Data Sciences* is a peer-reviewed open access journal.