Information Network Security Agency (INSA)

Secure Software Development and Management Standard

Version 1.0

2014 EC

Information Network Security Agency (INSA)

Secure Software Development and Management Standard

Version 1.0

2014 EC

Forward

Secure Software development and management need attention during the complete lifecycle in detail. Security requirements should be added to each SDLC model to ensure the software being developed is well secured. This standard highly recommends a core set of high-level secure software development requirements to be integrated within each SDLC implementation. The standard enables organizations to develop and manage secured software in order to reduce the number of vulnerabilities, mitigate the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences.

Draft national policies, laws, standards, and strategies that enable to ensure the information and computer-based key infrastructures security and oversight their enforcement upon approval is one of the power and duties are given to the Information Network Security Agency (INSA).

Therefore, this standard is issued by Information Network Security Agency (INSA) pursuant to Article 13 of Information Network Security Agency Re-establishment proclamation Execution council of ministers Regulation No.320/2014.

Topic **page**

Terminology and Acronyms

Terminology

The definitions of terminologies that are used in this standard are to be interpreted as described below.

- **Must**: This word means that the requirement is mandatory (an absolute requirement) of the standard.
- **Must Not**: This phrase means that the requirement is an absolute prohibition of the standard.
- **Should:** This word means that the requirement is “HIGHLY RECOMMENDED” to be implemented. There may exist valid reasons in particular circumstances to ignore a particular requirement, but the full implications must be understood and carefully weighed before choosing a different option. There must be a valid justification for ignoring the requirement or choosing a different option.
- **Should not:** This phrase means that the requirement is "NOT RECOMMENDED" to be implemented. There may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any requirement described with this label.
- **May:** This word, or the adjective “OPTIONAL”, means that an item is truly optional.

1. Introduction

Software assurance cannot be achieved by a single practice, tool, heroic effort, or checklist; rather it is the result of a comprehensive security software engineering process that spans all parts of development from early planning through end of life. It is also important to realize that, even within a single organization and associated Secure Software Development Lifecycle, there is no one-size-fits-all approach. The SDL must be firm in its approach to security but flexible enough in its application to accommodate variations in several factors, including different technologies and development methodologies in use and the risk profile of the applications in question.

1. Objective

1.1. General Objective

The main objective of SSDMS is to embed and incorporate proactive security controls throughout the development of software.

a) Providing concepts, principles, components, and processes;
b) Providing process-oriented and role-based mechanisms for establishing security requirements, assessing security risks, assigning a Targeted Level of Trust and selecting corresponding security controls and verification measures;

1.2. Specific Objectives

The specific objectives of the SSDMS enable to supports organizations during/while they are

- ✓ Governing and managing the entire software development life cycle within administrative security tools as well as assure the core security principles.
- ✓ Ensuring compliance to governance, regulations, and privacy. A software security professional needs to be well proficient in meeting regulatory and privacy requirements
- ✓ Enhancing professional capacity by raising awareness, providing training and educating on how to design, develop, and deploy secure software through tailored self-assessment model.
- ✓ Designing and developing security features embedded software.
- ✓ Testing and implementing the developed software in actual environment.

- ✓ Evaluating and maintaining the operations of software iteratively and adaptably.
- ✓ Providing an effective and measurable way for all types of organizations to analyze and improve their software security posture.
- ✓ Auditing the security and assuring their actual deployment with the seated controls.
- ✓ Leading/assuring software security stipulated security metrics that meet with:
    - Confidentiality: protection from disclosure
    - Integrity: protection from alteration
    - Availability: protection from destruction
    - Authentication: who is making the request
    - Authorization: what rights and privileges does the requestor have
    - Auditing: the ability to build historical evidence and
    - The management of configuration, sessions, and exceptions
- ✓ Determining procurement selection criteria for software procurement and outsourcing processes.
- ✓ Acquiring personnel richen with software security competencies.
- ✓ Defining and integrating software security features within careers' job description.

2. Target Audience Groups

2.1. Organizations and Departments

- ✓ Critical federal and regional government organizations that have responsibility and accountability to develop software, design software, maintain software, test software, procure software, outsource software, audit software security, consult software security, govern software security and use software must use this standard as a **mandatory** to ensure the security of their organization.
- ✓ Any private organizations (small, medium, large scale) which have responsibility and accountability to develop software, design software, maintain software, test software, procure software, outsource software, audit software security, consult software security, and govern software security are **encouraged** to use this standard.

2.2. Professional Careers

All Personnel who granted security clearance in their specific duty separation such as -

- ✓ Software Architects
- ✓ Software Developers
- ✓ Software Security Development Consultants
- ✓ Software Maintainers
- ✓ Software Testers
- ✓ Security Auditors
- ✓ Software Procurers
- ✓ Software Outsourcers

3. Scope

This standard covers security controls integrations via five critical security perspectives into each software development sub-lifecycles. These security perspectives are organized as follows:

- ✓ Governing and planning of software requirements.
- ✓ Designing of software architecture
- ✓ Implementation of software development and coding
- ✓ Verification and testing
- ✓ Deploying, maintenance and operation

Writing Convention

Throughout this document, statements are written using the word such as “Purpose”, “benefit” “standard” and “MUST”. The following paragraphs are intended to clarify how these standard statements are to be interpreted.

A subtitle “Purpose” indicates why the standard exists and “benefits” what advantage will get from applying this standard in the SDLC.

A reference that uses “**MUST**” indicates **mandatory** compliance. This makes all “MUST” statements easier to locate and interpret from the context of the topic. The IAO will adhere to the instructions as written.

A reference that uses “**SHOULD**” indicates **highly recommended** compliance. There may exist valid reasons in particular circumstances to ignore a particular requirement, but the full

implications must be understood and carefully weighed before choosing a different option. There must be a valid justification for ignoring the requirement or choosing a different option.

A reference that uses **May** indicates "OPTIONAL" compliance, means that an item is truly optional.

4. Characteristics/ Principles

The main characteristics of this standard are narrated as follows.

- ✓ **Actionable**: the SSDMS seated clear pathways for assuring security and improving maturity levels (security objective).
- ✓ **Auditable**: The SSDMS required countable and measurable security requirements within key performance indicators/metrics seated per each of the security practice activities.
- ✓ **Measurable**: The SSDMS fully defined clear maturity levels across every emerging security practice. This standard is simple to use and measurable. The solution details are easy enough to follow even for non-security personnel. It helps organizations analyze their current software security practices, build a security program in defined iterations, show progressive improvements insecure practices, define, and measure security-related activities.
- ✓ **Flexible**: SSDMS was defined with flexibility in mind so that small, medium, and large organizations (organizations of the federal government, regional government, private) using tailoring considerations and adaptive environments. It provides a means of knowing where an organization is on its journey towards software assurance and understanding what is recommended to move to the next level of maturity.
- ✓ **Adaptable**: the SDDS designed the security requirements as they can align with various emerging programming language technologies and platforms.
- ✓ **Extendable**: Organizations shall develop a specific standard, policy, or scheme that is the best fit and adapt to their specific needs and software category. For each software type, software cluster, software development model,

5. Governance

This Security Perspective contains the following three security requirement sets.

5.1. Strategy and Metrics Processes

Software assurance requires many different activities and concerns. The organization should be spending potential effort to build security. Organization efforts must be aligned, proportional and productive. The goal of the Strategy and Metrics security requirement set is to build an efficient and effective plan for realizing software security objectives within the organization.

A software security program should select and prioritize activities of the rest of the model, serves as the foundation for the organization's efforts. The practice should work on building the grand plan, maintaining and disseminating it.

At the same time, an organization should commit to keeping track of its security posture and program improvements. A metrics-driven approach should be included to ensure an accurate view of the organization's activities. The organization should measure security performance by key indicators for further maintenance.

5.1.1. Streams

1. **Create and Promote:** This activity stream creates and promotes a software security roadmap to set the objectives of the organization on this topic and increase alignment among stakeholders.
2. **Measure and Improve:** This activity stream intends to drive the validity, relevance, and improvement of the software security roadmap through measurements of performance within the organization.

5.1.2. Security Activity

5.1.2.1. Strategy and Metrics Process (SMP1)

- ✓ Organization's risk appetite should be identified.
- ✓ Basic security metrics should be defined.

Purpose

The objective of this process is to identify objectives and means of measuring the effectiveness of the security program.

Benefits

- ✓ Common understanding of organization's security posture,
- ✓ Basic insights into secure software program's effectiveness and efficiency.

Standard

1. The organization **must** understand, based on application risk exposure, what threats exist or may exist, as well as how tolerant executive leadership is of these risks. This understanding is a key component of determining software security assurance priorities.

2. The organization **must** ascertain these threats, interview business owners and stakeholders, and document drivers specific to industries where the organization operates as well as drivers specific to the organization.

3. The organization **must** have gathered information includes worst-case scenarios that could impact the organization, as well as opportunities where an optimized software development lifecycle and more secure applications could provide a market differentiator or create additional opportunities.

4. Gathered information **must** provide a baseline for the organization to develop and promote its application security program.

5. Items in the program **should be** prioritized to address threats and opportunities most important to the organization.

6. The baseline **must be** split into several risk factors and drivers linked directly to the organization's priorities and used to help build a risk profile of each custom-developed application by documenting how they can impact the organization if they are compromised.

7. The baseline and individual risk factors **should be** published and made available to application development teams to ensure a more transparent process of creating application risk profiles and incorporating the organization's priorities into the program.

8. Additionally, these goals **should** provide a set of objectives that should be used to ensure all application security program enhancements provide direct support of the organization's current and future needs.

9. The quality criteria for an organization's security posture **must be**:

- ✓ Capturing the risk appetite of the organization's executive leadership.
- ✓ The organization's leadership shall vets and approves the set of risks.
- ✓ Identifying the main business and technical threats to assets and data.
- ✓ Documenting risks and stores them in an accessible location.

10. The Organization **must** define and document metrics to evaluate the effectiveness and efficiency of the software security program.

11. Improvements are measurable and the organization **shall** use them to secure future support and funding for the program.

12. Considering the dynamic nature of most development environments, metrics **must be** comprised of measurements in the following categories

- ✓ Effort metrics measure the effort spent on security. For example, training hours, time spent performing code reviews, and the number of applications scanned for vulnerabilities.
- ✓ Result metrics measure the results of security efforts. Examples include the number of un patched security defects and the number of security incidents involving application vulnerabilities.
- ✓ Environment metrics measure the environment where security efforts take place. Examples include the number of applications or lines of code as a measure of difficulty or complexity.

13. Each measure by itself is useful for a specific purpose, but a combination of two or three metrics together helps explain spikes in metrics trends. For example, a spike in the total number of vulnerabilities may be caused by the organization on boarding several new applications that have not been previously exposed to the implemented application security mechanisms.

14. Alternatively, an increase in the environment metrics without a corresponding increase in the effort or result could be an indicator of a mature and efficient security program.

15. While identifying metrics, it's always recommended to stick to the metrics that meet several criteria

- ✓ Consistently Measured
- ✓ Inexpensive to gather
- ✓ Expressed as a cardinal number or a percentage Expressed as a unit of measure

16. Document metrics and include descriptions of best and most efficient methods for gathering data, as well as recommended methods for combining individual measures into meaningful metrics. For example, the number of applications and a total number of defects across all applications may not be useful by themselves but, when combined with the number of outstanding high-severity defects per application, they provide a more actionable metric.

17. The quality criteria for basic security metrics **should be**:

- ✓ Documenting each metric, including a description of the sources, measurement coverage, and guidance on how to use it to explain application security trends
- ✓ Metrics include measures of efforts, results, and the environment measurement categories
- ✓ Most of the metrics are frequently measured, easy or inexpensive to gather, and expressed as a cardinal number or a percentage
- ✓ Software security and development teams publish metrics

5.1.2.2. Strategy and Metrics Process (SMP2)

- ✓ Security strategy should be defined.
- ✓ Strategic key performance indicators should be seated

Purpose

- ✓ The objective of this process is to establish a unified strategic roadmap for software security within the organization.

Benefits

- ✓ Available and agreed up roadmap of secure software program
- ✓ Transparency on organization secure software program's performance

Standard

1. Based on the magnitude of assets, threats, and risk tolerance, organization **must** develop a security strategic plan and budget to address business priorities around application security.

2. The plan **must** cover 1 to 3 years and includes milestones consistent with the organization's business drivers and risks.

3. It **should** provide tactical and strategic initiatives and follows a roadmap that makes its alignment with business priorities and needs visible.

4. In the roadmap, the organization **must** reach a balance between changes requiring financial expenditures, changes of processes and procedures, and changes impacting the organization's culture. This balance helps accomplish multiple milestones concurrently and without overloading or exhausting available resources or development teams.

5. The milestones **should be** frequent enough to help monitor program success and trigger timely roadmap adjustments.

6. For the program to be successful, the application security team **must** obtain buy-in from the organization's stakeholders and application development teams.

7. A published plan **must be** available to anyone who is required to support or participate in its implementation.

8. The quality criteria for security strategy plan **should**:

- ✓ Reflects the organization's business priorities and risk appetite.
- ✓ Includes measurable milestones and a budget.
- ✓ Be consistent with the organization's business drivers and risks.
- ✓ Lays out a roadmap for strategic and tactical initiatives.
- ✓ Having buy-in from stakeholders, including development teams.

9. Once the organization has defined its application security metrics, **must** collect enough information to establish realistic goals.

10. Test identified metrics to ensure the organization can gather data consistently and efficiently over a short period. After the initial testing period, the organization **should** have enough information to commit to goals and objectives expressed through Key Performance Indicators (KPIs).

11. While several measurements are useful for monitoring the information security program and its effectiveness, KPIs are comprised of the most meaningful and effective metrics. Aim to remove volatility common in application development environments from KPIs to reduce

chances of unfavorable numbers resulting from temporary or misleading individual measurements. Base KPIs on metrics considered valuable not only to Information Security professionals but also to individuals responsible for the overall success of the application, and the organization's leadership. The organization **must** view KPIs as definitive indicators of the success of the whole program and consider them actionable.

12. The organization **must** fully document KPIs and distribute them to the teams contributing to the success of the program as well as the organization's leadership. Ideally, include a brief explanation of the information sources for each KPI and the meaning of the numbers are high or low.

13. The organization **must** include short and long-term goals and ranges for unacceptable measurements requiring immediate intervention.

14. The organization **must** share action plans with application security and application development teams to ensure full transparency in the understanding of the organization's objectives and goals.

15. The quality criteria for strategic key performance indicators should be:

- ✓ Defined KPIs after gathering enough information to establish realistic objectives.
- ✓ Developed KPIs with the buy-in from the leadership and teams responsible for software security KPIs are available to the application teams and include acceptability thresholds and guidance in case teams need to take action.
- ✓ Success of the application security program is visible based on defined KPIs.

5.1.2.3. Strategy and Metrics Process (SMP3)

- ✓ Security and business strategies should be aligned.
- ✓ Security program through metrics should be initiated.

Purpose

The objective of this process is to align security efforts with the relevant organizational indicators and asset values.

Benefits

- ✓ Continuous secure software program alignment with the organization's business goals
- ✓ Continuous improvement of your program according to results.

Standard

1: -The organization **must** review the application security plan periodically for ongoing applicability and support of the organization's evolving needs and future growth.

2: -To do this, the organization **must** repeat the steps from the first two maturity levels of this Security Practice at least annually.

3: -The goal is for the plan to always support the current and future needs of the organization, which **must** ensure the program is aligned with the business.

4: -In addition to reviewing the business drivers, the organization **must** closely monitor the success of the implementation of each of the roadmap milestones.

5: - The organization **must** evaluate the success of the milestones based on a wide range of criteria, including completeness and efficiency of the implementation, budget considerations, and any cultural impacts or changes resulting from the initiative.

6: - The organization **must** review missed or unsatisfactory milestones and evaluate possible changes to the overall program.

7: -The organization **must** develop dashboards and measurements for management and teams responsible for software development to monitor the implementation of the roadmap.

8: - These dashboards are detailed enough to identify individual projects and initiatives and **should** provide a clear understanding of whether the program is successful and aligned with the organization's needs.

9: -The quality criteria for security and business strategies should be:

- ✓ Reviewing and updating the plan in response to significant changes in the business environment, the organization, or its risk appetite
- ✓ Plan update steps include reviewing the plan with all the stakeholders and updating the

- ✓ Adjusting the plan and roadmap based on lessons learned from completed roadmap activities The organization publish progress information on roadmap activities, making sure they are available to all stakeholders

10: -The organization **must** define guidelines for influencing the Application Security program based on the KPIs and other application security metrics.

11: -These guidelines **must** combine the maturity of the application development process and procedures with different metrics to make the program more efficient.

12: -The following examples show a relationship between measurements and ways of evolving and improving application security:

- ✓ Focus on the maturity of the development lifecycle makes the relative cost per defect lower by applying security proactively.
- ✓ Monitoring the balance between effort, result, and environment metrics improves the program's efficiency and justifies additional automation and other methods for improving the overall application security baselines.
- ✓ Individual Security Practices could provide indicators of the success or failure of individual application security initiatives.
- ✓ Effort metrics help ensure application security work is directed at the more relevant and important technologies and disciplines.

13: -When defining the overall metrics strategy, **must** keep the end goal in mind and define what decisions can be made as a result of changes in KPIs and metrics as soon as possible, to help guide the development of metrics.

14: - The quality criteria for Security program through metrics:

- ✓ Reviewing KPIs at least yearly for their efficiency and effectiveness.
- ✓ KPIs and application security metrics trigger most of the changes to the application security strategy.

5.2. Policy and Compliance Processes

The Policy and Compliance security requirement set should focus on understanding and meeting external legal and regulatory requirements while driving internal security frameworks to ensure compliance in a way that's aligned with the business purpose of the organization.

A driving theme for improvement within this practice is describing the organization's standards and third-party obligations as software requirements, enabling efficient and automated audits that should be leveraged within the software development life cycle and continuously demonstrate that all controls are met.

Provision of this practice should require an organization-wide understanding of both internal standards and external compliance drivers while also maintaining low latency checkpoints with project teams to ensure no project is operating outside expectations without visibility.

5.2.1. Streams

1. **Policy and Standards:** This activity stream focuses on maintaining policies and standards and providing them to support integration into the software development life cycle.
2. **Compliance Management:** This activity stream focuses on identifying and providing compliance requirements to support integration into the software development life cycle.

5.2.2. Security Activity

5.2.2.1. Policy and Compliance Process (PCP1)

- ✓ Security governance frameworks should be defined.
- ✓ Compliance requirements should be identified.

Purpose

The objective of this process is to identify and document governance and compliance drivers relevant to the organization.

Benefits

- ✓ Clear expectation of minimum security level in the organization
- ✓ Security policies and standards aligned with external compliance drivers

Standard

1: -The organization **must** develop a library of policies and standards to govern all aspects of software development in the organization.

2: - Policies and standards **must be** based on existing industry standards and appropriate for the organization's industry.

3: -Due to the full range of technology-specific limitations and best practices, the organization **must** review proposed standards with the various product teams.

4: -With the overarching objective of increasing security of the applications and computing infrastructure, the organization **should** invite product teams to offer feedback on any aspects of the standards that would not be feasible or cost-effective to implement, as well as opportunities for standards to go further with little effort on the product teams.

5: -For policies, the organization **must** emphasize high-level definitions and aspects of application security that do not depend on a specific technology or hosting environment.

6: -The organization **must** focus on broader objectives of the organization to protect the integrity of its computing environment, safety and privacy of the data, and maturity of the software development life-cycles.

7: -For larger organizations, policies may qualify specific requirements based on data classification or application functionality, but **should** not be detailed enough to offer technology-specific guidance.

8: -For standards, the organization **must** incorporate requirements set forth by policies, and focus on technology-specific implementation guidance intended to capture and take advantage of the security features of different programming languages and frameworks.

9: -Standards **should** require input from senior developers and architects considered experts in various technologies in use by the organization.

10: -The organization **must** create them in a format that allows for periodic updates.

11: -The organization **must** label or tag individual requirements with the policy or a 3rd party requirement, to make maintenance and audits easier and more efficient.

12: -The quality criteria for security governance frameworks:

- ✓ Adapting existing standards appropriate for the organization's industry to account for domain-specific considerations.
- ✓ The adapted standards are aligned with policies and incorporate technology-specific implementation guidance

13: -The organization **must** create a comprehensive list of all compliance requirements, including any triggers that could help determine which applications are in scope.

14: - Compliance requirements **may be** considered in scope based on factors such as geographic location, types of data, or contractual obligations with clients or business partners. **15: -**The organization **must** review each identified compliance requirement with the appropriate experts and legal, to ensure the obligation is understood.

16: -Since many compliance obligations vary in applicability based on how the data is processed, stored, or transmitted across the computing environment, compliance drivers **should** always indicate opportunities for lowering the overall compliance burden by changing how the data is handled.

17: -The organization **must** evaluate publishing a compliance matrix to help identity which factors could put an application in scope for a specific regulatory requirement.

18: - The organization **must** have the matrix indicate which compliance requirements are applicable at the organization level and do not depend on individual applications.

19: - The matrix **must** provide at least a basic understanding of useful compliance requirements to review obligations around different applications.

20: -Since many compliance standards are focused around security best practices, many compliance requirements **may** already be a part of the Policy and Standards library published by the organization. Therefore, once review compliance requirements, **must map** them to any applicable existing policies and standards.

21: -Whenever there are discrepancies, the organization **must** update the policies and

standards to include organization-wide compliance requirements. Then, begin creating compliance-specific standards only applicable to individual compliance requirements.

22: -The goal is to have a compliance matrix that indicates which policies and standards have more detailed information about compliance requirements, as well as **must** ensure individual policies and standards reference applicable compliance requirements.

23: -The quality criteria for compliance requirements:

- ✓ Identified sources of external compliance obligations.
- ✓ Captured and reconciled compliance obligations from all sources.

5.2.2.2. Policy and Compliance Process (PCP2)

- ✓ Security verifications and security audit procedures should be established.
- ✓ Policy and compliance requirements should be standardized and documented.

Purpose

The objective of this process is to establish application-specific security and compliance baseline.

Benefits

- ✓ Common understanding of how to reach compliance with security policies for product teams.
- ✓ Common understanding how to reach compliance with external compliance drivers for product teams.

Standard

1: -To assist with the ongoing implementation and verification of compliance with policies and standards, the organization must develop application security and appropriate test scripts related to each applicable requirement.

2: - The organization **must** organize these documents into libraries and make them available to all application teams in formats most conducive for inclusion into each application.

3: -The organization **must** clearly label the documents and link them to the policies and

standards they represent, to assist with the ongoing updates and maintenance.

4: - The organization **must** have version policies and standards and include detailed changelogs with each iterative update to make ongoing inclusion into different products' SDLC easier.

5: - The organization **must** write application security requirements in a format consistent with the existing requirements management processes.

6: - The organization **may** need more than one version catering to different development methodologies or technologies.

7: -The goal is to make it easy for various product teams to incorporate policies and standards into their existing development life cycles needing minimal interpretation of requirements.

8: -The organization **must** test scripts to help reinforce application security requirements through clear expectations of application functionality, and guide automated or manual testing efforts that may already be part of the development process.

9: -These efforts not only help each team establish the current state of compliance with existing policies and standards, but also **must** ensure compliance as applications continue to change.

10: -The quality criteria for security verifications and security audit procedures:

- ✓ Creating verification checklists and test scripts where applicable, aligned with the policy's requirements and the implementation guidance in the associated standards.
- ✓ Creating versions adapted to each development methodology and technology the organization uses

11: -The organization **must** develop a library of application requirements and test scripts to establish and verify the regulatory compliance of applications. Some of these are tied to individual compliance requirements like PCI or GDPR, while others are more general and address global compliance requirements such as ISO.

12: - The library **must be** available to all application development teams.

13: - It **must** include guidance for determining all applicable requirements including

considerations for reducing the compliance burden and scope.

14: -The organization **must** implement a process to periodically re-assess each application's compliance requirements.

15: -Re-assessment **must** include reviewing all application functionality and opportunities to reduce scope to lower the overall cost of compliance.

16: -Requirements **must** include enough information for developers to understand functional and non-functional requirements of the different compliance obligations.

17: -They **must** include references to policies and standards, and provide explicit references to regulations.

18: - If there are questions about the implementation of a particular requirement, the original text of the regulation **may** help interpret the intent more accurately.

19: -Each requirement **must** include a set of test scripts for verifying compliance. In addition to assisting QA with compliance verification, these **may** help clarify compliance requirements for developers and make the compliance process transparent.

20: -Requirements **must** have a format that allows importing them into individual requirements repositories. Further, clarify compliance requirements for developers and ensure the process of achieving compliance is fully transparent.

21: -The quality criteria for policy and compliance requirements:

- ✓ Mapping each external compliance obligation to a well-defined set of application requirements.
- ✓ Defining verification procedures, including automated tests, to verify compliance with compliance-related requirements.

5.2.2.3. Policy and Compliance Process (PCP3)

- ✓ Compliance to security administrative controls should be measured.
- ✓ Compliance to external requirements should be measured.

Purpose

The objective of this process is to measure adherence to policies, standards, and 3rd-

party requirements.

Benefits

- ✓ Understanding of organization's compliance with policies and standards.
- ✓ Understanding of organization's compliance with external compliance drivers.

Standard

1: -The organization **must** develop a program to measure each application's compliance with existing policies and standards. Mandatory requirements should be motivated and reported consistently across all teams.

2: -Whenever possible, **must** tie compliance status into automated testing and report with each version.

3: -The organization **must** compliance reporting includes the version of policies and standards and appropriate code coverage factors.

4: - The organization **must** encourage non-compliant teams to review available resources such as security requirements and test scripts, to ensure non-compliance is not a result of inadequate guidance.

5: -Must forward issues resulting from insufficient guidance to the teams responsible for publishing application requirements and test scripts, to include them in future releases.

6: - The organization **must** escalate issues resulting from the inability to meet the policies and standards of the teams that handle application security risks.

7: -The quality criteria for Compliance to security administrative controls:

- ✓ Organization may have procedures (automated, if possible) to regularly generate compliance reports.
- ✓ Deliver compliance reports to all relevant stakeholders.
- ✓ Stakeholders use the reported compliance status information to identify areas for improvement.

8: -The organization **must** develop a program for measuring and reporting on the status of compliance between different applications. Application requirements and test scripts help

determine the status of compliance.

9: - The organization **must** leverage testing automation to promptly detect compliance regressions infrequently updated applications and ensure compliance is maintained through the different application versions.

10: -Whenever fully automated testing is not possible, QA, Internal Audit, or Information Security teams **must** assess compliance periodically through a combination of manual testing and interview.

11: -While full compliance is always the ultimate goal, the organization **must** include tracking remediation actions and periodic updates in the program.

12: -The organization **must** review compliance remediation activities periodically to check teams are making appropriate progress, and those remediation strategies will be effective in achieving compliance.

13: - To further improve the process, **must** develop a series of standard reports and compliance scorecards. These help individual teams understand the current state of compliance, and the organization manages assistance for remediating compliance gaps more effectively.

14: -The organization **must** review compliance gaps requiring significant expenses or development with the subject-matter experts and compare them against the cost of reducing the application's functionality, minimizing scope, or eliminating the compliance requirement.

15: -Long-term compliance gaps require management approval and a formal compliance risk acceptance, so they **must** receive appropriate attention and scrutiny from the organization's leadership.

16: -The quality criteria for compliance to external requirements:

- ✓ Organization may have established well-defined compliance metrics.
- ✓ Organization may measure and report on applications' compliance metrics regularly.
- ✓ Stakeholders use the reported compliance status information to identify compliance gaps and prioritize gap remediation efforts

5.3. Capability Building Processes

The Capability Building security requirement is set to focus on arming personnel involved in the software lifecycle with knowledge and resources to design, develop, and deploy secure software.

The organization provides training for employees and prospers their security awareness for improvement across the objectives. As an organization progresses, it shall build a broad base of training starting with developers and moving to other roles, culminating with the addition of role-based training to ensure applicability and effectiveness.

In addition to training, this security requirement set also requires the organization to make a significant investment in improving organizational culture to promote application security through collaboration between teams. Collaboration tools and increased transparency between technologies and tools support this approach to improve software security.

5.3.1. Streams

1. **Training and Awareness:** This activity stream focuses on increasing the overall knowledge around software security among the different stakeholders within the organization.
2. **Organization and Culture:** This activity stream focuses on promoting the culture of software security within the organization as an important success factor of a software development project.

5.3.2. Security Activity

5.3.2.1. Capability Building Process (CBP1)

- ✓ Stakeholders' capability should be built.
- ✓ Security champions and exemplary should be identified

Purpose

The objective of this process is to offer staff access to resources around the topics of secure development and deployment.

Benefits

- Basic security awareness for all relevant employees.
- Basic embedding of security in the development organization.

Standard

1: -The organization **must** conduct security awareness training for all roles currently involved in the management, development, testing, or auditing of the software. The goal is to increase the awareness of application security threats and risks, security best practices, and secure software design principles.

2: -The organization **must** develop training internally or procure it externally.

3: - Ideally, **must** deliver training in person so participants can have discussions as a team, but Computer- Based Training (CBT) is also an option.

4: -Course content **should** include a range of topics relevant to application security and privacy while remaining accessible to a non-technical audience.

5: -Suitable concepts **must be** secure design principles including Least Privilege, Defense-in-Depth, Fail Secure (Safe), Complete Mediation, Session Management, Open Design, and Psychological Acceptability.

6: -Additionally, the training **should** include references to any organization-wide standards, policies, and procedures defined to improve application security.

7: -Training **must be** mandatory for all employees and contractors involved with software development and include an auditable sign-off to demonstrate compliance.

8: - The organization **must** consider incorporating innovative ways of delivery to maximize its effectiveness and combat desensitization.

9: -The quality criteria for stakeholders' capability:

- Organization may have continuously updated training.
- Training is repeatable, consistent, and available to anyone involved with software development lifecycle Training includes concepts such as Least Privilege, Defense-in-Depth, Fail Secure (Safe), Complete Mediation, Session Management, Open Design, and Psychological Acceptability.

- ✓ Training requires a sign-off or an acknowledgment from attendees.
- ✓ Training is required during employees' on boarding process.

10: - The organization **must** implement a program where each software development team has a member considered a "Security Champion" who is the liaison between Information Security and developers.

11: -Depending on the size and structure of the team the "Security Champion" **may be** a software developer, tester, or product manager. The "Security Champion" has a set number of hours per week for Information Security related activities.

12: -They **must** participate in periodic briefings to increase awareness and expertise in different security disciplines.

13: - "Security Champions" **should** have additional training to help develop these roles as Software Security subject- matter experts.

14: - The organization **may** need to customize the way it creates and support "Security Champions" for cultural reasons.

15: -The goals of the position **should be** to increase the effectiveness and efficiency of application security and compliance and to strengthen the relationship between various teams and Information Security.

16: -To achieve these objectives, "Security Champions" **must** assist with researching, verifying, and prioritizing security and compliance-related software defects.

17: -They **must be** involved in all Risk Assessments, Threat Assessments, and Architectural Reviews to help identify opportunities to remediate security defects by making the architecture of the application more resilient and reducing the attack threat surface.

18: -In addition to assisting Information Security, "Security Champions" **must** provide periodic reviews of all security-related issues for the project team so everyone is aware of the problems and any current and future remediation efforts.

19: -These reviews **must be** leveraged to help brainstorm solutions to more complex problems by engaging the entire development team.

20: -The quality criteria for security champions and exemplary:

- ✓ Software Security and Development teams receive periodic briefings from Security Champions on the overall status of security initiatives and fixes.
- ✓ The Security Champion reviews the results of external testing before adding to the application backlog.
- ✓ Security Champions receive appropriate training.

5.3.2.2. Capability Building Process (CBP2)

- ✓ Tailoring considerations and scoping should be done for secure capability building.
- ✓ Knowledge management and centers of excellence should be implemented.

Purpose

The objective of this process is to educate all personnel in the software lifecycle with technology and role- specific guidance on secure development.

Benefits

- ✓ Relevant employee roles trained according to their specific role.
- ✓ Specific security best practices tailored to the organization.

Standard

1: -The organization **must** conduct instructor-led or CBT (Computer Based Training) security training specific to the organization's roles and technologies, starting with the core development team.

2: -The organization **must** customize training for product managers, software developers, testers, and security auditors, based on each group's technical needs.

3: -Product managers **should** train on topics related to Security Perspectives and security requirements, with emphasis on security requirements, threat modeling, and defect tracking.

4: -Developers **must** train on coding standards and best practices for the technologies they work with to ensure the training directly benefits application security.

5: - Developers **should** have a solid technical understanding of vulnerabilities and similar weaknesses relevant to the technologies and frameworks used and the most common remediation strategies for each issue.

6: -Testers **should** train on the different testing tools and best practices for technologies used in the organization, and in tools that identify security defects.

7: -Security auditors **must** train on the software development lifecycle, application security mechanisms used in the organization, and the process for submitting security defects for remediation.

8: -Security Champions **must** train on security topics from various phases of the SDLC. They receive the same training as developers and testers, but also **must** understand threat modeling and secure design, as well as security tools and technologies that can be integrated into the built environment.

9: -Ideally, the organization must identify a subject-matter expert in each technology to assist with procuring or developing the training content and updating it regularly.

10: -The training **should** consist of demonstrations of vulnerability exploitation using intentionally weakened applications. Include results of the previous penetration as examples of vulnerabilities and implement remediation strategies. Ask a penetration tester to assist with developing examples of vulnerability exploitation demonstrations.

11: -Training **should be** mandatory for all employees and contractors involved with software development, and include an auditable sign-off to demonstrate compliance.

12: -Whenever possible, training **should** also include a test to ensure understanding, not just compliance.

13: - The organization **must** update and deliver training annually to include changes in the organization, technology, and trends.

14: - The organization **should** survey training participants to evaluate the quality and relevance of the training.

15: - The organization **must** gather suggestions of other information relevant to their work or environments.

16: -The quality criteria for tailoring considerations and scoping:

- ✓ Training is mandatory for all employees and contractors.
- ✓ Training includes input from in-house SMEs and trainees.
- ✓ Training includes demonstrations of tools and techniques developed in-house.
- ✓ Training includes all topics and adds more specific tools, techniques, and demonstrations.
- ✓ The organization may use feedback to enhance and make future training more relevant.

5.3.2.3. Capability Building Process (CBP3)

- ✓ Security guidance should be standardized and documented.
- ✓ Security community and liaison committee should be established.

Purpose

The objective of this process is to develop in-house training programs facilitated by developers across different teams.

Benefits

- ✓ Adequate security knowledge of all employees ensured prior to working on critical tasks.
- ✓ Collective development of security know-how among all product teams.

Standard

1: -The organization **must** implement a formal training program requiring anyone involved with the software development lifecycle to complete appropriate role and technology-specific training as part of the on boarding process.

2: -Based on the criticality of the application and user's role, **must** consider restricting access until the on boarding training has been completed.

3: -While the organization may source some modules externally, the program **must be** facilitated and managed in-house and includes content specific to the organization going beyond general security best practices.

4: -The program **should** have a defined curriculum, checks participation, and tests understanding and competence.

5: -The training **must be** consisting of a combination of industry best practices and organization's internal standards, including training on specific systems used by the organization.

6: -In addition to issues directly related to security, the organization **must** include other standards to the program, such as code complexity, code documentation, naming convention, and other process-related disciplines.

7: -This training minimizes issues resulting from employees following practices incorporated outside the organization and **must** ensures continuity in the style and competency of the code.

8: -To facilitate progress monitoring and successful completion of each training module the organization **must** have a learning management platform or another centralized portal with similar functionality.

9: -Employees **may** monitor their progress and have access to all training resources even after they complete initial training.

10: - The organization **must** review issues resulting from employees not following established standards, policies, procedures, or security best practices at least annually to gauge the effectiveness of the training and ensure it covers all issues relevant to the organization.

11: - The organization **must** update the training periodically and train employees on any changes and most prevalent security deficiencies.

12: -The quality criteria for security guidance:

- ✓ Training may be based on internal standards, policies, and procedures.
- ✓ Organization may have a learning management system that is used to track trainings and certifications.
- ✓ Organization may use certification programs or attendance records to determine access to development systems and resources.

13: -Security **should be** the responsibility of all employees, not just the Information Security team.

14: - The organization **must** deploy communication and knowledge sharing platforms to help developers build communities around different technologies, tools, and programming languages. In these communities' employees share information, discuss challenges with other developers, and search the knowledge base for answers to previously discussed issues.

15: - The organization **should** form communities around roles and responsibilities and enable developers and engineers from different teams and business units to communicate freely and benefit from each other's expertise.

16: -The organization **must** encourage participation, set up a program to promote those who help the most people as thought leaders, and have management recognize them.

17: - In addition to improving application security, this platform **may** help identify future members of the Secure Software Center of Excellence, or 'Security Champions' based on their expertise and willingness to help others.

18: -The Secure Software Center of Excellence and Application Security teams **must** review the information portal regularly for insights into the new and upcoming technologies, as well as opportunities to assist the development community with new initiatives, tools, programs, and training resources.

19: -The organization **must** use the portal to disseminate information about new standards, tools, and resources to all developers for the continued improvement of SDLC maturity and application security.

20: -The quality criteria for security community and liaison committee:

- ✓ The organization promotes use of a single portal across different teams and business units.
- ✓ The portal is widely recognized by developers and architects as a centralized repository of the organization-specific application security information.
- ✓ The portal is used for timely information such as notification of security incidents, tool updates, architectural standard changes, and other related announcements.

- ✓ The portal provides access to application-specific security metrics.
- ✓ All content is considered persistent and searchable.

1: -The organization **must** implement a formal training program requiring anyone involved with the software development lifecycle to complete appropriate role and technology-specific training as part of the on boarding process.

2: -Based on the criticality of the application and the user's role, **must** consider restricting access until the on boarding training has been completed.

3: -While the organization may source some modules externally, the program **must be** facilitated and managed in-house and includes content specific to the organization going beyond general security best practices.

4: -The program **should** have a defined curriculum, checks participation, and test understanding and competence.

5: -The training **must be** consisting of a combination of industry best practices and the organization's internal standards, including training on specific systems used by the organization.

6: -In addition to issues directly related to security, the organization **must** include other standards to the program, such as code complexity, code documentation, naming convention, and other process-related disciplines.

7: -This training minimizes issues resulting from employees following practices incorporated outside the organization and **must** ensure continuity in the style and competency of the code.

8: -To facilitate progress monitoring and successful completion of each training module the organization **must** have a learning management platform or another centralized portal with similar functionality.

9: -Employees **may** monitor their progress and have access to all training resources even after they complete initial training.

10: - The organization **must** review issues resulting from employees not following established standards, policies, procedures, or security best practices at least annually to

gauge the effectiveness of the training and ensure it covers all issues relevant to the organization.

11: - The organization **must** update the training periodically and train employees on any changes and most prevalent security deficiencies.

12: -The quality criteria for security guidance:

- ✓ Training may be based on internal standards, policies, and procedures.
- ✓ Organization may have a learning management system that is used to track training and certifications.
- ✓ Organization may use certification programs or attendance records to determine access to development systems and resources.

13: -Security **should be** the responsibility of all employees, not just the Information Security team.

14: - The organization **must** deploy communication and knowledge-sharing platforms to help developers build communities around different technologies, tools, and programming languages. In these communities' employees share information, discuss challenges with other developers, and search the knowledge base for answers to previously discussed issues.

15: - The organization **should** form communities around roles and responsibilities and enable developers and engineers from different teams and business units to communicate freely and benefit from each other's expertise.

16: - The organization **must** encourage participation, set up a program to promote those who help the most people as thought leaders, and have management recognize them.

17: - In addition to improving application security, this platform **may** help identify future members of the Secure Software Center of Excellence or 'Security Champions' based on their expertise and willingness to help others.

18: -The Secure Software Center of Excellence and Application Security teams **must** review the information portal regularly for insights into the new and upcoming technologies, as well as opportunities to assist the development community

with new initiatives, tools, programs, and training resources.

19: - The organization **must** use the portal to disseminate information about new standards, tools, and resources to all developers for the continued improvement of SDLC maturity and application security.

20: -The quality criteria for the security community and liaison committee:

- ✓ The organization promotes the use of a single portal across different teams and business units.
- ✓ The portal is widely recognized by developers and architects as a centralized repository of the organization-specific application security information.
- ✓ The portal is used for timely information such as notification of security incidents, tool updates, architectural standard changes, and other related announcements.
- ✓ The portal provides access to application-specific security metrics.
- ✓ All content is considered persistent and searchable.

6. Architecture and Design

This business function contains the following three security practices

6.1. Threat Assessment

The Threat Assessment (TA) practice focuses on identifying and understanding project-level risks based on the functionality of the software being developed and the characteristics of the runtime environment. From details about threats and likely attacks against each project, the organization as a whole operates more effectively through better decisions about prioritization of initiatives for security. Additionally, decisions for risk acceptance are more informed, therefore better aligned to the business.

By starting with simple threat models and building application risk profiles, an organization improves over time. Ultimately, a sophisticated organization would maintain this information in a way that is tightly coupled to the compensating factors and pass-through risks from external entities. This provides a greater breadth of understanding for potential downstream impacts from security issues while keeping a close watch on the organization's current performance against known threats.

6.1.1. Streams

1. **Application Risk Profile:** - An application risk profile helps to identify which applications can pose a serious threat to the organization if they were attacked or breached.
2. **Threat Modeling: -** Threat modeling is intended to help software development teams understand what risks exist in what is being built, what could go wrong, and how we the risks can be mitigated or remediated.

6.1.2. Security Activity

6.1.2.1. Threat Assessment (TA1)

In this security practice level, there are an activity that should have to be performed in order to secure the software

- ✓ Perform application risk assessment

- ✓ Perform basic threat modeling

Purpose:

The objective of this process is to consider security explicitly during the software requirements process.

Benefit:

- ✓ Ability to classify applications according to risk
- ✓ Identification of architectural design flaws in the applications

Standard:

1: - The Designer should estimate the potential business impact that it poses for the organization in case of an attack.

2: - The Designer should evaluate the impact of a breach in the confidentiality, integrity and availability of the data or service.

3: - The Designer must use a scheme to classify applications according to the risk.

4: - The Designer must use a qualitative scheme that translates these characteristics into a value that is often effective.

5: - The Designer **should** be model threats in a structured activity for identifying, evaluating, and managing system threats, architectural design flaws, and recommended security mitigations.

6: - Threat modeling **must** be exercised within the team including, product owners, architects, security champions and security testers.

7: - The Designer **should** avoid lengthy workshops and overly detailed lists of low – relevant threats.

8: - The Designer **should** perform threat modeling iteratively to align to more iterative development paradigms.

6.1.2.2. Threat Assessment (TA2)

- ✓ Inventories' risk profiles
- ✓ Standardize and scale threat modeling

Purpose:

The objective of this process is to increase granularity of security requirements derived from business logic and known risks

Benefit:

- ✓ Solid understanding of the risk level of the application portfolio
- ✓ Clear expectations of the quality of threat modeling activities

Standard:

1: - The Designer **must** create an extensive and standardized way to evaluate the risk of the application, among others via their impact on information security (confidentiality, integrity, and availability of data).

2: - The Designer **must** understand the data that the application processes and what Potential privacy violations are relevant.

3: - The Designer **should** study the impact that is the application has on other applications within the organization (e.g. the application might be modifying data that was considered ready only in another context).

4: - The Designer **must** evaluate all applications within the organization, including all existing and legacy ones.

5: -The Designer **should** leverage business impact analysis to quantify and classify application risk.

6: - The Designer **must** be leveraged the classification to define the risk profile to build a centralized inventory of risk profiles and manage accountability.

7: - The Designer **must** be used a standardized threat modeling methodology for the

6.1.2.3. Threat Assessment (TA3)

- ✓ Periodic review of risk profiles
- ✓ Optimize threat modeling

Purpose:

The objective of this process is to mandate security requirements process for all software projects and third party dependencies.

Benefit:

- ✓ Timely update of the application classification in case of changes
- ✓ Assurance of continuous improvement of threat modeling activities

Standard:

1: - The Designer **should** be reviewed periodically the risk inventory to ensure the correctness of the risk evaluations of the different applications

2: - The Designer **must** apply reusable risk patterns, comprising of related threat libraries, design flaws, and security mitigations, which are created and improved based on the organization's threat models.

3: - The Designer **should** optimize the threat modeling methodology

4: - The Designer **should** capture lessons learned from threat models and use these to improve the threat modeling methodology.

6.2. Security Requirements

The Security Requirements (SR) practice focuses on security requirements that are important in the context of secure software. The first type deals with typical software-related requirements, to specify objectives and expectations to protect the service and data at the core of the application. A second type deals with requirements relative to supplier organizations that are part of the development context of the application, in particular for outsourced development. It is important to streamline the expectations in terms of secure development because outsourced development can have a significant impact on the security of the application. The security of 3rd party (technical) libraries is part of the software supply chains stream (see Secure Build), and it is not included in this practice.

6.2.1. Streams

1. **Software Requirements:** - Software requirements specify objectives and expectations to protect the service and data at the core of the application.
2. **Supplier Security:** - Supplier security deals with requirements that are relative to supplier organizations within the development context of the application, in particular for outsourced development.

6.2.2. Security Activity

6.2.2.1. Security Requirements (SR1)

- ✓ Identify security requirements
- ✓ Perform vendor assessment

Purpose:

The objective of this process is to consider security explicitly during the software requirements process.

Benefit:

- ✓ Understanding of key security requirements during development
- ✓ Transparency of security practices of the software suppliers

Standard:

1: - The Designer **should** review the functional requirements of the software project.

2: - The Designer **should** identify relevant security requirements (i.e. expectations) for the functionality by reasoning on the desired confidentiality, integrity, or availability of the service or data offered by the software project.

3: - The Designer **must** review the functionality from an attacker's perspective to understand how it could be misused.

4: - The Designer **should** carry out a vendor assessment to understand the strengths and weakness of the suppliers

5: - The Designer **must** use a basic checklist or conduct interviews to review the supplier's typical practices and deliveries.

6.2.2.2. Security Requirements (SR2)

- ✓ Standardize and integrate security requirements
- ✓ Discuss security responsibilities with suppliers

Purpose:

The objective of this process is to increase granularity of security requirements derived from business logic and known risks.

Benefit:

- ✓ Alignment of security requirements with other types of requirements
- ✓ Clearly defined security responsibilities of the software suppliers

Standard:

1: -The Designer **must** be analyzed sources of security requirements like policies and legislation, known problems within the application, and intelligence from metrics and feedback.

2: -The Designer **should** use a structured notation of security requirements across applications and an appropriate formalism that integrates well with how to specify other (functional) requirements for the project

(e.g. Extending analysis documents, writing user stories, etc.)

3: - The Designer **should** articulate concrete responsibilities and expectations from the suppliers and the organization.

4: -The Designer **must** ensure in which type the supplier responsibilities can be specific quality requirements or particular tasks, and minimal service can be detailed in a Service Level Agreement (SLA).

(e.g. perform continuous static code analysis, or perform an independent penetration test before a major release)

6.2.2.3. Security Requirements (SR3)

- ✓ Develop a security requirements framework
- ✓ Align security methodologies with suppliers

Purpose: -

The objective of this process is to mandate security requirements process for all software projects and third party dependencies.

Benefit:

- ✓ Efficient and effective handling of security requirements in the organization
- ✓ Alignment of software development practices with suppliers to limit security risks.

Standard:

1: - The Designer **should** set up a security requirement framework to help projects elicit an appropriate and complete requirement set for the project.

2: - The Designer **should** be adapted the organizational habits and culture and provide effective methodology and guidance in the elicitation and formation of requirements to develop a framework.

3: - The Designer **should** ensure the vendor/supplier has a secure SDLC that includes secure build, secure deployment, defect management, and incident management that align within the organization.

6.3. Security Architecture

The Security Architecture (SA) practice focuses on the security linked to components and technology we deal with during the architectural design of the software. Secure Architecture Design looks at the selection and composition of components that form the foundation of our solution, focusing on its security properties. Technology management looks at the security of supporting technologies used during development, deployment, and operations, such as development stacks and tooling, deployment tooling, and operating systems and tooling.

6.3.1. Streams

1. **Architecture Design:** - The design of software architecture can significantly impact the security posture of software and the use of good security practices will improve the overall design.
2. **Technology Management**: - Technologies and frameworks are the cornerstones of any software solution. The security properties of these must be looked into to ensure an appropriate security level and to anticipate any potential issues herein.

6.3.2. Security Activity

6.3.2.1. Security Architecture (SA1)

- ✓ Adhere to basic security principles
- ✓ Identify tools and technologies

Purpose: -

The objective of this process is to insert consideration of proactive security guidance into the software design process.

Benefit:

- ✓ Sets of security principles available to product teams
- ✓ Transparency of technologies that introduce security risk

Standard:

1: - Technical staff on the product team **should** use a short checklist of security principles.

2: - The Designer **should** ensure security principles include defense-in-depth, securing the weakest link, use of secure defaults, simplicity in design of security functionality, secure failure, the balance of security and usability, running with least privilege, avoidance of security by obscurity, etc.

3: - The Designer considers each principle in the context of the overall system and identifies features that can be added to bolster security at each such interface.

4: - The Designer **should** identify the most important technologies, frameworks, tools, and integrations being used for each application.

5: - The Designer **should** use the knowledge of the architect to study the development and operating environment as well as artifacts and evaluate them for their security quality and raise important findings to be managed.

6.3.2.2. Security Architecture (SA2)

- ✓ Provide Preferred security solutions
- ✓ Promote preferred tools and technologies

Purpose: -

The objective of this process is to direct the software design process toward known secure services and secure-by-default designs.

Benefit:

- ✓ Reusable security services available for product teams
- ✓ Technologies with appropriate security level available to product teams

Standard:

1: - The Designer **should** identify shared infrastructure or services with security functionality. These typically include single-sign-on services, access control or entitlements services, logging and monitoring services, or application-level firewalling

2: - The Designer **should** collect and evaluate reusable systems to assemble a list of such resources and categorize them by the security mechanism they fulfill.

3: - If the Designer identifies multiple resources exist in each category, select and standardize on one or more shared service per category.

4: - For each selected service, the Designer **should** create design guidance for product teams to understand how to integrate with the system. Make the guidance available through training, mentorship, guidelines, and standards.

5: - The Designer **should** establish a set of best practices representing sound methods of implementing security functionality (e.g. a single-sign-on subsystem, a cross-tier delegation model, separation of duties authorization model, a centralized logging pattern, etc.)

6: - The Designer **should** identify commonly used technologies, frameworks, and tools in use across software projects in the organization, whereby the focus on capturing the high-level technologies.

7: - The Designer **should** create a list and share it across the development organization as recommended technologies. When selecting them, consider incident history, the track record for responding to vulnerabilities, appropriateness of functionality for the organization, excessive complexity in the usage of the third-party component, and sufficient knowledge within the organization.

8: - The Designer **should** perform a periodic review of these technologies for security and appropriateness.

6.3.2.3. Security Architecture (SA3)

- ✓ Build reference architectures
- ✓ Enforce the use of recommended technologies

Purpose: -

The objective of this process is to formally control the software design process and validate utilization of secure components.

Benefit:

- ✓ Full transparency of quality and usability of centrally provided security solutions

- ✓ Limited attack surface due to usage of vetted technologies

Standard:

1: - The Designer **must** build a set of reference architectures that select and combine a verified set of security components to ensure a proper design of security

2: - The Designer **should** continuously maintain and improve the reference architecture based on new insights in the organization and within the community

3: - The Designer **must** monitor weaknesses or gaps in the set of security solutions available in your organization continuously in the context of discussions on architecture, development, or operations.

4: - The Designer **should** ensure reference architectures may materialize into a set of software libraries and tools upon which project teams build their software

5: - For all Proprietary development (in-house or acquired), the Designer **should** impose and monitor the use of standardized technology

6: - The Designer **should** identify the use of non-recommended technologies to determine if there are gaps in recommendations versus the organization's need

7: - The Designer **must** examine unused or incorrectly used design patterns and reference platform modules to determine if updates are needed.

7. Implementation or Coding

This business function contains the following three security practices.

7.1. Secure Build

The Secure Build (SB) practice emphasizes the importance of building software in a standardized, repeatable manner, and of doing so using secure components, including 3rd party software dependencies.

The first stream focuses on removing any subjectivity from the build process by striving for full automation. An automated build pipeline can include additional automated security checks such as SAST and DAST to gain further assurance and flag security regressions early by failing the build, for example.

The second stream acknowledges the prevalence of software dependencies in modern applications. It aims to identify them and track their security status to contain the impact of their insecurity on an otherwise secure application. In an advanced form, it applies similar security checks to software dependencies as to the application itself.

7.1.1. Streams

3. **Build Process: -** A consistent build process ensures the software thatis deploying is predictable and directly linked to the source code. Furthermore, a developer can take advantage of the software build process for various security activities.

4. **Software Dependencies: -** External libraries are a significant part of modern software. The activities in this stream help create a view of external libraries and ensure their security strength is adequate.

7.1.2. Security Activity

7.1.2.1. Build Process (BP1)

- ✓ Will define a consistent build process

- ✓ Identify application dependencies

Purpose:

- ✓ The objective of this process is to build process is repeatable and consistent.

Benefits:

- ✓ Limited risk of human error during build process minimizing security issues.
- ✓ Available information on known security issues in dependencies

Standard:

1: - The Developer **must** define the build process, breaking it down into a set of clear instructions to either be followed by a person or an automated tool.

2: -The build process definition describes the whole process end-to-end so that the person or tool **must** follow it consistently each time and produce the same result,

3: - The build process definition **must be** stored centrally and accessible to any tools or people.

4: - The developer **must** avoid storing multiple copies as they may become unaligned and outdated.

5: - The coder reviews any build tools, **must** ensure that they are actively maintained by vendors and up-to-date with security patches.

6: - The developer **must** harden each tool's configuration so that it is aligned with vendor guidelines and industry best practices.

7: - The developer **must** determine the value for each generated artifact that can be later used to verify its integrity, such as a signature or a hash. The developer **must** protect this value and if the artifact is signed, the private signing certificate.

8: - The developer **must** ensure that build tools are routinely patched and properly hardened.

9: - The developer **must** keep a record of all dependencies used throughout the target

production environment. This is sometimes referred to as a Bill of Materials (BOM).

10: - The developer **must** consider that different component of the application may consume entirely different dependencies. For example, if the software package is a web application, cover both the server-side application code and client-side scripts.

11: - In building these records, the developer **must** consider the various locations where dependencies might be specified such as configuration files, the project's directory on disk, a package management tool, or the actual code (e.g. via an IDE that supports listing dependencies).

12: - The developer **must** gather the following information about each dependency:

Where it is used or referenced

- ✓ Version used
- ✓ License
- ✓ Source information (link to repository, author's name, etc.)
- ✓ Support and maintenance status of the dependency

13: - The developer **must** check the records to discover any dependencies with known vulnerabilities and update or replace them accordingly.

14: -The developer **must** use the leverage of security frameworks and libraries to accomplish security goals more efficiently and accurately. But must have a good practice when the developer adds a third-party framework or library in the software.

- ✓ Use libraries and frameworks from trusted sources that are actively maintained and widely used by many applications.
- ✓ Create and maintain an inventory catalog of all the third-party libraries.
- ✓ Proactively keep libraries and components up to date.
- ✓ Use a tool like OWASP Dependency-Check and Retire.JS to identify project dependencies and check if there are any known, publicly disclosed vulnerabilities for all third-party code.

- ✓ Reduce the attack surface by encapsulating the library and expose only the required behavior into the software.

7.1.2.2. Build Process (BP2)

- ✓ Automate the build process
- ✓ Review application dependencies for security

Purpose

The objective of this process is to build process is optimized and fully integrated into the workflow.

Benefits

- ✓ Efficient build process with integrated security tools
- ✓ Full transparency of known security issues in dependencies

Standard

1: - The build process **must** automate so that builds can be executed consistently anytime.

2: - The build process **must not** typically require any intervention, further reducing the likelihood of human error.

3: - The developer **must** avoid the exposure of a build tool to the network could allow a malicious actor to tamper with the integrity of the process. This might, for example, allow malicious code to be built into software.

4: -The automated process may require access to credentials and secrets required to build the software, such as the code signing certificate or access to repositories. The developer **must** handle these with care.

5: - The developer **must** check the sign generated artifacts using a certificate that identifies the organization or business unit that built it, so you can verify its integrity.

6: - The developer **must** add appropriate automated security checks (e.g. using SAST tools) in the pipeline to leverage the automation for security benefit.

7: - The developer **must** evaluate used dependencies and establish a list of acceptable ones

approved for use within a project, team, or the wider organization according to a defined set of criteria.

8: - The developer **must** introduce a central repository of dependencies that all software can be built from.

9: - The developer **must** review used dependencies regularly to ensure that:

- ✓ They remain correctly licensed
- ✓ No known and significant vulnerabilities impacting the applications are present the dependency is still actively supported and maintained
- ✓ The developer is using a current version
- ✓ There is a valid reason to include the dependency

10: - The developer **must** react timely and appropriately to non-conformities by handling these as defects.

11: - The developer **must** consider using an automated tool to scan for vulnerable dependencies and assign the identified issues to the respective development teams.

7.1.2.3. Build Process (BP3)

- ✓ Enforce a security bassline during build
- ✓ Test application dependencies

Purpose:

The objective of this process is to build process helps prevent known defects from entering the production environment.

Benefits:

- ✓ Assurance that we build software complying with a security baseline
- ✓ Handling of security issues in dependencies comparable to those in our own code

Standard:

1: -The developer **must** define security checks suitable to be carried out during the build process, as well as minimum criteria for passing the build - these might differ

according to the risk profiles of various applications.

2: -The developer includes the respective security checks in the build and **must** enforce breaking the build process in case the predefined criteria are not met.

3: -The developer **must** trigger warnings for issues below the threshold and log these to a centralized system to track them and take actions.

4: - If sensible, the developer **may** implement an exception mechanism to bypass this behavior if the risk of a particular vulnerability has been accepted or mitigated. However, **must** ensure these cases are explicitly approved first and log their occurrence together with a rationale.

5: -If technical limitations prevent the organization from breaking the build automatically. The developer **must** ensure the same effect via other measures, such as a clear policy and regular audit.

6: -The developer **may** handle code signing on a separate centralized server which does not expose the certificate to the system executing the build. Where possible, use a deterministic method that outputs byte-for-byte reproducible artifacts.

7: - The developer **must** maintain a whitelist of approved dependencies and versions, and ensure that the build process fails upon a presence of dependency not being on the list.

8: - The developer **must** include a sign-off process for handling exceptions to this rule if sensible.

9: - The developer **must** handle all errors and exception. Exception handling is a programming concept that allows an application to respond to different error states (like network down, or database connection failed, etc.) in various ways. Handling exceptions and errors correctly is critical to making our code reliable and secure.

12: - The developer **must** implement Security Logging and Monitoring. Logging is a concept that most developers already use for debugging and diagnostic purposes. Security logging is an equally basic concept: to log security information during the

runtime operation of an application. Monitoring is the live review of application and security logs using various forms of automation. The same tools and patterns can be used for operations, debugging and security purposes.

13: - The developer **must** validate all inputs.Input validation is a programming technique that ensures only properly formatted data may enter a software system component.

14: - The developer **must** perform security verification activities against dependencies on the whitelist in a comparable way to the target applications themselves (esp. using SAST and analyzing transitive dependencies).

15: - The developer **must** ensure that these checks also aim to identify possible backdoors or easter eggs in the dependencies.

16: -The developer **must** establish vulnerability disclosure processes with the dependency authors including SLAs for fixing issues.

18: - In case enforcing SLAs is not realistic (e.g. with open source vulnerabilities), the developer **must** ensure that the most probable cases are expected and able to implement compensating measures in a timely manner.

19: -The developer **must** implement regression tests for the fixes to identified issues.

20: - The developer **must** track all identified issues and their state using defect tracking system.

21: - The developer **must** integrate the build pipeline with this system to enable failing the build whenever the included dependencies contain issues above a defined criticality level.

1: - The developer **must** define security checks suitable to be carried out during the build process, as well as minimum criteria for passing the build - these might differ according to the risk profiles of various applications.

2: - The developer includes the respective security checks in the build and **must** enforce

breaking the build process in case the predefined criteria are not met.

3: - The developer **must** trigger warnings for issues below the threshold and log these to a centralized system to track them and take action.

4: - If sensible, the developer **may** implement an exception mechanism to bypass this behavior if the risk of a particular vulnerability has been accepted or mitigated. However, **must** ensure these cases are explicitly approved first and log their occurrence together with a rationale.

5: - If technical limitations prevent the organization from breaking the build automatically. The developer **must** ensure the same effect via other measures, such as a clear policy and regular audit.

6: - The developer **may** handle code signing on a separate centralized server that does not expose the certificate to the system executing the build. Where possible, use a deterministic method that outputs byte-for-byte reproducible artifacts.

7: - The developer **must** maintain a white list of approved dependencies and versions, and ensure that the build process fails upon a presence of dependency not being on the list.

8: - The developer **must** include a sign-off process for handling exceptions to this rule if sensible.

9: - The developer **must** handle all errors and exceptions. Exception handling is a programming concept that allows an application to respond to different error states (like network down, or database connection failed, etc.) in various ways. Handling exceptions and errors correctly are critical to making our code reliable and secure.

12: - The developer **must** implement Security Logging and Monitoring. Logging is a concept that most developers already use for debugging and diagnostic purposes. Security logging is an equally basic concept: to log security information during the runtime operation of an application. Monitoring is the live review of application and security logs using various forms of automation. The same tools and patterns can be

used for operations, debugging, and security purposes.

13: - The developer **must** validate all inputs. Input validation is a programming technique that ensures only properly formatted data may enter a software system component.

14: - The developer **must** perform security verification activities against dependencies on the white list in a comparable way to the target applications themselves (esp. using SAST and analyzing transitive dependencies).

15: - The developer **must** ensure that these checks also aim to identify possible backdoors or eastern eggs in the dependencies.

16: -The developer **must** establish vulnerability disclosure processes with the dependency authors including SLAs for fixing issues.

18: - In case enforcing SLAs is not realistic (e.g. with open source vulnerabilities), the developer **must** ensure that the most probable cases are expected and able to implement compensating measures in a timely manner.

19: -The developer **must** implement regression tests for the fixes to identified issues.

20: - The developer **must** track all identified issues and their state using the defect tracking system.

21: - The developer **must** integrate the build pipeline with this system to enable failing the build whenever the included dependencies contain issues above a defined criticality level.

7.2. Secure Deployment

One of the final stages in delivering secure software is ensuring the security and integrity of developed applications are not compromised during deployment. The Secure Deployment (SD) practice focuses on this. To this end, the practice's first stream focuses on removing manual error by automating the deployment process as much as possible and making its success contingent upon the outcomes of integrated security verification checks. It also fosters the Separation of Duties by making adequately trained, non-

developers responsible for deployment.

The second stream goes beyond the mechanics of deployment and focuses on protecting the privacy and integrity of sensitive data, such as passwords, tokens, and other secrets, required for applications to operate in production environments. In its simplest form, suitable production secrets are moved from repositories and configuration files into adequately managed digital vaults. In more advanced forms, secrets are dynamically generated at deployment time and routine processes detect and mitigate the presence of any unprotected secrets in the environment.

7.2.1. Streams

1. **Deployment Process**: - A repeatable and consistent deployment process ensures only deploy correct software artifacts to production. It also paves the way for representative test environments before production.
2. **Secret Management: -** As the secure execution of any software system requires credentials, this stream ensures proper handling of these sensitive data elements within the organization's environment.

7.2.2. Security Activity

7.2.2.1. Secure Deployment (SD1): -

- ✓ Use a repeatable deployment process.
- ✓ Protect application secrets in configuration and code.

Purpose

The goal of this practice level is to Deployment processes are fully documented.

Benefits

- ✓ Limited risk of human error during deployment process minimizing security issues
- ✓ Defined and limited access to production secrets.

Standard

1: - The deployment team **must** define the deployment process's overall stages, breaking it down into a set of clear instructions to either be followed by a person or

automated tooling.

2: - The deployment process definition **must be** describing the whole process end-to-end so that it can be consistently followed each time to produce the same result.

3: - The definition **must be** stored centrally and accessible to all relevant personnel.

4: - The deployment team **must not** store or distribute multiple copies, some of which may become outdated.

5: - The deployment team **must** deploy applications to production either using an automated process or manually by personnel other than the developers.

6: - The deployment team **must** ensure that developers do not need direct access to the production environment for application deployment.

7: - The deployment team **must** review any deployment tools, ensuring that they are actively maintained by vendors and up to date with security patches.

8: - The deployment person **must** harden each tool's configuration so that it is aligned with vendor guidelines and industry best practices. Given that most of these tools require access to the production environment, their security is extremely critical.

9: - The deployment team **must** ensure the integrity of the tools themselves and the workflows they follow and configure access rules to these tools according to the least privilege principle.

10: - The deployment team **must** have some personnel with access to the production environment go through at least a minimum level of training or certification to ensure their competency in this matter.

11: -The developers **must** not have access to secrets or credentials for production environments.

12: - The deployment team **must** have a mechanism in place to adequately protect production secrets, for instance by

- ✓ Having specific persons adding them to relevant configuration files upon deployment

(the separation of duty principle) or

- ✓ By encrypting the production secrets contained in the configuration files upfront.

13: - The deployment team or person **must** not use production secrets in configuration files for development or testing environments, as such environments may have a significantly lower security posture.

14: The deployment team or person **must** not keep secrets unprotected in configuration files stored in code repositories.

7.2.2.2. Secure Deployment (SD2): -

- ✓ Automate deployment and integrate security checks.
- ✓ Include application secrets during deployment

Purpose

The goal of this practice level is to Deployment processes include security verification milestones.

Benefits

- ✓ Efficient deployment process with integrated security tools
- ✓ Detection of potential leakage of production secrets

Standard

1: - The deployment team **must** automate the deployment process to cover various stages so that no manual configuration steps are needed and the risk of isolated human errors is eliminated.

2: - The deployment team **must** ensure and verify that the deployment is consistent over all stages.

3: - The deployment team **must** integrate automated security checks in the deployment process, e.g. using Dynamic Analysis Security Testing (DAST) and vulnerability scanning tools. Also, verify the integrity of the deployed artifacts where this makes sense.

4: - The deployment team **must** have logged the results from these tests centrally and take any necessary actions.

5: - The deployment team **must** ensure that in case any defects are detected, relevant personnel has to be notified automatically.

6: -In case any issues exceeding predefined criticality are identified, the deployment team **must** stop or reverse the deployment either automatically, or introduce a separate manual approval workflow so that this decision is recorded, containing an explanation for the exception.

7: - The deployment team **must** have an account for and audit all deployments to all stages.

8: - The deployment team **must** have a system in place to record each deployment, including information about who conducted it, the software version that was deployed, and any relevant variables specific to the deployment.

9: - The deployment team **must** have an automated process to add credentials and secrets to configuration files during the deployment process to respective stages. This way, developers and implementer do not see or handle those sensitive values.

10: - The deployment team **must** implement checks that detect the presence of secrets in code repositories and files, and run them periodically.

11: - The deployment team **must** configure tools to look for known strings and unknown high entropy strings. In systems such as code repositories, where there is a history, including the versions in the checks.

12: - The deployment team **must** mark potential secrets that discover as sensitive values, and remove them where appropriate.

13: - If the developer cannot remove them from a historic file in a code repository, for example, the deployment team **must** refresh the value on the system that consumes the secret. This way, if an attacker discovers the secret, it will not be useful to them.

14: - The deployment team **should** make the system used to store and process the

secrets and credentials robust from a security perspective.

15: - The deployment team **must** encrypt all secrets at rest and in transit.

16: - Users who configure this system and the secrets it contains **must be** subject to the principle of least privilege. For example, a developer might need to manage the secrets for a development environment, but not a user acceptance test or production environment.

7.2.2.3. Secure Deployment (SD3): -

- ✓ Verify the integrity of deployment artifacts
- ✓ Enforce lifecycle management of application secrets

Purpose

The goal of this practice level is to Deployment process is fully automated and incorporates automated verification of all critical milestones.

Benefits

- ✓ Assured integrity of artifacts being deployed to production
- ✓ Minimized possibility and timely detection of production secret abuse

Standard

1: - The deployment team **must** take advantage of binaries being signed at the build time and include automatic verification of the integrity of software being deployed by checking their signatures against trusted certificates. This may include binaries developed and built in-house, as well as third-party artifacts.

2: - The deployment team **must** not deploy artifacts if their signatures cannot be verified, including those with invalid or expired certificates.

3: - If the list of trusted certificates includes third-party developers, the deployment team **must** check them periodically, and keep them in line with the organization's wider governance surrounding trusted third-party suppliers.

4: - The deployment team **must** manually approve the deployment at least once during

an automated deployment.

5: - Whenever a human check is significantly more accurate than an automated one during the deployment process, the deployment team **must** go for this option.

6: - The deployment team **must** implement lifecycle management for production secrets, and ensure the generation of new secrets as much as possible, and for every application instance.

7: - The use of secrets per application instance **must** ensure that unexpected application behavior can be traced back and properly analyzed. Tools can help in automatically and seamlessly updating the secrets in all relevant places upon change.

8: - The deployment team **must** ensure that all access to secrets (both reading and writing) is logged in a central infrastructure.

9: - The deployment team **must r**eview these logs regularly to identify unexpected behavior and perform proper analysis to understand why this happened.

10: - The deployment team **must** feed issues and root causes into the defect management practice to make sure that the organization will resolve any unacceptable situations.

7.3. Defect Management

The Defect Management (DM) practice focuses on collecting, recording, and analyzing software security defects and enriching them with information to drive metrics-based decisions.

The practice's first stream deals with the process of handling and managing defects to ensure released software has a given assurance level. The second stream focuses on enriching the information about the defects and deriving metrics to guide decisions about the security of individual projects and of the security assurance program as a whole.

In a sophisticated form, the practice requires formalized, independent defect management and real-time, correlated information to detect trends and influence security strategy.

7.3.1. Streams

1. **Defect tracking: -**Defect tracking manages the collection and follow-up of all potential issues in a piece of software, from architectural flaws to coding issues and run-time vulnerabilities.
2. **Metrics and Feedback: -** Defect tracking can drive the improvement of security activities within the organization through metrics and feedback.

7.3.2. Security Activity

7.3.2.1. Defect Management (DM1)

- ✓ Track security defects centrally
- ✓ Define basic defect metrics

Purpose

The goal of this practice level is to All defects are tracked within each project.

Benefits

- ✓ Transparency of known security defects impacting particular applications
- ✓ Identification of quick wins derived from available defect information

Standard

1: - The developer **must** introduce a common definition/understanding of a security defect and define the most common ways of identifying these.

2: - The developer **must do** threat assessments Penetration tests.

3: -The application **must** have an output from static and dynamic analysis scanning tools Responsible disclosure processes or bug bounties

4: - The developer **must** foster a culture of transparency and avoid blaming any teams for introducing or identifying security defects.

5: - The developer **must** have a record and track all security defects in a defined location.

6: - The developer **must** ensure that he/she be able to get an overview of all defects affecting a particular application at any single point in time.

7: - The developer **must** define and apply access rules for the tracked security defects to mitigate the risk of leakage and abuse of this information.

8: - The developer **must** introduce at least a rudimentary qualitative classification of security defects so that he/she be able to prioritize fixing efforts accordingly.

9: - The developer **must** strive for limiting duplication of information and the presence of false positives to increase the trustworthiness of the process.

10: - The developer **must** once per define period (typically at least once per year), go over his both resolved and still open recorded security defects in every team, and extract basic metrics from the available data. These might include:

- ✓ The total number of defects versus the total number of verification activities. This could give the developer an idea of whether he's looking for defects with adequate intensity and quality.
- ✓ The software components the defects reside in. This is indicative of where attention might be most required, and where security flaws might be more likely to appear in the future again.
- ✓ The type or category of the defect, which suggests areas where the development team needs further training.
- ✓ The severity of the defect, which can help the team understand the software's risk exposure.

11: - The developer **must** identify and carry out sensible quick-win activities which he can derive from the newly acquired knowledge. These might include things like a knowledge-sharing session about one particular vulnerability type or carry out / automating a security scan.

7.3.2.2. Defect Management (DM2): -

- ✓ Rate and track security defects

- ✓ Define advanced defect metrics

Purpose

The goal of this practice level is to Defect tracking used to influence the deployment process.

Benefits

- ✓ Consistent classification of security defects with clear expectations of their handling
- ✓ Improved learning from security defects in the organization

Standard

1: - The developer **must** introduce and apply a well-defined rating methodology for his security defects consistently across the whole organization, based on the probability and expected impact of the defect being exploited. This will allow him to identify applications that need higher attention and investments.

2: - In case the developer doesn't store the information about security defects centrally, **must** ensure that you're still able to easily pull the information from all sources and get an overview about "hot spots" needing your attention.

3: - The developer **must** introduce SLAs for timely fixing of security defects according to their criticality rating and centrally monitor and regularly report on SLA breaches.

4: - The developer **must** define a process for cases where it's not feasible or economical to fix a defect within the time defined by the SLAs. This should at least ensure that all relevant stakeholders have a solid understanding of the imposed risk. If suitable, employ compensating controls for these cases.

5: - Even if you don't have any formal SLAs for fixing low severity defects, the developer **must** ensure that responsible teams still get a regular overview about issues affecting their applications and understand how particular issues affect or amplify each other.

6: -The developer **must** define, collect and calculate unified metrics across the whole organization. These might include:

- ✓ Total amount of verification activities and identified defects.
- ✓ Types and severities of identified defects.
- ✓ Time to detect and time to resolve defects.
- ✓ Windows of exposure of defects being present on live systems.
- ✓ Number of regressions / reopened vulnerabilities.
- ✓ Coverage of verification activities for particular software components.
- ✓ Amount of accepted risk.
- ✓ Ratio of security incidents caused due to unknown or undocumented security defects.

7: - The developer **must** generate a regular (e.g. monthly) report for a suitable audience. This would typically reach an audience like managers and security officers and engineers.

8: -The developer **may** use the information in the report as input for his security strategy, e.g. improving training or security verification activities.

8: - The developer **must** share the most prominent or interesting technical details about security defects including the fixing strategy to other teams once these defects are fixed, e.g. in a regular knowledge sharing meeting. This will help scale the learning effect from defects to the whole organization and limit their occurrence in the future.

7.3.2.3. Defect Management (DM3)

- ✓ Enforce an SLA for defect management
- ✓ Use metrics to improve the security strategy

Purpose

The goal of this practice level is to Defect tracking across multiple components is used to help reduce the number of new defects.

Benefits

- ✓ Assurance that security defects are handled within predefined SLAs
- ✓ Optimized security strategy based on defect information

Standard

1: - The developer **must** implement automated alerting on security defects if the fixed time breaches the defined SLAs.

2: - The developer **must** ensure that these defects are automatically transferred into the risk management process and rated by a consistent quantitative methodology.

3: - The developer **must** evaluate how particular defects influence / amplify each other not only on the level of separate teams but on the level of the whole organization.

4: - The developer **must** use the knowledge of the full kill chain to prioritize, introduce and track compensating controls mitigating the respective business risks.

5: - The developer **must** integrate the defect management system with the automated tooling introduced by other practices.

- ✓ Build and Deployment: Fail the build/deployment process if security defects above certain severity affect the final artifact unless someone explicitly signs off the exception.
- ✓ Monitoring: If possible, ensure that abuse of the security defect in a production environment is recognized and alerted.

6: - The developer **must** regularly (at least once per year) revisit the defect management metrics he's collecting and compare the effort needed to collect and track these to the expected outcomes.

7: - The developer **should** make a knowledgeable decision about removing metrics that don't deliver the overall expected value.

8: - Wherever possible, the developer **must** include and automate verification activities for the quality of the collected data and ensure sustainable improvement if any differences are detected.

9: - The developer **must** aggregate the data with his threat intelligence and incident management metrics and use the results as input for other initiatives over the whole

organization, such as:

- ✓ Planning security training for various personnel
- ✓ Improvement of security verification activities for both internally and externally developed collected
- ✓ Supply chain management, e.g. carrying out security audits of partner organizations
- ✓ Monitoring of attacks against the organization infrastructure and applications Investing in security infrastructure or compensating controls Staffing the security team and setting up the security budget

8. Verification

This Security Perspective contains the following three security practices.

8.1. Architecture Assessment

The Architecture Assessment (AA) practice ensures that the application and infrastructure architecture adequately meets all relevant security and compliance requirements, and sufficiently mitigates identified security threats. The first stream focuses on verifying that the security and compliance requirements identified in the Policy & Compliance and Security Requirements practices are met, first in an ad-hoc manner, then more systematically for each interface in the system. The second stream reviews the architecture, first for mitigations against typical threats, then against the specific threats identified in the Threat Assessment practice.

In its more advanced form, the practice formalizes the security architecture review process, continuously evaluates the effectiveness of the architecture's security controls, their scalability, and strategic alignment. Identified weaknesses and possible improvements are fed back to the Security Architecture practice to improve reference architectures.

8.1.1. Streams

1. **Architecture Validation:** Architecture validation confirms the security of the software and supporting architecture by identifying application and infrastructure architecture components and verifying their provision of security

objectives and requirements.

2. **Architecture Mitigation:** Architecture mitigation focuses on ensuring that all threats identified during Threat Assessment are adequately mitigated, and existing reference architectures updated to address any unhandled threats.

8.1.2. Security Activity

8.1.2.1. Architecture Assessment (AA1)

- ✓ Assess application architecture
- ✓ Evaluate architecture for typical threats

Purpose

The goal of this practice level is to Review the architecture to ensure baseline mitigations are in place for typical risks.

Benefits

- ✓ Understanding of high-level architecture and sensible security measures
- ✓ Assures that the architecture protects against typical security threats.

Standard

1: - The tester **must** create a view of the overall architecture and examine it for the correct provision of general security mechanisms such as authentication, authorization, user and rights management, secure communication, data protection, key management, and log management.

2: - When the tester creates a view, he **must** consider the support for privacy.

3: - The assessment **should be** based on project artifacts such as architecture or design documents, or interviews with business owners and technical staff.

4: - The assessment **must** consider the infrastructure components - these are all the systems, components, and libraries (including SDKs) that are not specific to the application, but provide direct support to use or manage the application(s) in the organization.

5: - Any security-related functionality in the architecture and review **must** in its correct provision.

6: -The assessment **must be** done in an ad hoc manner, from the point of view of anonymous users, authorized users, and specific application roles.

6: - The tester **must** review the architecture for typical security threats.

7: - Security-savvy technical staff **must** conduct this analysis with input from architects, developers, managers, and business owners as needed, to ensure the architecture addresses all common threats which development teams lacking specialized security expertise may have overlooked.

8: - Typical threats in architecture **may** relate to incorrect assumptions in, or overly reliance on, the provisioning of security mechanisms such as authentication, authorization, user and rights management, secure communication, data protection, key management, and log management.

9: -Threats, on the other hand, **may** also relate to known limitations of, or issues in, technological components or frameworks that are part of the solution and for which insufficient mitigation has been put in place.

8.1.2.2. Architecture Assessment (AA2)

- ✓ Structurally verify the architecture for identified threats
- ✓ Verify the application architecture for security methodically

Purpose

The goal of this practice level is to Review the complete provision of security mechanisms in the architecture.

Benefits

- ✓ Consistent architecture review process across the organization.
- ✓ All identified threats to the application are adequately handled.

Standard

1: - The tester **must** verify that the solution architecture addresses all identified security and compliance requirements.

2: - For each interface in the application, the tester **must** iterate through the list of security and

compliance requirements and analyze the architecture for their provision.

3: - The tester **must** perform an interaction or data flow analysis to ensure that the requirements are adequately addressed over different components.

4: - The tester **must** elaborate the analysis to show the design-level features that address each requirement.

5: - The tester **must** perform this type of analysis on both internal interfaces, e.g. between tiers, as well as external ones, e.g. those comprising the attack surface.

6: - The tester **must** identify and validate important design decisions made as part of the architecture, in particular when they deviate from the available shared security solutions in the organization.

7: - The tester **must** update the findings based on changes made during the development cycle, and note any requirements that are not provided at the design level as assessment findings.

8: - The tester **must** systematically review each threat identified during the Threat Assessment activities and examine how the architecture mitigates them.

9: - The tester **must** use a standardized process for analyzing system architectures and the flow of data within them. This is typically linked to the threat model used (e.g. STRIDE) to identify the relevant security objectives which address each type of threat.

10: - For each threat, the tester **must** identify the design-level features of the architecture which counter it and assess their effectiveness in doing so.

11: - Where available, review architectural decision records to understand the architectural constraints and tradeoffs made during design. The tester **must** consider their impact along with any security assumptions on which the safe operation of the system relies and re-evaluate them.

12: - The tester **must** enrich his previously created threat model such that each threat and its estimated impact are linked to the corresponding countermeasure.

13: - The tester **must** produce a mapping document, or dashboard in a specialized tool, to make the information available and visible to the relevant stakeholders.

8.1.2.3. Architecture Assessment (AA3)

- ✓ Feed review results back to improve reference architectures
- ✓ Verify the effectiveness of security components

Purpose

The goal of this practice level is to Review the architecture effectiveness and feedback results to improve the security architecture.

Benefits

- ✓ Continuous improvement of enterprise architecture based on architecture reviews
- ✓ Assurance of effectiveness of architecture controls

Standard

1: - The tester **must** review the effectiveness of the architecture components and their provided security mechanisms in terms of alignment with the overall strategy of the organization, and scrutinize the degree of availability, scalability, and enterprise-readiness of the chosen security solutions.

2: - While tactical choices for a particular application can make sense in specific contexts, it is important to keep an eye on the bigger picture and **must** ensure the future readiness of the designed solution.

3: - The tester **must** feed any findings back into defect management to trigger further improvements to the architecture.

4: - As an organization, the tester **must** further improve the software security posture by understanding which threats remain unaddressed in the software architectures and adapting his tactics to prevent this.

5: - The tester **must** formalize a process to use recurring architecture findings as a trigger to identify the causes of gaps in the security assessment and deal with them.

6: - The tester **must** feed findings back to the Design phase by creating, or updating relevant reference architectures, existing security solutions, or organization design principles and patterns.

8.2. Requirements-driven Testing

The goal of the Requirements-driven Testing (RT) practice is to ensure that the implemented security controls operate as expected and satisfy the project's stated security requirements. It does so by incrementally building a set of security tests and regression cases and executing them regularly.

A key aspect of this practice is its attention to both positive and negative testing. The former verifies that the application's security controls satisfy stated security requirements and validates their correct functioning.

These requirements are typically functional. Negative testing addresses the quality of the implementation of the security controls and aims to detect unexpected design flaws and implementation bugs through misuse and abuse testing. In its more advanced forms, the practice promotes security stress testing, such as denial of service, and strives to continuously improve application security by consistently automating security unit tests and creating security regression tests for all bugs identified and fixed.

Although both the Requirements-driven Testing and Security Testing practices are concerned with security testing, the former focuses on verifying the correct implementation of security requirements, while the latter aims to uncover technical implementation weaknesses in an application, irrespective of requirements.

8.2.1. Streams

1. **Control Verification:** Control Verification validates that security controls and requirements are met through testing derived from requirements, and prevents the introduction of bugs into later releases through regression testing.
2. **Misuse/Abuse Testing:** Misuse/Abuse Testing leverages fuzzing, misuse/abuse cases, and the identification of any functionality or resources in the software that can be abused to identify weaknesses in features to attack an application.

8.2.2. Security Activity

8.2.2.1. Requirements-driven Testing (RT1)

- ✓ Test the effectiveness of security controls
- ✓ Perform fuzz testing

Purpose

The goal of this practice level is to opportunistically find basic vulnerabilities and other security issues.

Benefits

- ✓ Insight into behavior of the applications when dealing with unexpected input
- ✓ Verified effectiveness of the standard security controls

Standard

1: - The tester **must** conduct security tests to verify that the standard software security controls operate as expected. At a high level, this means testing the correct functioning of the confidentiality, integrity, and availability controls of the data as well as the service.

2: - Security tests at least **must** include testing for authentication, access control, input validation, encoding, and escaping data and encryption controls.

3: -The test objective **should** validate the security controls are correctly implemented.

4: -The security testing **must be** validating the relevant software security controls.

5: - The tester **must** perform control-verification security tests manually or with tools, each time the application changes its use of the controls.

6: - Techniques such as feature toggles and A/B testing **may be** used to progressively expose features to broader audiences as they are sufficiently validated.

7: - Software control verification is mandatory for all software that is part of the SSDS program.

8: - The tester **must** perform fuzzing, sending random or malformed data to the test subject in an attempt to make it crash. Fuzz testing or Fuzzing is a Black Box software testing technique,

which consists of finding implementation bugs using automated malformed or semi-malformed data injection.

9: - The tester **must** cover at least a minimum fuzzing for vulnerabilities against the main input parameters of the application. The advantage of fuzz testing is the simplicity of the test design and its lack of preconceptions about system behavior. The stochastic approach results in bugs that human eyes or structured testing would often miss. It is also one of the few means of assessing the quality of a closed system (such as a SIP phone). The simplicity of fuzzing a target is offset by the difficulty in accurately detecting and triaging crashes. Favor existing fuzzing tools and frameworks to leverage their supporting tooling.

8.2.2.2. Requirements-driven Testing (RT2)

- ✓ Define and run security abuse cases from requirements
- ✓ Define and run security test cases from requirements

Purpose

The goal of this practice level is to opportunistically find basic vulnerabilities and other security

The goal of this practice level is to perform implementation review to discover application-specific risks against the security requirements.

Benefits

- ✓ Integration of security requirements into test scenarios
- ✓ Detection of application business logic flaws

Standard

1: -From the security requirements, the tester **must** identify and implement a set of security test cases to check the software for correct functionality.

2: -To have a successful testing program, the tester **must** know the testing objectives, specified by the security requirements.

3: - The tester **must** derive security test cases for the applications in scope from the security requirements created as part of the "Security Requirements" SSDS security practice.

4: -to validate security requirements with security tests, security requirements **must be** function-driven and highlight the expected functionality (the what) and, implicitly, the implementation (the how). These requirements are also referred to as "positive requirements" since they state the expected functionality that can be validated through security tests. Examples of positive requirements include "the application will lockout the user after six failed login attempts" or "passwords need to be a minimum of six alphanumeric characters".

5: -The validation of positive requirements consists of asserting the expected functionality. The tester **may** do it by re-creating the testing conditions and running the test according to predefined inputs.

6: - The tester **may** show the results as a fail or pass condition.

7: -Often, it is most effective to use the project team's time to build application-specific test cases, and publicly available resources or purchased knowledge bases to select applicable general test cases for security.

8: -Relevant development, security, and quality assurance staff **must** review candidate test cases for applicability, efficacy, and feasibility.

9: -The tester **must** derive the test cases during the requirements and/or design phase of the functionality.

10: - Testing the security requirements **must** be part of the functional testing of the software.

11: - The tester **must** misuse and abuse cases describe unintended and malicious use scenarios of the application, describing how an attacker could do this.

12: - The tester **must** create misuse and abuse cases to misuse or exploit the weaknesses of controls in software features to attack an application.

13: - The tester **must** use abuse-case models for an application to serve as fuel for the identification of concrete security tests that directly or indirectly exploit the abuse scenarios.

14: - The tester **must** abuse functionality, sometimes referred to as a "business logic attack", depends on the design and implementation of application functions and features. An example is using a password reset flow to enumerate accounts.

15: - As part of business logic testing, the tester **must** identify the business rules that are

important for the application and turn them into experiments to verify whether the application properly enforces the business rule. For example, on a stock trading application, is the attacker allowed to start a trade at the beginning of the day and lock in a price, hold the transaction open until the end of the day, then complete the sale if the stock price has risen or cancel if the price dropped?

8.2.2.3. Requirements-driven Testing (RT3)

- ✓ Automate security requirements testing
- ✓ Perform security stress testing

Purpose

The goal of this practice level is to maintain the application security level after bug fixes, changes or during maintenance.

Benefits

- ✓ Timely and reliable detection of violations to security requirements
- ✓ Transparency of resilience against denial of service attacks

Standard

1: - The tester **must** write and automate regression tests for all identified (and fixed) bugs to ensure that these become a test harness preventing similar issues from being introduced during later releases.

2: - Security unit tests **should** verify dynamically (i.e., at run time) that the components function as expected and should validate that code changes are properly implemented.

3: - A good practice for developers is to build security test cases as a generic security test suite that is part of the existing unit testing framework.

4: - A generic security test suite **may** include security test cases to validate both positive and negative requirements for security controls such as Identity, Authentication & Access Control, Input Validation & Encoding, User and Session Management, Error and Exception Handling, Encryption, and Auditing and Logging.

5: - The tester **must** verify the correct execution of the security tests as early as possible.

6: - If feasible for example, the tester **must** consider the passing of security tests as part of merge requirements before allowing new code to enter the main codebase. Alternatively, consider their passing a requirement for validating a build.

7: - For security functional tests, the tester **must** use unit-level tests for the functionality of security controls at the software component level, such as functions, methods, or classes. For example, a test case could check input and output validation (e.g., variable sanitation) and boundary checks for variables by asserting the expected functionality of the component.

8: - Applications are particularly susceptible to denial of service attacks. The tester **must** perform denial of service and security stress testing against them in controlled conditions, preferably on application acceptance environments.

9: - Load testing tools generate synthetic traffic, allowing the tester to test the application's performance under heavy load.

10: - One important test is how many requests per second an application can handle while remaining within its performance requirements. Testing from a single IP address is still useful as it gives an indication of how many requests an attacker must generate to impact the application.

11: - Denial of service attacks typically result in application resource starvation or exhaustion. To determine if any resources can be used to create a denial of service, the tester **must** analyze each application resource to see how it can be exhausted. Prioritize actions unauthenticated use can do.

12: - The tester **must** complement overall denial of service tests with security stress tests to perform actions or create conditions that cause delays, disruptions, or failures of the application under test.

8.3. Security Testing

The Security Testing (ST) practice leverages the fact that, while automated security testing is fast and scales well to numerous applications, in-depth testing based on good knowledge of an application and its business logic is often only possible via slower, manual expert security testing. Each stream, therefore, has one approach at its core.

The first stream focuses on establishing a common security baseline to automatically detect so-called "low hanging fruit". Progressively customize the automated tests for each application and increase their frequency of execution to detect more bugs and regressions earlier than, as close as possible to their inception. The more bugs the automated processes can detect, the more time experts have to use their knowledge and creativity to focus on more complex attack vectors and ensure in-depth application testing in the second stream. As the manual review is slow and hard to scale, reviewers prioritize testing components based on their risk, recent relevant changes, or upcoming major releases. Organizations can also access external expertise by participating in bug bounty programs, for example.

Unlike the Requirements-driven testing practice which focuses on verifying that applications correctly implement their requirements, the goal of this practice is to uncover technical and business-logic weaknesses in the application and make them visible to management and business stakeholders, irrespective of requirements.

8.3.1. Streams

1. **Scalable Baseline:** Scalable baseline focuses on the use of application-specific automated testing tools that integrate security validation into the build and deploy process. The goal of this stream is to favor width (a broad spectrum of applications) over the depth of testing.

2. **Deep Understanding:** Deep understanding focuses on performing manual security testing of high-risk components, using complex attack vectors to make advanced security testing an integral part of the development process. The goal of this stream is to favor testing depth (testing rigors) over testing width (the portfolio of applications).

8.3.2. Security Activity

8.3.2.1. Security Testing (ST1)

- ✓ Perform automated security testing
- ✓ Test high risk application components manually

Purpose

The goal of this practice level is to Perform security testing (both manual and tool based) to discover security.

Benefits

- ✓ Detection of common easy-to-find vulnerabilities
- ✓ Detection of manually identifiable vulnerabilities in critical components

Standard

1: - The tester **must** use automated static and dynamic security test tools for software, resulting in more efficient security testing and higher quality results.

2: - The tester **must** progressively increase the frequency of security tests and extend code coverage.

3: - The tester **may** perform the application security testing statically, by inspecting an application's source code without running it, or dynamically by simply observing the application's behavior in response to various input conditions. The former approach is often referred to as Static Application Security Testing (SAST), the latter as Dynamic Application Security Testing (DAST).

4: - The tester **may** choose a hybrid approach, known as Interactive Application Security Testing (IAST), combines the strengths of both approaches (at the cost of additional overhead) by dynamically testing automatically instrumented applications, allowing accurate monitoring of the application's internal state in response to external input.

5: -Many security vulnerabilities are very hard to detect without carefully inspecting the source code. While this **must** be ideally performed by expert or peer review, it is a slow and expensive task. Although "noisier" and frequently less accurate than expert-led reviews, automated SAST

tools are cheaper, much faster, and more consistent than humans. A number of commercial and free tools are able to efficiently detect sufficiently important bugs and vulnerabilities in large code bases.

6: -Dynamic testing does not require application source code, making it ideal for cases where source code is not available. It also identifies concrete instances of vulnerabilities. Due to its "black-box" approach, without instrumentation, it is more likely to uncover shallow bugs. Dynamic testing tools need a large source of test data whose manual test generation is prohibitive. Many tools exist which generate suitable test data automatically, leading to more efficient security testing and higher quality results.

7: - The tester **must** select appropriate tools based on several factors, including depth and accuracy of inspection, robustness and accuracy of security test cases, available integrations with other tools, usage and cost model, etc.

8: -When selecting tools, the tester **must** use input from security-savvy technical staff as well as developers and development managers and review results with stakeholders.

9: - The tester **must** perform selective manual security testing, possibly using a combination of static and dynamic analysis tools to guide or focus the review, in order to more thoroughly analyze parts of the application, as an attacker.

10: -Automated tools are effective at finding various types of vulnerabilities but **may** never replace expert manual review.

11: -Code-level vulnerabilities in security-critical parts of software **may** have dramatically increased impact so project teams review high-risk modules for common vulnerabilities. Common examples of high-risk functionality include authentication modules, access control enforcement points, session management schemes, external interfaces, and input validators and data parsers.

12: - The tester teams **must** combine code-level metrics and focused automated scans to determine where best to focus their efforts. In practice, the activity can take many forms including pair programming and peer review, time-boxed security "pushes" involving the entire development team, or spontaneous independent reviews by members of a specialized security group.

13: -During development cycles where high-risk code **must be** changed and reviewed, development managers triage the findings and prioritize remediation appropriately with input from other project stakeholders.

8.3.2.2. Security Testing (ST2)

- ✓ Develop application-specific security test cases
- ✓ Establish a penetration testing process

Purpose

The goal of this practice level is to make security testing during development completer and more efficient through automation complemented with regular manual security penetration tests.

Benefits

- ✓ Detection of organization-specific easy-to-find vulnerabilities
- ✓ Understanding of application resilience from black-box perspective

Standard

1: - The tester **must** increase the effectiveness of automated security testing tools by tuning and customizing them for his particular technology stacks and applications. Automated security testing tools have 2 important characteristics: Their false positive rate, i.e. the non-existent bugs and vulnerabilities they incorrectly report; their false-negative rate, i.e. actual bugs and vulnerabilities which they fail to detect.

2: - As the tester matures in the use of automated testing tools, the tester **must** strive to minimize their false positive and false negative rates. This maximizes the time development teams spend reviewing and addressing real security issues in their applications, and reduces the friction typically associated with using unturned automated security analysis tools.

3: - The tester **must** start by disabling tool support for technologies and frameworks he/she does not use and target specific versions where possible. This will increase execution speed and reduce the number of spurious results reported. Rely on security tool champions or shared security teams to pilot the tools in coordination with a select group of motivated development

teams. This will identify likely false-positive findings to ignore or remove from the tools' output.

4: - The tester **must** identify specific security issues and anti-patterns and favor the best tool for detecting them.

5: - The tester **must** leverage available tool features to take application-specific and organizational coding styles, as well as technical standards into account. Many automated static analysis tools allow users to write their own rules or customize default analysis rules to the specific software interfaces in the project under test for improved accuracy and depth of coverage. For example, potentially dangerous input (aka tainted) can be marked as safe after it flows through a designated custom sanitization method.

6: - Strategically, it is better to reliably detect a limited subset of security issues via automated tooling, and incrementally extend coverage than attempting to detect all known issues immediately.

7: - Once the tools have been sufficiently tuned, they **should** be made available to more development teams. It is important to continuously monitor their perceived efficacy among development teams.

8: - In more advanced forms, machine learning techniques **may** have adopted to identify and automatically filter out likely false positives at scale.

9: - Using the set of security test cases identified for each project, the tester **must** conduct manual penetration testing to evaluate the system's performance against each case. Generally, this happens during the testing phase before release and includes both static and dynamic manual penetration testing.

10: - In cases where software cannot be realistically tested outside of production, the tester **must** use techniques such as blue-green deployments or A/B testing that can allow ring-fenced security testing in production.

11: - Penetration testing cases include both application-specific tests to check the soundness of business logic, and common vulnerability tests to check the design and implementation. Once specified, security-savvy quality assurance or development staff **must** execute security test cases.

12: - The central software security group monitors the first-time execution of security test cases for a project team to assist and coach the team security champions.

13: - Many organizations offer "Bug Bunty" programs that invite security researchers to find vulnerabilities in applications and report them responsibly in exchange for rewards. The approach allows organizations to access a bigger pool of talent, especially those lacking sufficient internal capacity or requiring additional assurance.

14: -Before release or mass deployment, stakeholders review the results of security tests and accept the risks indicated by failing security tests at release time.

15: - The tester **must** establish a concrete timeline to address the gaps over time.

16: - The tester **must** spread the knowledge of manual security testing and the results across the development team to improve security knowledge and awareness inside the organization.

8.3.2.3. Security Testing (ST1)

- ✓ Integrate security testing tools in the delivery pipeline
- ✓ Establish continuous, scalable security verification

Purpose

The goal of this practice level is to embed security testing as part of the development and deployment processes.

Benefits

- ✓ Identification of automatically identifiable vulnerabilities in earliest possible stages
- ✓ Identification of manually identifiable security issues in earliest possible stages

Standard

1: - The tester **must** project within the organization routinely run automated security tests and review results during development.

2: - The tester **must** configure security testing tools to automatically run as part of the build and deploy process to make this scalable with low overhead. Inspect findings as they occur.

3: - The tester **must be** conducting security tests as early as the requirements or design phases can be beneficial. While traditionally used for functional test cases, this type of test-driven

development approach involves identifying and running relevant security test cases early in the development cycle.

4: - With the automatic execution of security test cases, the tester **must** project enter the implementation phase with several failing tests for the non-existent functionality.

5: - Implementation **must be** complete when all the tests pass. This provides a clear, upfront goal for developers early in the development cycle, lowering the risk of release delays due to security concerns or forced acceptance of risk to meet project deadlines.

6: - The tester **must** make the results of automated and manual security tests visible via dashboards, and routinely present them to management and business stakeholders (e.g. before each release) for review. **7: -** If there are unaddressed findings that remain as accepted risks for the release, stakeholders and development managers **must** work together to establish a concrete timeframe for addressing them.

8: - The tester **must** continuously review and improve the quality of the security tests.

9: - The tester **must** consider and implement security test correlation tools to automate the matching and merging of test results from dynamic, static, and interactive application scanners into one central dashboard, providing direct input towards Defect Management.

10: - The tester **must** spread the knowledge of the created security tests and the results across the development team to improve security knowledge and awareness inside the organization.

11: - The tester **must** integrate security testing in parallel to all other development activities, including requirement analysis, software design, and construction.

12: - With the multiplicity of security tools running at every phase of development, the tester **must** be remediating security issues at a designated stage (such as pre-release testing) is no longer appropriate or desirable.

13: - Security issues **must** be quickly triaged and fixes planned in a tradeoff between risk and cost of remediation.

14: - The tester **must** continuously strive to detect issues earlier in the development lifecycle, via specific, low-friction automated tests integrated into development tools and build processes, which lowers the cost of remediation thereby increasing the likelihood of issues being quickly resolved.

15: - The tester **must** proactively improve the security testing effort integrated into the development process by adequately propagating the results of other security test activities. For example, if a security penetration test identifies issues with session management, any changes to session management should trigger explicit security tests before pushing the changes to production.

16: - Security champions and the central secure software group **must** continuously review results from automated and manual security tests during development, including these results as part of the security awareness training for the development teams.

17: - The tester **must** integrate lessons learned in overall playbooks to improve security testing as part of the organization's development.

18: - If there are unaddressed findings that remain as accepted risks for the release, stakeholders and development managers **should** work together to establish a concrete timeframe for addressing them.

9. Operations

This business function contains the following three security practices.

9.1. Incident Management

Once an organization has applications in operation, it's likely to face security incidents. In this model, we define a security incident as a breach, or the threat of an imminent breach, of at least one asset's security goals, whether due to malicious or negligent behavior. Examples of security incidents might include a successful Denial of Service (DoS) attack against a cloud application, an application user accessing the private data of another by abusing a security vulnerability, or an attacker modifying application source code. The Incident Management (IM) practice focuses on dealing with these in any organization.

Historically, many security incidents have been detected months, or even years, after the initial breach. During the "dwell time" before an incident is detected, significant damage can occur, increasing the difficulty of recovery. Our first activity stream, Incident Detection, focuses on decreasing that dwell time.

Once you have identified that you're suffering from a security incident, it's essential to

respond in a disciplined, thorough manner to limit the damage and return to normal operations as efficiently as possible. This is the focus of our second stream.

9.1.1. Streams

1. **Incident Detection: -**Incident Detection refers to the process of determining an identified security-relevant event is, in fact, a security incident. The activities in this stream focus on the organization's ability to identify security incidents when they occur, and to initiate appropriate incident response activities.
2. **Incident Response: -** Incident Response starts the moment you acknowledge and verify the existence of a security incident. Your goal is to act in a coordinated and efficient way so that further damage is limited as much as possible. The activities in this stream focus on the organization's ability to respond appropriately and effectively to reported security incidents.

9.1.2. Security Activities

9.1.2.1. Incident Management (IM1)

- ✓ Use best-effort incident detection
- ✓ Create an incident response plan

Purpose

The goal of this practice level is to Best-effort incident detection and handling.

Benefits

- ✓ Ability to detect the most obvious security incidents
- ✓ Ability to efficiently solve most common security incidents

Standard

1: - The security team **must** analyze available log data (e.g., access logs, application logs, infrastructure logs), to detect possible security incidents in accordance with known log data retention periods.

2: - In small setups, the security team **may** do this manually with the help of common

command-line tools. With larger log volumes, employ automation techniques. Even a crone job, running a simple script to look for suspicious events, is a step forward!

3: - The security team **must** analyze the logs there and employ basic log correlation principles if you send logs from different sources to a dedicated log aggregation system.

4: - Even if you don't have a 24/7 incident detection process, the security team **must** ensure that the unavailability of the responsible person (e.g., due to vacation or illness) doesn't significantly impact detection speed or quality.

5: - The security team **must** establish and share points of contact for the formal creation of security incidents.

6: - The security team **should** recognize the incident response competence as such, and define a responsible owner. Provide them the time and resources they need to keep up with the current state of incident handling best practices and forensic tooling.

7: - The security team **must** assign a single point of contact for the process, known to all relevant stakeholders.

8: - The security team **must** ensure that the point of contact knows how to reach each participant, and define on-call responsibilities for those who have them.

9: -When security incidents happen, document all actions taken. The security team **must** protect this information from unauthorized access.

9.1.2.2. Incident Management (IM2)

- ✓ Define an incident detection process
- ✓ Define and incident response process

Purpose

The goal of this practice level is to Formal incident management process in place.

Benefits

- ✓ Timely and consistent detection of expected security incidents

- ✓ Understanding and efficient handling of most security incidents

Standard

1: - The security team **must** establish a dedicated owner for the incident detection process, make clear documentation accessible to all process stakeholders, and ensure it is regularly reviewed and updated as necessary.

2: -. The security team **must** ensure employees responsible for incident detection follow this process (e.g., using training).

3: - The process typically relies on a high degree of automation, collecting and correlating log data from different sources, including application logs.

4: - The security team **may** aggregate logs in a central place, if suitable.

6: - The security team **must** periodically verify the integrity of analyzed data. If you add a new application, the security team **must** ensure the process covers it within a reasonable time.

7: - The security team **must** detect possible security incidents using an available checklist.

8: - The checklist **should** cover expected attack vectors and known or expected kill chains. The security team **must** evaluate and update it regularly.

9: - The security team **must** notify responsible staff immediately, when you determine an event is a security incident (with sufficiently high confidence), even outside business hours.

10: - The security team **must** perform further analysis, as appropriate, and start the escalation process.

11: - The security team **must** establish and document the formal security incident response process.

12: - The security team **must** ensure documentation includes information like.

- ✓ Most probable/common scenarios of security incidents and high-level instructions for handling them; for such scenarios, also use public knowledge about possibly relevant third-party incidents
- ✓ Rules for triaging each incident

- ✓ Rules for involvement of different stakeholders including senior management, Public Relations, Legal, privacy, Human Resources, external (law enforcement) authorities, and customers; specify the mandatory timeframe to do so, if needed
- ✓ The process for performing root-cause analysis and documentation of its results

13: - The security team **must** ensure a knowledgeable and properly trained incident response team is available both during and outside of business hours.

14: - The security team **must** define timelines for action and a war room.

15: - The security team **must** keep hardware and software tools up to date and ready for use anytime.

9.1.2.3. Incident Management (IM3)

- ✓ Improve the incident detection process
- ✓ Establish an incident response team

Purpose

- ✓ The goal of this practice level is to mature incident management

Benefits

- ✓ Improve the incident detection process
- ✓ Efficient incident response independent of time, location, or type of incident

Standard

1: - The security team **must** ensure process documentation includes measures for continuous process improvement.

2: - The security team **must** check the continuity of process improvement (e.g., via tracking of changes).

3: - The security team **must** ensure the checklist for suspicious event detection is correlated at least from:

- ✓ Sources and knowledge bases external to the company (e.g., new vulnerability announcements affecting the used technologies),

- ✓ Past security incidents, and
- ✓ Threat model outcomes.

4: - The security team **must** use correlation of logs for incident detection for all reasonable incident scenarios.

5: - If the log data for incident detection is not available, the security team **must** document its absence as a defect, triage, and handle it according to your established Defect Management process.

6: - The quality of the incident detection **must** not depend on the time or day of the event.

7: - If security events are not acknowledged and resolved within a specified time (e.g., 20 minutes), the security team **must** ensure further notifications are generated according to an established escalation path.

8: - The security team **must** establish a dedicated incident response team, continuously available and responsible for continuous process improvement with the help of regular RCAs.

9: - For distributed organizations, the security team **must** define and document logistics rules for all relevant locations if sensible.

10: - The security team **must** document detailed incident response procedures and keep them up to date.

11: - The security team **must** automate procedures where appropriate.

12: - The security team **must** keep all resources necessary for these procedures (e.g., separate communicating infrastructure or a reliable external location) ready to use.

13: - The security team **must** detect and correct the unavailability of these resources in a timely manner.

14: - The security team **must** carry out incident and emergency exercises are regularly. Use the results for process improvement.

15: - The security team **must** define, gather, evaluate, and act upon metrics on the incident response process, including its continuous improvement.

9.2. Environment Management

The organization's work on application security doesn't end once the application becomes operational. New security features and patches are regularly released for the various elements of the technology stack you're using until they become obsolete or are no longer supported.

Most of the technologies in any application stack are not secure by default. This is frequently intentional, to enhance backward compatibility or ease of setup. For this reason, ensuring the secure operation of the organization's technology stack requires the consistent application of secure baseline configurations to all components. The Environment Management (EM) practice focuses on keeping your environment clean and secure.

Vulnerabilities are discovered throughout the lifecycles of the technologies on which your organization relies, and new versions addressing them are released on various schedules. This makes it essential to monitor vulnerability reports and perform orderly, timely patching across all affected systems.

9.2.1. Streams

1. **Configuration Hardening: -**The activities in this stream focus on the organization's management of security-related configurations in all elements of the technology stack. The emphasis is on those elements (e.g., operating systems, containers, frameworks, services, appliances, and libraries) obtained from third parties because their architecture and design are not under the organization's control.
2. **Patching and Updating: -** The activities in this stream focus on the organization's handling of patches and updates for all elements of the technology stack. For software developed by the organization, these activities are concerned with delivering patches and updates to customers, as well as applying them to organization-managed solutions (e.g., software

as a service). For third-party elements, these activities are concerned with the organization's timely application of updates and patches received.

9.2.2. Security Activity

9.2.2.1. Environment Management (EM1)

- ✓ Use best-effort hardening
- ✓ Practice best-effort patching

Purpose

- ✓ The goal of this practice level is to Best-effort patching and hardening

Benefits

- ✓ Hardened basic configuration settings of your components
- ✓ Mitigation of well-known issues in third-party components

Standard

1: - The security team **must** understand the importance of securing the technology stacks they're using, apply a secure configuration to stack elements, based on readily available guidance (e.g., open-source projects, vendor documentation, blog articles).

2: - When teams develop configuration guidance for their applications, based on trial-and-error and information gathered by team members, the security team **must** encourage them to share their learnings across the organization.

3: - The security team **must** identify key elements of common technology stacks, and establish configuration standards for those, based on teams' experiences of what works.

4: - At this level of maturity, the security team doesn't yet have a formal process for managing configuration baselines. Configurations **may** not be applied consistently across applications and deployments, and monitoring of conformance is likely absent.

5 - The security team **must** identify applications and third-party components which

need to be updated or patched, including underlying operating systems, application servers, and third-party code libraries.

6: - At this level of maturity, identification and patching activities are best-effort and ad hoc, without a managed process for tracking component versions, available updates, and patch status. However, there **must be** high-level requirements for patching activities (e.g., testing patches before pushing to production) that may exist, and product teams **must be** achieving best-effort compliance with those requirements.

7: - Except for critical security updates (e.g., an exploit for a third-party component has been publicly released), there **must be** teams leverage maintenance windows established for other purposes to apply component patches.

8: - For software developed by the organization, there **should be** component patches are delivered to customers and organization-managed solutions only as part of feature releases.

9: - The security teams **must** share their awareness of available updates, and their experiences with patching, on an ad hoc basis.

10: - The security team **must** ensure teams can determine the versions of all components in use, to evaluate whether their products are affected by a security vulnerability when notified. However, the process for generating and maintaining component lists **may** require significant analyst effort.

9.2.2.2. Environment Management (EM2)

- ✓ Establish hardening baselines
- ✓ Formalize patch management

Purpose

The goal of this practice level is to Formal process with baselines in place

Benefits

- ✓ Consistent hardening of technology stack components in your organization

- ✓ Consistent and proactive patching of technology stack components

Standard

1: - The configuration team **must** establish configuration hardening baselines for all components in each technology stack used. To assist with consistent application of the hardening baselines, develop configuration guides for the components.

2: - The configuration team **must** require product teams to apply configuration baselines to all new systems, and existing systems when practicable.

3: - The configuration team **must** place hardening baselines and configuration guides under change management, and assign an owner to each.

4: - Owners **must** have an ongoing responsibility to keep them up-to-date, based on evolving best practices or changes to the relevant components (e.g., version updates, new features).

5: - In larger environments, the configuration team **must** derive configurations of instances from a locally maintained master, with relevant configuration baselines applied.

6: - The configuration team **must** employ automated tools for hardening configurations.

7: - The configuration team **must** develop and follow a well-defined process for managing patches to application components across the technology stacks in use.

8: - The configuration team **must** ensure processes include regular schedules for applying vendor updates, aligned with vendor update calendars (e.g., Microsoft Patch Tuesday).

9: - For software developed by the organization, the configuration team **must** deliver releases to customers and organization-managed solutions regularly (e.g., monthly), regardless of whether you are including new features.

10: - The configuration team **must** create guidance for prioritizing component

patching, reflecting your risk tolerance and management objectives.

11: - The configuration team **must** consider operational factors (e.g., the criticality of the application, severity of the vulnerabilities addressed) in determining priorities for testing and applying patches.

12: - In the event receive a notification for a critical vulnerability in a component, while no patch is yet available, the configuration team **must** triage and handle the situation as a risk management issue (e.g., implement compensating controls, obtain customer risk acceptance, or disable affected applications/features).

9.2.2.3. Environment Management (EM3)

- ✓ Perform continuous configuration monitoring
- ✓ Enforce timely patch management

Purpose

The goal of this practice level is to Conformity with continuously improving process enforced

Benefits

- ✓ Clear view on component configurations to avoid non-conformities
- ✓ Clear view on component patch state to avoid non-conformities

Standard

1: -The configuration/security team **must** actively monitor the security configurations of deployed technology stacks, performing regular checks against established baselines.

2: -The configuration/security team **must** ensure results of configuration checks are readily available, through published reports and dashboards.

3: - When we detect non-conforming configurations, the configuration/security team **must** treat each occurrence as a security finding, and the configuration/security team **must** manage corrective actions within the established Defect Management practice.

4: - Further gains **maybe** realized using automated measures, such as "self-healing" configurations and security information and event management (SIEM) alerts.

5: - As part of the process for updating components (e.g., new releases, vendor patches), the configuration/security team **must** review corresponding baselines and configuration guides, updating them as needed to maintain their relevance and accuracy.

6: -The configuration/security team **may** review other baselines and configuration guides at least annually.

7: -The configuration/security team **must** periodically review your baseline management process, incorporating feedback and lessons learned from teams applying and maintaining configuration baselines and configuration guides.

8: -The configuration/security team **must** develop and use management dashboards/reports to track compliance with patching processes and SLAs, across the portfolio.

9: -The configuration/security team **must** ensure dependency management and application packaging processes can support applying component-level patches at any time, to meet required SLAs.

10: -The configuration/security team **must** treat missed updates as security-related product defects, and manage their triage and correction in accordance with your established Defect Management practice.

9.3. Operational Management

The Operational Management (OM) practice focuses on activities to ensure security is maintained throughout operational support functions. Although these functions are not performed directly by an application, the overall security of the application and its data depends on their proper performance. Deploying an application on an unsupported operating system with unpatched vulnerabilities, or failing to store backup media securely, can make the protections built into that application irrelevant.

The functions covered by this practice include, but are not limited to: system provisioning, administration, and decommissioning; database provisioning and administration; and data backup, restore, and archival.

9.3.1. Streams

1. **Data Protection: -**The activities in this stream focus on ensuring the organization properly protect data in all aspects of their creation, handling, storage, and processing.

2. **System Decommissioning / Legacy Management: -** From the perspective of the organization as a consumer of resources, the activities in this stream focus on the identification, management, and tracking of systems, applications, application dependencies, and services that are no longer used, have reached the end of life, or are no longer actively developed or supported. Removal of unused systems and services improves manageability of the environment and reduces the organization's attack surface while affording direct and indirect cost savings (e.g., reduced license count, reduced logging volume, or reduced analyst effort).

9.3.2. Security Activities

9.3.2.1. Operational Management (OM1)

- ✓ Organize basic data protections
- ✓ Identify unused applications

Purpose

The goal of this practice level is to Foundational Practices

Benefits

- ✓ Identification of unused of software assets or components
- ✓ Understanding of sensitivity of processed data with derived quick-win measures

Standard

1: -The security team **must** understand the types and sensitivity of data stored and processed by your applications, and maintain awareness of the fate of processed data (e.g., backups, sharing with external partners).

2: - At this level of maturity, the information gathered **may be** captured in varying forms and different places; no organization-wide data catalog is assumed to exist.

3: -The security team **must** protect and handle all data associated with a given application according to protection requirements applying to the most sensitive data stored and processed.

4: - The security team **must** implement basic controls, to prevent the propagation of un-sanitized sensitive data from production environments to lower environments.

5: -The security team **must** be ensuring un-sanitized production data are never propagated to lower (non-production) environments, the security team can focus data protection policies and activities on production.

6: -The security team **must** identify unused applications on an ad hoc basis, either by chance observation or by occasionally performing a review.

7: - When you identify unused applications, process those findings for further action. If you have established a formal process for decommissioning unused applications, the security team **must** ensure teams are aware of and use it.

8: -The security team **must** manage customer/user migration from older versions of products for each product and customer/user group.

9: - When a product version is no longer in use by any customer/user group, the security team **must** discontinue support for that version.

10: - At this level of maturity you may have a large number of product versions in active use across the customer/user base, the security team**maybe** requiring significant developer effort to back-port product fixes.

9.3.2.2. Operational Management (OM2)

- ✓ Use best-effort incident detection
- ✓ Create an incident response plan

Purpose

To best-effort incident detection and handling

Benefits

- ✓ Ability to detect the most obvious security incidents
- ✓ Ability to efficiently solve most common security incidents

Standard

1: -At this maturity level, Data Protection activities **must** focus on actively managing your stewardship of data.

2: - The security team **must** establish technical and administrative controls to protect the confidentiality of sensitive data, and the integrity and availability of all data in your care, from its initial creation/receipt through the destruction of backups at the end of their retention period.

3: - The security team **must** identify the data stored, processed, and transmitted by applications, and capture information regarding their types, sensitivity (classification) levels, and storage location(s) in your data catalog.

4: - The security team **must** identify records or data elements subject to specific regulations.

5: - The security team **must** establish a single source of truth regarding the data you work with that supports a finer-grained selection of controls for their protection. Collecting this information enhances the accuracy, timeliness, and efficiency of your responses to data-related queries (e.g., from auditors, incident response teams, or customers), and supports threat modeling and compliance activities.

6: -Based on the Data Protection Policy, the security team **must** establish processes

and procedures for protecting and preserving data throughout their lifetime, whether at rest, while being processed, or in transit.

7: - The security team **must** pay particular attention to the handling and protection of sensitive data outside the active processing system, including, but not limited to: storage, retention, and destruction of backups; and the labeling, encryption, and physical protection of offline storage media.

8: -Processes and procedures **must** cover the implementation of all controls adopted to comply with regulatory, contractual, or other restrictions on storage locations, personnel access, and other factors.

9: -As part of decommissioning a system, application, or service, the security team **must** follow an established process for removing all relevant accounts, firewall rules, data, etc. from the operational environment. By removing these unused elements from configuration files, it improves the maintainability of infrastructure-as-code resources.

10: - The security team **must** follow a consistent process for timely replacement or upgrade of third-party applications, or application dependencies (e.g., operating system, utility applications, libraries), that have reached the end of life.

9: - The security team **must** engage with customers and user groups for products at or approaching the end of life, to migrate them to supported versions in a timely manner.

9.3.2.3. Operational Management (OM3)

- ✓ Use best-effort incident detection
- ✓ Create an incident response plan

Purpose

To best-effort incident detection and handling

Benefits

- ✓ Ability to detect the most obvious security incidents
- ✓ Ability to efficiently solve most common security incidents

Standard

1: -Activities at this maturity level **must** focus on automating data protection, reducing the reliance on human effort to assess and manage compliance with policies.

2: -There **must be** a focus on feedback mechanisms and proactive reviews, to identify and act on opportunities for process improvement.

3: - The security team **must** implement technical controls to enforce compliance with your Data Protection Policy, and put monitoring in place to detect attempted or actual violations.

4: - The security team **may** use a variety of available tools for data loss prevention, access control, and tracking, or anomalous behavior detection.

5: - The security team **must** regularly audit compliance with established administrative controls, and closely monitor the performance and operation of automated mechanisms, including backups and record deletions.

6: - The security team **must be** monitoring tools to quickly detect and report failures in automation, permitting you to take timely corrective action.

7: - The security team **must** review and update the data catalog regularly, to maintain its accurate reflection of your data landscape.

8: - The security team **must** regularly reviews and updates processes and procedures to maintain their alignment with your policies and priorities.

9: - The security team **must** regularly evaluate the lifecycle state and support status of every software asset and underlying infrastructure component and estimate their end-of-life.

10: - The security team **must** follow a well-defined process for actively mitigating

security risks arising as assets/components approach their end-of-life.

11: - The security team **must** regularly review and update your process, to reflect lessons learned.

12: - The security team **must** establish a product support plan, providing clear timelines for ending support on older product versions.

13: - The security team **must** limit product versions in active use to only a small number (e.g., N.x.x and N-1.x.x only).

14: - The security team **must** establish and publicize timelines for discontinuing support on prior versions, and proactively engage with customers and user groups to prevent disruption of service or support.

10.Appendix A: Acronyms

- ✓ INSA: - Information Network Security Agency
- ✓ OWASP: - Open Web Application Security Project
- ✓ SSDMS: - Secure Software Development and Management Standard
- ✓ ISO: - International Standard Organization
- ✓ CBT: - Computer Based Training
- ✓ SDLC: - Software Development Lifecycle
- ✓ PCI: - Payment Card Industry
- ✓ GDPR: - General Data Protection Regulation
- ✓ SMEs: - Small and Mid-Size Enterprises
- ✓ TA: - Threat Assessment
- ✓ SR: - Security Requirements
- ✓ KPI: - key Performance Indicator
- ✓ IAO: -Information Assurance Officers
- ✓ RCAs: -Root Cause Analysis
- ✓ SLAs: - Service Level Agreement
- ✓ DoS: -Denial of Service
- ✓ DAST: -Dynamic Application Security Testing
- ✓ SAST: - Static Application Security Testing
- ✓ IAST: - Interactive Application Security Testing
- ✓ IDE: - Integrated Development Environment
- ✓ BOM: - Bill of Materials

11.Appendix B: References

1. A Guide to the Project Management Body of Knowledge V6.0, **PMI**, 2017.
2. Agile Practice Guide, PMI, 2017.
3. Handbook of the Secure Agile Software Development Life Cycle, **Finland's University of Oulu**, 2014.
4. Certified Information Security Auditor Cert Guide, Michael Gregg & Rob Johnson, **ISACA**, 2018.
5. Certified Ethical Hacker Study Guide v10, **EC Council,** 2018.
6. Certified Information Systems Security Professional_ official study guide-John Wiley & Sons, **(ISC) ²**, 2018.
7. Certified Information Security Manager Review Manual v12.0, **ISACA**, 2014.
8. Critical Mass Cyber Security Requirement Standard v1.0, **INSA**, 2009 E.C.
9. Building Trustworthy Systems with Cisco Secure Development Lifecycle, **Cisco Systems Inc.**, January 2016.
10. Cisco Unified Communications Manager Security Guide, Release 10.0, **Cisco Systems Inc.**, May 24, 2019.
11. PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard version 3.2.1, **PCI Security Standards Council**, LLC. , July 2018.
12. Microsoft Software Development Lifecycle, **www.microsoft.com/sdl**, accessed on May 26, 2020.
13. Oracle Database Security Guide v2.0, **Oracle Corporation**, July 2012.
14. Oracle Database Security Checklist, **Oracle Corporation**, May 18th 2020.
15. OWASP Proactive Controls for Developers v3.0, **OWASP**, 2018.
16. Software Assurance Maturity Model v2.0, **OWASP**, May 1st 2020.
17. Siebel Security Hardening Guide, **Siebel CRM**, April 2016.
18. The Definitive Guide to Scrum: **The Rules of the Game**, November 2017.
19. Application security and development security technical implementation guide Version 3, Release 5, 26 July 2013 Developed by **DISA for the DoD**.

+251-11-371-71-14
Fax: +251-11-3-20-40-37
P.O.BOX: 124498
contact@insa.gov.et
www.facebook.com/INSA.ETHIOPIA
www.insa.gov.et
Addis Ababa
Ethiopia