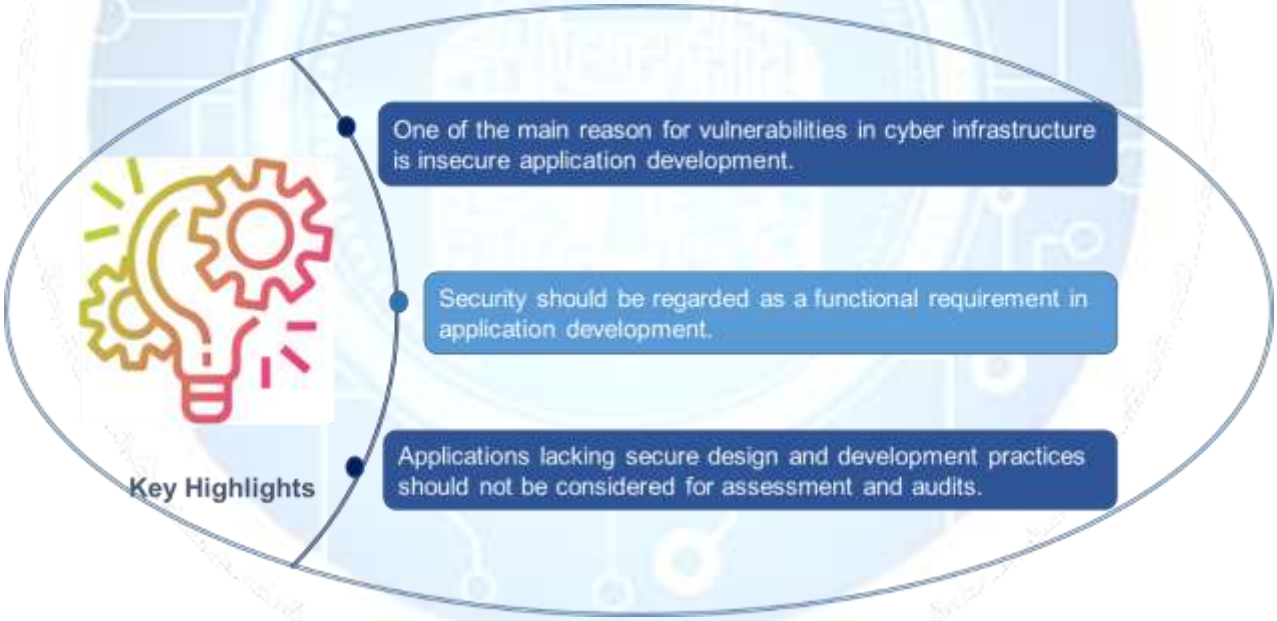
सत्यमेव जयते

Guidelines for Secure Application Design, Development, Implementation & Operations



Issued by:

Indian Computer Emergency Response Team (CERT-In)
Ministry of Electronics and Information Technology
Government of India

Table of Contents

1. Introduction and Purpose

One of the key reason for vulnerabilities in the applications are lack of secure design, development, implementation, and operations. Relying solely on post-development audits for security is inadequate. Instead, security must be an inherent and integral aspect, seamlessly integrated into the application's design and development lifecycle. Organization should incorporate secured application development practices and application owners should ask for adherence to the best practices highlighted in this document and should not only rely on the post audit. By adhering to these guidelines, applications can be developed with built-in security measures making it difficult target for security breaches and exploitation.

The guidelines have been divided into four phases, as depicted in Figure 1 below. After the adoption of secure application design and development guidelines, the application can undergo both source-code review and black-box testing by CERT-In empaneled auditing organization to identify any lapses / vulnerabilities in implementation of the security practices in the application. Key findings and recommendations in the guidelines are mined from the field data analysis of audits conducted by CERT-In empaneled auditing organizations.

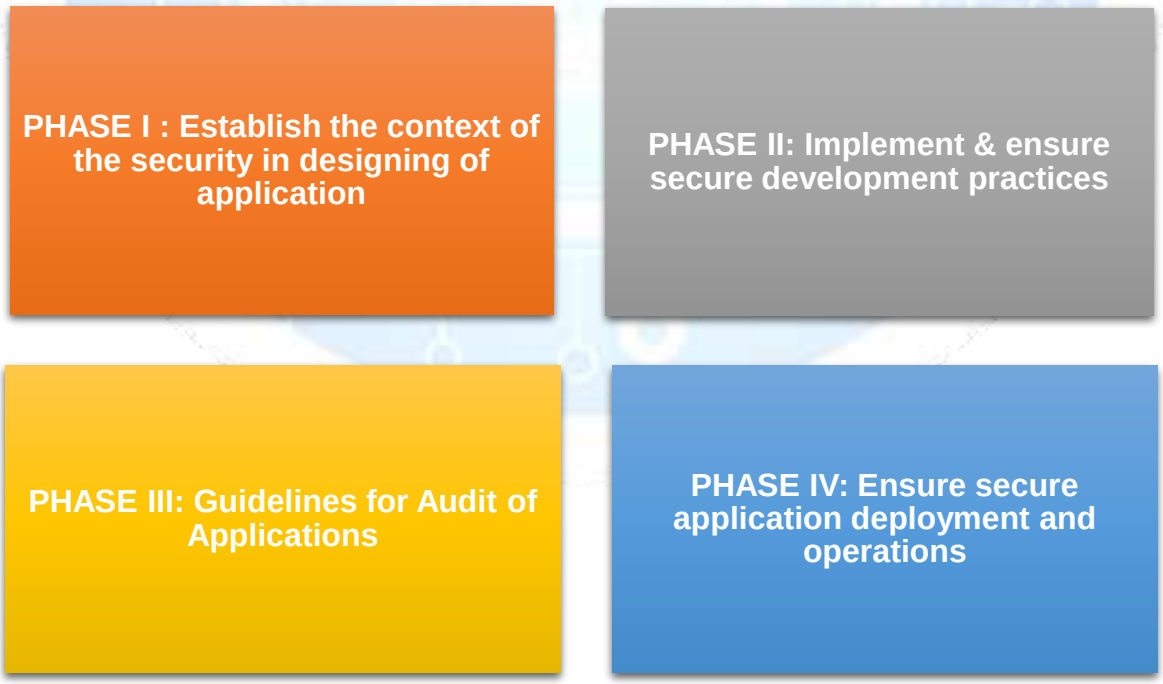


Figure 1: Phases of the guideline document

2. Applicability and Scope

This guideline has been issued by Indian Computer Emergency Response Team (CERT-In) for the entities engaged in developing or outsourcing application development (especially for government sector entities). The prime objective of this guideline is to establish a firm and robust application security baseline in application development lifecycle. The concept of application security pertains to the comprehensive approach for safeguarding all dimensions of an application. This includes its design, development, deployment, and maintenance phases. Hence, the purposeful inclusion of application security is driven by the intention to offer guidance to the target audience, aiming to foster the development of secure applications. This approach is crucial for ensuring the application's security right from the initial phase and progressively strengthening every phases of application development lifecycle.

The secure application development practices outlined in this document have been crafted to enable organizations to customize them according to their specific requirements and seamlessly integrate them into their application lifecycle right from the outset of an application development project. The establishment of context of the security in design process is addressed which underscores all security considerations, encompassing not only secure application design but also secure architectural design, by considering the environment to which the application will be integrated, and outlining strategies to ensure its comprehensive security. Furthermore, this document expounds upon how secure application deployment, operational maintenance, and response protocols can be structured to ensure the continued security of the application. Moreover, it offers insights into making security patches and updates more accessible for the organization, facilitating the seamless integration of frequent security practices over time. This document also emphasizes that applications lacking secure design and development practices are not suitable for assessments and audits. Auditee organizations and auditor organizations must confirm that application is designed & developed with secure practice prior to commencing any assessment.

3. PHASE – I: Establish the Context of the Security in Designing of Application

3.1 Security by Design Approach: It refers to an approach of incorporating security measures and considerations into the design and architecture of a system or application from early stages of the development process. It emphasizes the proactive integration of security controls, mechanisms, and safeguards at the foundational level of the system, rather than as an afterthought or add-on. The goal of security by design is to create systems that are inherently secure, resilient, and resistant to security threats, vulnerabilities, and attacks.

3.2 Adoption of Secure Software Development Life Cycle (SDLC): Secure SDLC is a methodological approach that integrates security practices throughout the software development life cycle. It enables organization to incorporate security as a key component of the development process ensuring compliance with global standards, build software with robust security measures, reducing the likelihood of security breaches, protecting sensitive data, and delivering secure & reliable software. Secure Software Development Life Cycle (SDLC) encompasses various models and frameworks:

a) "Microsoft Secure Development Lifecycle (SDL)" is a widely known and adopted SDLC framework with seven phases.

b) "Open Web Application Security Project (OWASP) Software Assurance Maturity Model (SAMM)" helps build mature software security programs with four levels and multiple security practices.

c) "Agile Secure Development Lifecycle" integrates security practices within agile methodologies, including security grooming, security testing, continuous integration & deployment, security feedback loop.

d) "NIST Secure Software Development Framework (SSDF)" is a comprehensive guide for developing secure software.

3.3 Engagement of security trained designers and developers in the application development: Designers and developers involved in application development

must possess a comprehensive understanding of the cyber security fundamentals and should possess practical knowledge of the security principles governing secure application development. This involves identifying the underlying reasons for weaknesses & vulnerabilities in the application.

4. PHASE – II: Implement & Ensure Secure Development Practices

4.1 Authentication, Authorization & Session Management: The implementation of data protection and privacy measures requires the application of diverse methodologies to ensure responsible and secure information management. It is imperative to incorporate coordinated actions that integrate secure authentication, session management, and access control, all in harmony with the fundamental principles of safeguarding data privacy and protection. Secure authentication and authorization mechanisms must be seamlessly integrated to strengthen against unauthorized access and to uphold the integrity of both systems and application applications. Authentication involves the meticulous validation of user or system identities, while authorization governs the judicious allowance or denial of resource access based on user identity. Simultaneously, vigilant session management is essential to prevent session-related vulnerabilities and unauthorized access. This involves systematically generating, administering, and terminating user sessions while implementing concurrent security protocols. Furthermore, access control achieved through the delineation and enforcement of regulatory rules and policies, stands as the cornerstone to ensure that exclusive privileges for specific assets or information within an application are only extended to authorized users.

4.2 Cryptographic Practices: It refers to set of procedures, techniques, and methods used to secure the sensitive information from unauthorized access & data breaches through various mathematical and computational techniques. Cryptographic practices should ensure data is encrypted to make it secure and unreadable (encryption), creates distinct signatures to confirm the validity and

integrity of data (digital signatures) and produce data representations with defined sizes (hashing). The outdated cryptographic techniques and custom implementation should be avoided.

4.3 **Version Control and Change management**: These are set of practices & procedures that are used to manage and control modifications in application configurations and source code. In version control, a system that monitors and controls changes to files or source code in order to facilitate collaboration, maintain a change history, and allow revert / rollback to previous versions for testing and compliance checks. Change management, on the other hand, is the process of planning, analysing, approving, and implementing changes to application or systems. It ensures that changes have been carefully assessed, tested, well-documented, and implemented in a controlled manner.

4.4 **Ensure Secure Coding:** Proper input validation using regex techniques and built-in security controls provided by programming frameworks in order to validate and sanitize all user inputs should be implemented to prevent common vulnerabilities like SQL injection, cross-site scripting (XSS) and command injection. Ensure that special characters or sequences in the application code are converted into an alternative, yet equivalent format that poses no threat when processed by the intended interpreter. This is done to prevent injection attacks and unexpected behavior, using encoding methods like URL encoding. In addition to encoding, escaping techniques involving adding extra characters or sequences before a special character or sequence to ensure correct interpretation and maintain the intended behavior of the code.

4.5 **File & Memory Management**: To develop a secure and robust application, it is imperative to prioritize meticulous file and memory management within your coding practices. File management involves handling files and file-related operations in a secure manner to prevent vulnerabilities, while memory management refer to proper allocation and deallocation of memory in a program to prevent vulnerabilities like buffer overflows and memory leaks.

4.6 **Optimise to Manage Complexity:** It involves implementing techniques to streamline complex systems, processes, or structures. The complex aspects of development phase are Concurrency, Dependency Management & Coupling. The coupling and dependencies should be minimized by including modular architectures, dependency injections, design patterns, building loosely coupled system. Concurrency may be achieved by dividing the application into multiple independent units of execution, namely processes and threads, and then executing them in parallel.

4.7 **Develop Software Technology Specific Security Checklist**: Organisation should develop a software technology / language specific secure coding checklist containing defined set of rules and guidelines dealing specifically with the peculiarities, defects and non-standard extensions in respective programming language / technology. This checklist may be used as a measure to evaluate the security of software or application. The conditions for defining the set of rules in the checklist should include:

a) Checking the violation of coding practices w.r.t. security flaws that may result in exploitable vulnerability.
b) Identification of the exceptional conditions that may require deviating from secure coding practices to ensure the proper functioning of the application.
c) Verification of conformance to the secure coding practice.

4.8 **Security Test Driven Development (STDD)**: The applications should be developed using STDD approach which incorporates security testing throughout the software development lifecycle (SDLC). It involves writing security tests before writing the actual code to ensure that security vulnerabilities are identified early and addressed before the code is deployed to production. The source code must undergo a static scan or Static Application Security Testing (SAST) before deployment to the production environment, followed by a dynamic scan or Dynamic Application Security Testing (DAST) to identify vulnerabilities in a live application. Thereafter, detected security gaps in code should be refactored in accordance with the security recommendations without impacting the code's functionality.

4.9 **Implementation of Threat Modelling in Application Development:** Threat modelling should be implemented in the application development process by first identifying the boundaries of the application, including its components, data flows, and interfaces. Thereafter, data flow diagram should be created to visualize the possible access points that attackers may exploit, along with the associated threats. Subsequently, countermeasures should be developed and tested to mitigate these threats. It is important to conduct threat modelling periodically to ensure the ongoing security of the application.

4.10 **Build Secure Environment for Application Development:** The application should be developed & build in secure environment. The environment should be separated and protected to prevent unauthorized access. Logging, monitoring, and auditing of trust relationships used for authorization and access are crucial among components within each environment. Enforce multi-factor authentication and conditional access across all environments to mitigate security risks. Encrypt sensitive data like Personal Identifiable Information (PII) and credentials. Implement defensive cybersecurity practices, including continuous monitoring and timely response to cyber incidents. Maintain a consistent practice of continuous monitoring and validation, requiring users to undergo re-validation at specified intervals before granting access. In the case of multi-factor authentication, users must navigate through multiple layers of authentication, extending beyond simple password-OTP verification. Implement stringent device control measures to monitor both users and their devices. This involves tracking device connections, resource utilization, and verifying device authorization status.

4.11 **Secure Use of Environment Variables:** To enhance the security of controlled file access and provide an additional layer of protection, it is recommended to utilize environment variables instead of plain text files. Furthermore, it is important to treat input from environment variables as untrusted and validate it accordingly. Storing sensitive information in environment variables should be avoided to minimize the risk of unauthorized access or exposure of sensitive data.

4.12 **Stored Procedures Over SQL Statements:** Stored procedures should be used instead of constructing SQL statements to reduce the chances of injection attacks.

As stored procedures restrict direct user access to the database table and provide an additional layer of security. Additionally, stored procedures should be integrated with security measures such as encryption, access control policies, and network security measures. It is important to address issues like version control and potential database vendor lock-in, as they can pose obstacles to secure implementation.

4.13 **Handle Error Messages, Commented Code and Exceptions:** Prior to deploying an application into a production environment, it is crucial to remove all commented code. Additionally, it is important to ensure that sensitive information, such as system details, network configurations, passwords, server names, IP addresses, or file system paths, is not exposed through error messages or URL contents. Proper exception handling should be implemented as a mandatory process, which involves identifying potential exceptions, using try-catch blocks, handling the exceptions appropriately, providing feedback to users when errors occur, logging the errors, and thoroughly testing the exception handling mechanism. It is recommended to keep necessary access and error logs of the application.

4.14 **Linear Data Structure and Multiple Inheritances:** Allocate space for linear data structures like arrays, linked lists, stacks, and vectors based on their required size. Implement proper access control using 'public,' 'private,' or 'protected' access modifiers in the code to specify the location or physical address of these data structures. Avoid multiple inheritances to mitigate potential code vulnerabilities, as declaring essential variables in the public scope can compromise the abstraction principle in object-oriented programming.

4.15 **Third Party and Open-Source Libraries, Components and APIs:** Security wrapper classes should be created for the open-source and third-party libraries used in the code. It is important to validate a library's ability to throw exceptions, including handling "out of memory" exceptions. Additionally, data entry and exit points of the application should be carefully examined and exceptional handling mechanisms should be implemented to ensure proper security measures.

An organization should integrate third-party components, open-source components and APIs into their development projects only after conducting a

thorough evaluation of their credibility and security posture. Implementing rate limits is crucial to prevent excessive API usage, thereby safeguarding the stability of both the organization's system and the third-party service. It is important to ensure secure API key management by employing robust cryptographic algorithms for key generation, regularly rotating API keys, and setting expiration dates for keys to minimize the risk of potential breaches. Additionally, organizations should maintain a Software Bill of Material (SBOM) for their applications, and avoid using vulnerable components & libraries.

4.16 **Build Trust Boundaries:** A trust boundary should be established between the front-end entry, exit points and the back-end of the application. This means that data received from client-side validations should be treated as untrusted and undergo validation on the back-end.

4.17 **Principle of Least Privileges:** It is essential to employ the principle of least privilege, ensuring that only the minimum necessary privileges are assigned. Additionally, privileges should be granted when required and promptly revoked once their intended purpose has been fulfilled. This principle should be encapsulated within a set of rules, tailored for distinct users and devices, thus ensuring consistent enforcement across all users and devices.

4.18 **Enhancing Maturity of Software Security:** Software security can be improved by integrating models such as CMMI (Capability Maturity Model Integration), which offers organizations a framework to evaluate, define, and improve their software development processes. Models such as CMMI can assess the development process and highlight areas for improvement. CMMI comprises various levels, from Level 1 characterized by ad hoc and unstable processes to Level 5 with well-organized, documented, and continuously improving processes. Code reviews typically come into play when an organization reaches levels 2 and 3.

5. PHASE – III: Guidelines for Audit of Applications

5.1 Source Code Review: It is a process that examines an application's source code to identify security flaws or vulnerabilities. It ensures adherence to coding standards, uncovers bugs, and identifies potential threats, such as SQL injection and cross-site scripting. Source code reviews can help detect errors and vulnerabilities that may have been overlooked during the development phase. Source code reviews encompass both manual and automated reviews. Manual reviews can evaluate the logic and design of the code, identifying issues that automated tools may overlook. Automated tools can scan large codebases for common vulnerabilities and coding errors. To enhance the quality of source code review in terms of security, it is imperative to integrate both static and dynamic scan i.e. Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). SAST provides early detection by analysing static code, addressing potential coding errors, while DAST evaluates the running application, simulating real-world attacks and identifying runtime vulnerabilities. The combination of SAST and DAST offers a comprehensive approach, fortifying the source code review process throughout the software development lifecycle and in real-world scenarios, thereby enhancing the overall security of the application.

5.2 Conduct Security Vulnerability Assessment: Organisation should engage CERT-In empanelled auditing organisation to conduct the security audit of the developed application and its related components. The objective of the audit should be discovery of all known vulnerabilities based on the comprehensive standards/framework such as Cyber Security Audit Baseline Requirements, OWASP Application Security Verification Standard (ASVS), OWASP Web Security Testing Guide, Mobile Application Security Verification Standard (MASVS), Mobile Application Security Testing Guide (MASTG), OWASP Application Programming

Interface (API) security checklist along with applicable regulatory framework, directions, guidelines & whitepapers issued by agencies. Critical vulnerabilities highlighted in audit reports should be patched by organisation immediately. After remediation actions, follow-up audits should be performed by auditing organisation to verify closure of vulnerabilities & nonconformities highlighted in the previous audit. Vulnerability assessments should be conducted periodically at-least once in a year or when there is any change in the application.

5.3 **Timeline for Completion of Audit:** Audit report should mention appropriate timelines for closure of vulnerabilities according to severity.

5.4 **Penetration Testing:** It involves simulation of real-world cyberattacks on a computer system, network, or application to provide valuable insights into potential security threats that could be exploited. Penetration Testing enables organizations to prioritize and fix vulnerabilities before they are exploited by malicious actors. It is advisable to conduct penetration testing iteratively with each release or modification of the source code to continually identify and address potential security threats in the application.

5.5 **Logging and Audit Trails:** Logging and audit trail functionality should be integrated in the application for troubleshooting and compliance requirement. Logging helps in error detection, troubleshooting, and performance optimization by maintaining a detailed record of application activities, enabling swift identification and resolution of issues. On the other hand, audit trails enhance security, compliance, and accountability by documenting every user action and system event within the application.

5.6 **Precondition for Assessment and Audit:** Application developed without any secure design and development practices should not be considered for assessment and audits. Auditee organizations and auditor organizations must confirm that application is designed & developed with secure practice prior to commencing any assessment.

6. PHASE – IV: Ensure Secure Application Deployment and Operations

6.1 Secure Deployment and Configuration: There should not be any changes in audited application code or configurations and application should be hosted in secure & tested environment. Ensure continuous logging and monitoring of logs. Utilize secure configuration management tools like Ansible, Puppet, etc., to establish secure deployment configurations. Implement secure communication protocols and access controls for the deployment environment, including firewalls, VPNs, and IP restrictions. When employing automated deployment tools such as Docker, Kubernetes, and Jenkins, verify and apply appropriate security measures. Regularly update software and dependencies to maintain a secure deployment environment.

6.2 Provision for Patch and Update: The comprehensive documentation detailing the security features integrated into the architecture, codebase, APIs, and data interactions of the application should be prepared. Also, ensure that all features are meticulously designed with backward compatibility in mind, and steer clear of making incompatible changes to existing APIs and data structures. In case vulnerabilities are discovered in the application, there should be a structured process to apply patches or fixes. This involves testing patches to ensure they don't introduce new issues / vulnerabilities, maintaining a change control process for documentation and accountability, and communicating changes to relevant stakeholders.

6.3 Secure Development of Update, Patch & Release to Mitigate Against Supply Chain Risk from Developers: Secure development of updates, patches, and

releases is essential to protect against supply chain risks originating from developers. This includes thorough testing, digital signing of code with digital certificates, and employing secure update distribution mechanisms to prevent tampering or unauthorized modifications. By prioritizing security at every stage of application development, organizations can mitigate the potential risks associated with supply chain vulnerabilities and ensure the integrity and safety of their software updates. Prioritize developer account security on code sharing platforms through access controls and monitoring. Implement and maintain secure build and deployment pipelines with robust enforcement of security checks.

****