Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Secure Software Development Practices for Information Security

Nayan Goel
Senior Application Security Engineer, Upgrade, Inc., Foster City, California, USA.

ARTICLE INFO

Received: 02 Feb 2022
Revised: 12 March 2022
Accepted: 22 March 2022

ABSTRACT

The importance of secure software development in information system security against a more complex and changing threat environment is significant. With software applications playing a crucial role in sustaining key organizational and societal operations, inconsistencies that may be introduced in the development process may reveal sensitive data, spoil services, and destroy trust. The issue under discussion is secure software development as a systematic method of incorporating security principles, controls, and risk management activities throughout the software development life cycle. It highlights secure-by-design techniques, threat modeling, secure coding guidelines, security testing, and ongoing monitoring as important elements in minimizing the use of software in risks to security. The importance of organizational processes, awareness of the developer, and the introduction of security in the current development processes are also emphasized. The secure software development practices can play a role in ensuring information systems confidentiality, integrity and availability by incorporating security thinking based on requirements analysis into deployment and maintenance of information systems thus enhancing the overall information security posture.

Keywords: Secure software development, Information security, Secure coding, Software vulnerabilities, Threat modeling, Security testing, Secure software lifecycle

I. Introduction

The growing use of software-based systems in key industries has only accelerated the anxieties over information security since software vulnerability continues to be a significant source of cyber attacks and data theft. The environments in which modern applications are implemented are very interconnected and a vulnerability that is created during the development stage may jeopardize the confidentiality, integrity, and availability of the information assets. At its most fundamental level, computer security is aimed at keeping systems and information away from unauthorized access, abuse, and sabotage, thus secure software development is not a nice addition but the basic requirement in its formation (Bishop, 2003; Whitman and Mattord, 2009).

SSDPs focus on the systematic approach to integration of security considerations during the entire software development life cycle, which is requirements analysis and design, implementation, testing, deployment, and maintenance. Initial studies also note that design and architectural choices are often the source of many security flaws and not only due to errors in coding, hence the necessity to have systematic approach to secure software design (Fernandez, 2004; McGraw, 2004).Best-practice frameworks and lifecycle models, such as the Security Development Lifecycle, advocate proactive risk identification and mitigation to reduce vulnerabilities before software is released into production environments (Howard & Lipner, 2006; Preuveneers et al., 2008).

Even though well-developed models and guidelines have been put in place, the successful implementation of secure software development practices has not been uniform throughout the organizations. Organizational culture, skill levels of developers, resource limitations, and lack of

alignment between development and security teams remain to be the obstacles to the regular implementation (Kanniah and Mahrin, 2016; Van Wyk and McGraw, 2005). Empirical research goes further to show that effective implementation of SSDPs must involve more than merely technical control, encompassing managerial dedication, process harmonization, and constant enhancement systems (Al-Amin et al., 2018; Boppana, 2019).

It is against this background that secure software development has been a critical approach to enhancing information security in modern computing setup. Organisations can also greatly mitigate software vulnerabilities, build resilience to the changing threats and make sure that information security goals are progressively achieved throughout the software lifecycle by integrating security principles and practices into development processes.

II. Foundations of Secure Software Development

The principles of secure software development lie in the understanding that software security is not a localized technical endeavor but a methodical field that needs to be incorporated throughout the entire development cycle. Secure software development focuses on preventing risks instead of responding to vulnerability exposure, which is implemented proactively, and security concerns are considered at all stages of the conception phase into deployment and maintenance. This view correlates with the general meaning of computer security as security against unauthorized access, interruption, or misuse of information systems to ensure the confidentiality, integrity, and availability of information (Bishop, 2003; Whitman and Mattord, 2009).

A fundamental tenet of a secure software development is the in-secure-by-design principle, which will insist on the addition of security mechanisms to the system architecture and not to add them post-implementation. Secure design techniques promote a developer to consider the possible threats, misuse cases, and capabilities of attackers when modeling systems and planning their architecture (Fernandez, 2004; Preuveneers et al., 2008). This approach reduces structural weaknesses that are often difficult and costly to correct later in the software lifecycle. Closely related is the concept of defense in depth, which promotes layered security controls so that the failure of a single safeguard does not compromise the entire system (McGraw, 2004; Howard & Lipner, 2006).

The other core pillar is ensuring that security principles and standards are used to inform development decisions. The conceptual framework of minimizing attack surfaces and restricting the potential damage of exploiting the platform is offered by principles like least privilege, separation of duties, secure defaulting and fail-safe mechanisms (Whitman and Mattord, 2009; Bishop, 2003). These principles will ensure the abstract security goals are converted into concrete development practices, which will allow uniformity among teams and projects. The studies point out that the ability to follow established security principles enhances the efficiency of secure software development procedures in organizations to a significant extent (Kanniah and Mahrin, 2016; Boppana, 2019).

Critical basis is also risk awareness and threat-oriented thinking. To develop software securely, the software developer must know the manner in which the vulnerability develops due to design errors, coding mistakes, and misconfigurations, and how the vulnerabilities may be used by the attackers. Early risk assessment and threat modeling assist in identifying the vulnerabilities that are most likely to have high impacts and focus on mitigating them according to the business and security impacts they may have (McGraw, 2004; Al-Amin et al., 2018). Such an attitude of risk is a way to bridge the historical

divide between software engineering and information security by achieving organizational security goals with technical controls (Van Wyk and McGraw, 2005).

Lastly, the organizational and human aspect are part and parcel of the secure software development principle. Research has shown that the awareness of developers, the support of the management, and the culture of institutional security have a strong impact on the effective implementation of secure developing practices (Kanniah and Mahrin, 2016; Al-Amin et al., 2018). Security should then be understood as a collective effort between the developers, architects and the security professionals and be implemented through well-organized mechanisms like the Security Development Lifecycle (Howard & Lipner, 2006). Collectively, these background factors provide a strong foundation on the creation of software systems capable of withstanding the changing information security threats as well as facilitating functional and operational needs.

III. Secure Software Development Life Cycle (SSDLC)

The Secure Software Development Life Cycle (SSDLC) extends the traditional software development life cycle by systematically embedding security considerations into every development phase. Rather than treating security as a post-development activity, SSDLC ensures that information security requirements are identified early, implemented consistently, and continuously refined throughout the software's operational lifespan. This lifecycle-oriented approach is widely recognized as essential for reducing vulnerabilities, minimizing remediation costs, and aligning software functionality with organizational security objectives (McGraw, 2004; Howard & Lipner, 2006).

SSDLC is grounded in the principle that software security is a process, not a product. By integrating preventive, detective, and corrective controls across all stages of development, organizations can proactively address threats and manage risks before they materialize into exploitable weaknesses (Van Wyk & McGraw, 2005; Whitman & Mattord, 2009).

3.1 Security Requirements Analysis

The SSDLC begins with the identification and documentation of security requirements alongside functional requirements. At this stage, potential threats, regulatory obligations, and organizational security policies are analyzed to define clear security objectives. Failure to address security at this phase often leads to architectural weaknesses that are costly or impossible to fix later (Bishop, 2003; Kanniah & Mahrin, 2016).

Key activities include stakeholder engagement, asset identification, and preliminary risk assessment to ensure confidentiality, integrity, and availability requirements are explicitly defined (Whitman & Mattord, 2009).

Table 1: Key Activities in Security Requirements Analysis

Activity	Description	Security Contribution
Asset identification	Identification of critical data and system components	Clarifies protection priorities

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Risk assessment	Evaluation of potential threats and impacts	Guides security control selection
Security requirement specification	Documentation of security needs	Prevents ambiguity in later stages
Compliance analysis	Identification of legal and regulatory requirements	Ensures adherence to standards

3.2 Secure Design and Architecture

During the design phase, security requirements are translated into a robust system architecture. Secure design emphasizes principles such as least privilege, defense-in-depth, separation of concerns, and fail-safe defaults (Fernandez, 2004; Preuveneers et al., 2008). Threat modeling is commonly applied at this stage to identify potential attack vectors and to design appropriate countermeasures.

Architectural decisions made during this phase have a long-term impact on the system's security posture. Secure design patterns and reference architectures help developers avoid recurring vulnerabilities and enforce consistent security controls across components (McGraw, 2004; Boppana, 2019).

3.3 Secure Implementation and Coding

The implementation phase focuses on translating secure designs into reliable and resilient code. Secure coding standards, guidelines, and frameworks are applied to prevent common vulnerabilities such as injection flaws, improper authentication, and insecure data handling (Howard & Lipner, 2006).

Developers play a critical role at this stage, making security awareness and adherence to best practices essential success factors. Studies indicate that organizational support and developer training significantly influence the effective adoption of secure coding practices (Al-Amin et al., 2018; Kanniah & Mahrin, 2016).

Table 2: Secure Implementation Practices and Objectives

Practice	**Description**	**Security Objective**
Secure coding standards	Use of approved coding guidelines	Reduces implementation flaws
Input validation	Validation of all external inputs	Prevents injection attacks
Error and exception handling	Controlled error reporting	Avoids information leakage
Secure use of APIs and libraries	Proper configuration of dependencies	Limits inherited vulnerabilities

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.4 Security Testing and Verification

Security testing ensures that implemented controls function as intended and that vulnerabilities are identified before deployment. This phase complements functional testing by focusing specifically on security weaknesses through techniques such as code reviews, static analysis, dynamic testing, and penetration testing (McGraw, 2004; Preuveneers et al., 2008).

Security verification is iterative and should be repeated whenever code changes occur. Integrating automated security testing into development workflows improves consistency and supports continuous risk reduction (Van Wyk & McGraw, 2005).

3.5 Secure Deployment and Maintenance

The final stages of SSDLC address secure deployment, operation, and ongoing maintenance. Secure configuration of environments, controlled release management, and continuous monitoring are essential to preserving software security after deployment (Howard & Lipner, 2006).

As threats evolve, maintenance activities such as patch management, vulnerability reassessment, and incident response become critical. SSDLC therefore extends beyond initial release, reinforcing the notion that secure software development is a continuous and adaptive process rather than a one-time effort (Boppana, 2019; Whitman & Mattord, 2009).

Overall, the Secure Software Development Life Cycle provides a structured and proactive framework for embedding information security into software systems. By integrating security requirements, design principles, implementation controls, testing mechanisms, and maintenance practices, SSDLC significantly enhances the resilience of software against emerging threats while aligning development processes with established information security principles (Al-Amin et al., 2018; Howard & Lipner, 2006).

IV. Secure Coding Practices

Secure coding practices constitute a critical layer of defense in protecting software systems against vulnerabilities that threaten information security. These practices translate security requirements and design decisions into robust implementation techniques that minimize exploitable weaknesses at the code level. Research consistently shows that a significant proportion of security breaches originate from insecure coding, underscoring the need for disciplined, standardized, and repeatable secure coding methods across development teams (McGraw, 2004; Van Wyk & McGraw, 2005).

Secure coding is grounded in established information security principles confidentiality, integrity, and availability and seeks to ensure that software behaves predictably even in the presence of malicious inputs or unexpected operational conditions (Bishop, 2003; Whitman & Mattord, 2009). Rather than treating security as an afterthought, secure coding embeds protection mechanisms directly into application logic, data handling, and error management processes.

4.1 Input Validation and Data Handling

One of the most fundamental secure coding practices is rigorous input validation. Applications must treat all external inputs as untrusted and enforce strict validation rules to prevent injection attacks, buffer overflows, and data corruption. Effective input validation includes type checking, length

validation, format enforcement, and boundary checking (Howard & Lipner, 2006). Secure data handling further requires proper output encoding to prevent cross-site scripting and similar attacks, ensuring that data is interpreted only in its intended context (Preuveneers et al., 2008).

4.2 Authentication and Authorization Controls

Secure coding practices mandate the correct implementation of authentication and authorization mechanisms to ensure that only legitimate users gain access to protected resources. Authentication logic should be resistant to brute-force attacks, credential reuse, and improper session handling, while authorization checks must be enforced consistently at every access point (Fernandez, 2004). Failure to embed authorization checks directly within application logic often leads to privilege escalation vulnerabilities (McGraw, 2004).

4.3 Error Handling and Logging

Improper error handling can inadvertently disclose sensitive system information that attackers may exploit. Secure coding requires developers to design error messages that are informative for administrators but opaque to end users. Internally, detailed logs should be maintained to support monitoring, forensic analysis, and incident response, while externally exposed messages should avoid revealing implementation details (Howard & Lipner, 2006; Boppana, 2019).

4.4 Secure Session and State Management

Managing user sessions securely is essential to preventing session hijacking, fixation, and replay attacks. Secure coding practices include generating unpredictable session identifiers, enforcing session expiration, and protecting session data during transmission and storage. These measures help preserve user identity integrity and reduce the attack surface associated with long-lived or improperly managed sessions (Whitman & Mattord, 2009).

4.5 Prevention of Common Software Vulnerabilities

Secure coding guidelines aim to mitigate widely recognized classes of vulnerabilities through standardized defensive techniques. Adopting secure coding standards and patterns enables developers to proactively address known weaknesses and maintain consistency across projects (Kanniah & Mahrin, 2016; Al-Amin et al., 2018).

Table 3: Common Software Vulnerabilities and Secure Coding Control

Vulnerability Type	Description	Secure Coding Control
Injection attacks	Malicious input alters program execution	Input validation, parameterized queries
Buffer overflow	Memory overwrite through unchecked input	Bounds checking, safe memory functions
Broken authentication	Weak credential or login handling	Strong authentication logic, hashing
Insecure session handling	Unauthorized session access	Secure tokens, session expiration

Information leakage	Exposure of sensitive system details	Controlled error handling and logging

Source: Adapted from McGraw (2004); Preuveneers et al. (2008)

4.6 Standardization and Developer Responsibility

The effectiveness of secure coding practices depends heavily on organizational commitment and developer competence. Establishing coding standards, conducting peer reviews, and providing targeted security training enhance adherence to secure coding guidelines (Van Wyk & McGraw, 2005). Studies emphasize that secure coding adoption improves significantly when supported by management policies and integrated into everyday development workflows (Al-Amin et al., 2018).

Table 4: Key Secure Coding Practice Categories and Objectives

Practice Category	Primary Objective
Input validation	Prevent malicious or malformed data entry
Authentication controls	Ensure verified user identity
Authorization enforcement	Restrict access based on privileges
Error handling	Avoid disclosure of sensitive information
Logging and monitoring	Support detection and response to incidents

Source: Fernandez (2004); Howard & Lipner (2006)

Overall, secure coding practices serve as a practical bridge between secure software design and operational information security. By systematically applying these practices, organizations can significantly reduce vulnerability exposure and enhance the resilience of software systems against evolving threats (McGraw, 2004; Kanniah & Mahrin, 2016).

V. Threat Modeling and Risk Assessment

Threat modeling and risk assessment form a fundamental part of the safe software development habit that offers a systematic approach of identification, examination, and reduction of security dangers in advance of them turning into exploitable vulnerabilities. Threat modeling integrates proactive thinking concerning adversaries, assets, and attack vectors into the development process instead of viewing security as a reactive exercise, making overall information security posture strong (McGraw, 2004; Van Wyk and McGraw, 2005).

In the threat modeling process, system characterization is the first step in which critical assets, system boundaries, trust zones, and data flows are outlined. This is done to make sure that the developers are aware of what is to be safeguarded and where security measures are the most sensitive (Fernandez, 2004). The typical approaches focus on breaking down the system into components and interactions and thus permitting structured discovery of threats that might exist such as unauthorized access,

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

tampering of data, denial of service, and escalation of privileges (Bishop, 2003; Preuveneers et al., 2008).

Risk assessment is the next step in which the probability and possible effects of the threats are assessed after the threats have been identified. This technique ranks risks in order of severity and allows the development teams to spend scarce resources on the most serious security risks (Whitman and Mattord, 2009). Risk assessment assists in making informed decisions by balancing technical feasibility, business consequences and security needs, an aspect that has been found to have a significant impact in the successful implementation of secure practices in software development (Kanniah and Mahrin, 2016; Al-Amin et al., 2018).

Another advantage of threat modeling is that it helps in the decision of the right security controls at the design stage. Through mapping the risks likely to be faced by the system that developers need to take measures to mitigate, developers can reduce the size of attack surfaces at an early stage when mitigation becomes less expensive and more efficient (Howard and Lipner, 2006; Fernandez, 2004). It is consistent with best-practice advice which focuses on integrating security activities with software development life cycle early on (Preuveneers et al., 2008; Boppana, 2019).

Notably, threat modelling and risk assessment are not one time efforts.As systems evolve through new features, integrations, and deployment environments, risks must be continuously reassessed. Iterative threat modeling supports agile and DevOps workflows by enabling continuous security evaluation alongside functional changes, thereby bridging the traditional gap between software development and information security functions (Van Wyk & McGraw, 2005; Al-Amin et al., 2018).

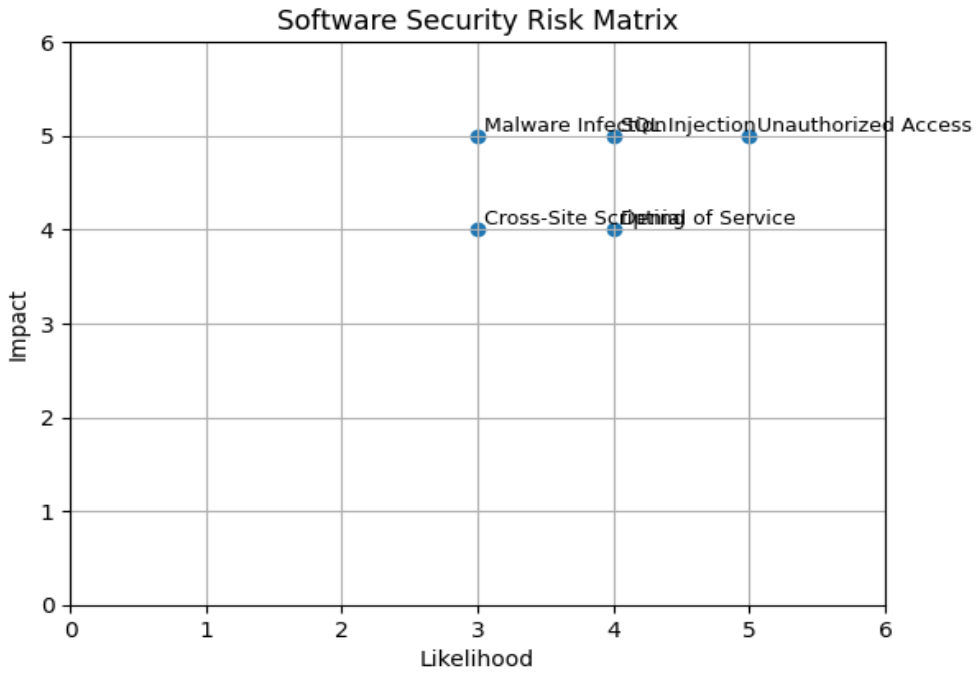

Fig 1: This risk matrix visualizes identified software security threats by mapping their estimated likelihood of occurrence against potential impact, enabling clear prioritization of mitigation and control efforts.

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Threat modeling and risk assessment provide a disciplined framework for anticipating and mitigating security risks throughout software development. By systematically identifying threats, evaluating risks, and aligning mitigation strategies with secure design principles, organizations can significantly enhance the resilience of software systems and protect critical information assets (McGraw, 2004; Whitman & Mattord, 2009).

VI. Security Testing and Code Review

Security testing and code review constitute a critical control layer within secure software development practices, serving as mechanisms for identifying, validating, and mitigating vulnerabilities before software deployment. These practices bridge the gap between theoretical secure design principles and their correct implementation in real-world systems, thereby reinforcing information security objectives such as confidentiality, integrity, and availability (Van Wyk & McGraw, 2005; Whitman & Mattord, 2009).

6.1 Role of Security Testing in Software Assurance

Security testing focuses on evaluating software behavior under both expected and adversarial conditions. Unlike functional testing, which validates correctness, security testing assesses resistance to misuse, attack, and failure. It is designed to uncover flaws that could be exploited to compromise system assets, making it an indispensable component of secure software development life cycles (McGraw, 2004; Howard & Lipner, 2006).

Effective security testing integrates multiple techniques, including static, dynamic, and manual approaches, to achieve broader coverage of potential vulnerabilities. Research emphasizes that organizations adopting layered security testing practices experience significantly improved detection of security defects compared to ad hoc testing approaches (Al-Amin et al., 2018; Boppana, 2019).

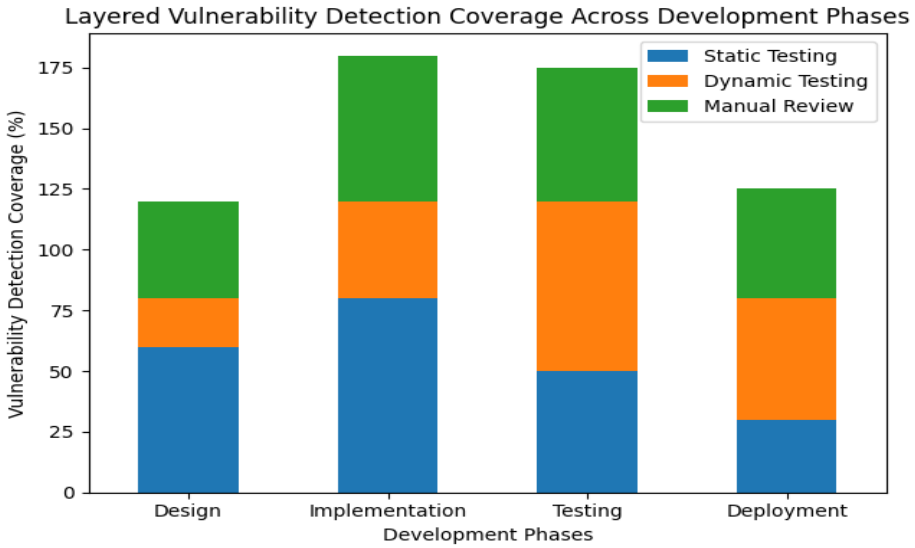

Fig 2: The layered bars indicate the relative contribution of static testing, dynamic testing, and manual review to vulnerability detection at each stage of the software development lifecycle, highlighting how different assurance techniques complement one another across phases.

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

6.2 Static Application Security Testing (SAST)

Static application security testing examines source code, bytecode, or binaries without executing the application. SAST tools are effective in identifying coding flaws such as buffer overflows, improper input validation, insecure cryptographic usage, and logic errors early in the development process (Howard & Lipner, 2006; Preuveneers et al., 2008).

The primary advantage of SAST lies in its early integration into development workflows, allowing developers to remediate vulnerabilities before they propagate into later stages. However, studies note challenges such as false positives and the need for skilled interpretation of results to ensure effectiveness (Kanniah & Mahrin, 2016; Al-Amin et al., 2018).

6.3 Dynamic Application Security Testing (DAST)

Dynamic application security testing evaluates applications during runtime, simulating real attack scenarios against deployed or staging environments. DAST is particularly effective in identifying vulnerabilities related to authentication, session management, access control, and configuration errors that are not visible through static analysis (McGraw, 2004; Bishop, 2003).

While DAST provides insight into application behavior from an attacker's perspective, its effectiveness depends on realistic test environments and comprehensive test case design. Consequently, best practices recommend combining DAST with static techniques to compensate for individual limitations (Howard & Lipner, 2006; Boppana, 2019).

6.4 Manual Code Review and Secure Peer Review

Manual code review remains one of the most effective methods for identifying complex security flaws, particularly those involving business logic, architectural weaknesses, and improper trust assumptions. Secure peer reviews enable experienced developers and security professionals to evaluate code against established security standards and design principles (Fernandez, 2004; Van Wyk & McGraw, 2005).

Manual reviews are especially valuable in validating whether secure design patterns are correctly implemented and aligned with system requirements. Although resource-intensive, literature consistently highlights manual review as a high-impact practice when integrated with automated tools (Preuveneers et al., 2008; Whitman & Mattord, 2009).

6.5 Penetration Testing and Vulnerability Validation

Penetration testing complements code-level testing by simulating real-world attacks on deployed systems. It validates whether identified vulnerabilities are exploitable and assesses the potential impact on organizational assets. This form of testing supports risk-based decision-making by prioritizing remediation efforts based on exploitability and severity (Bishop, 2003; McGraw, 2004).

Penetration testing is most effective when conducted iteratively and aligned with secure development processes rather than treated as a one-time compliance activity (Howard & Lipner, 2006; Al-Amin et al., 2018).

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

6.6 Comparative Overview of Security Testing Techniques

The integration of multiple security testing methods enhances overall software resilience by addressing vulnerabilities from complementary perspectives. Table 1 summarizes key security testing techniques and their primary strengths.

Table 5: Comparison of Security Testing and Code Review Techniques

Technique	Primary Focus	Strengths	Limitations
Static Application Security Testing (SAST)	Source code analysis	Early detection of coding flaws	High false positives
Dynamic Application Security Testing (DAST)	Runtime behavior	Identifies exploitable vulnerabilities	Limited code visibility
Manual Code Review	Logic and design validation	Detects complex flaws	Resource-intensive
Penetration Testing	Attack simulation	Realistic risk assessment	Requires skilled testers

6.7 Integration into Secure Development Processes

For security testing and code review to be effective, they must be systematically embedded within development workflows. The security development lifecycle emphasizes continuous testing, automated enforcement, and feedback mechanisms that support ongoing improvement (Howard & Lipner, 2006; Van Wyk & McGraw, 2005).

Organizations that institutionalize these practices demonstrate stronger alignment between software engineering and information security goals, resulting in reduced exposure to software-based threats and improved trust in digital systems (Kanniah & Mahrin, 2016; Whitman & Mattord, 2009).

Conclusion

Secure software development practices are still considered as a major cornerstone of successful information security practices because the vulnerabilities in software are still one of the main sources

of security breach. Applying the concepts of security all over the software development life cycle instead of viewing them as an addition to the software as it matures will go a long way in minimizing the exposure of the system to threats and will result in more resilience. The conceptual basis of these practices is provided by the foundational principles of confidentiality, integrity, and availability; the controlled methodologies like the secure design, threat modeling, and ongoing testing can bring the theory into practice (Bishop, 2003; Whitman and Mattord, 2009).

According to the literature, it is always stressed that secure software development is not solely a matter of technical mechanisms but also organizational commitment, awareness of the developers and process maturity. The effectiveness of secure coding standards, regular code reviews, and thorough security testing can be achieved only with the help of clear policies, appropriate training, and cross-functional work of the development and security teams (Kanniah and Mahrin, 2016; Van Wyk and McGraw, 2005). Security development lifecycle models show that the integration of security checkpoints at every stage of development enhances the vulnerability identification and mitigation stages prior to deployment (Howard and Lipner, 2006; McGraw, 2004).

In addition, secure software development practices are a dynamic process that should adjust to the new threats, new technologies and more complex systems. Empirical research demonstrates that organizations that integrate the best practices like risk-based decisions in security, secure design patterns and continuous improvement mechanisms are well placed to attain the sustainable outcomes in information security (Preuveneers et al., 2008; Fernandez, 2004; Boppana, 2019). Finally, secure software development is not an event but a continual, organizational wide practice that builds trust and safeguards information resources and promotes the reliability of systems over time (Al-Amin et al., 2018).

References

[1] Al-Amin, S., Ajmeri, N., Du, H., Berglund, E. Z., & Singh, M. P. (2018). Toward effective adoption of secure software development practices. *Simulation Modelling Practice and Theory*, *85*, 33-46.
[2] Kanniah, S. L., & Mahrin, M. N. R. (2016). A review on factors influencing implementation of secure software development practices. *International Journal of Computer and Systems Engineering*, *10*(8), 3032-3039.
[3] Preuveneers, D., Berbers, Y., & Bhatti, G. (2008, December). Best practices for software security: An overview. In *2008 IEEE International Multitopic Conference* (pp. 169-173). IEEE.
[4] Fernandez, E. B. (2004, June). A Methodology for Secure Software Design. In *Software Engineering Research and Practice* (pp. 130-136).
[5] McGraw, G. (2004). Software security. *IEEE Security & Privacy*, *2*(2), 80-83.
[6] Boppana, V. (2019). Secure Practices in Software Development. *Global Research Review in Business and Economics (GRRBE)*, *10*.
[7] Van Wyk, K. R., & McGraw, G. (2005). Bridging the gap between software development and information security. *IEEE Security & Privacy*, *3*(5), 75-79.
[8] Whitman, M. E., & Mattord, H. J. (2009). *Principles of information security* (p. 656). Boston, MA: Thomson Course Technology.
[9] Howard, M., & Lipner, S. (2006). *The security development lifecycle* (Vol. 8). Redmond: Microsoft Press.
[10] Bishop, M. (2003). What is computer security?. *IEEE Security & Privacy*, *1*(1), 67-69.