INDEX

1. REVISION AND APPROVAL HISTORY 1

2. OVERVIEW, PURPOSE AND SCOPE 1

3. SECURE DEVELOPMENT AND MAINTENANCE 2

4. MANAGEMENT OF RECORDS MAINTAINED BASED ON THIS DOCUMENT 7

1. REVISION AND APPROVAL HISTORY

Revision No.	Description	Page(s)	Made by	Date	Approved by	Date
1	General Review	All	C.Santos	08/10/2024	Miguel Sousa	08/10/2024

2. Overview, Purpose and Scope

The purpose of this document is to define the basic rules for the secure development of software and systems.
This document is applied to all development and support of all services, architecture, software and systems that are part of the Information Security Management System.
The users of this document are all collaborators working in development and support in the Organization.

3. Secure development and maintenance

3.1 Risk assessment for the development process

For risk assessment according to the risk assessment and risk treatment methodology, the Developer Manager shall periodically carry out the assessment of the following points:

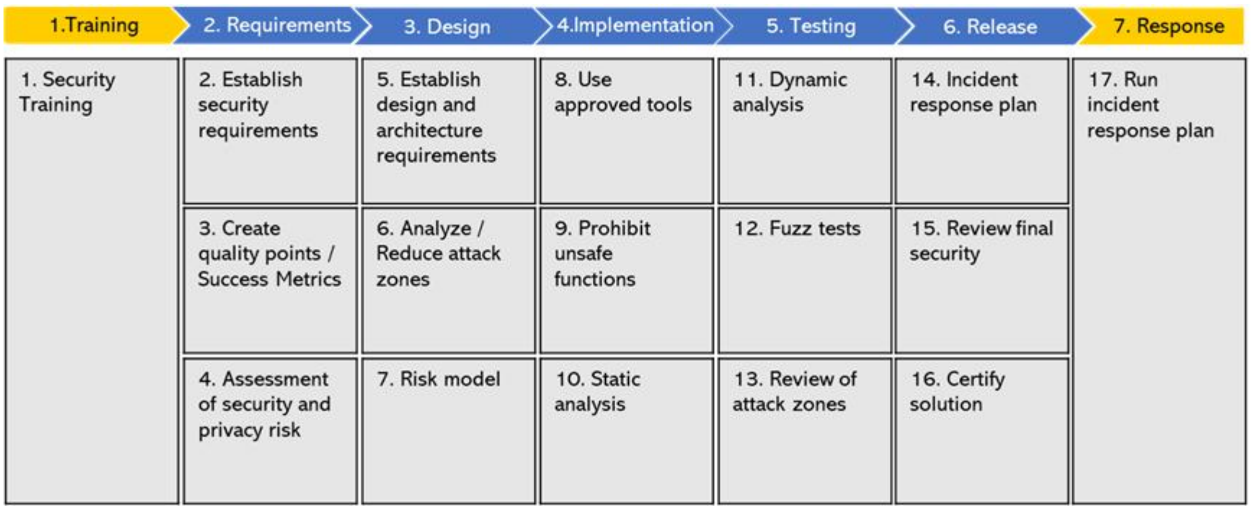
- The risks inherent in unauthorized access to the development environment;
- The risks inherent in unauthorized changes to the development environment;
- Technical vulnerabilities of IT systems used in the Organization;
- The risks that a new technology can bring if used in the Organization.

3.2 Making the development environment Secure

Resources accessible by employees in the development process should only allow access through the access credentials assigned to the employee.
Development servers must be covered by the Backup Policy.
The usage of containers (Docker), with the respective construction of images in the process of build solutions, should allow the construction of test and staging environments in an agile and fast way, also allowing to reset data for testing in a transparent way – ensuring that the test data is the same and that they comply with the privacy rules.

3.3 Secure development cycle

The Secure development cycle is designed in stages and each step-in requirement:



1.Training	2. Requirements	3. Design	4.Implementation	5. Testing	6. Release	7. Response
1. Security Training	2. Establish security requirements	5. Establish design and architecture requirements	8. Use approved tools	11. Dynamic analysis	14. Incident response plan	17. Run incident response plan
	3. Create quality points / Success Metrics	6. Analyze / Reduce attack zones	9. Prohibit unsafe functions	12. Fuzz tests	15. Review final security	
	4. Assessment of security and privacy risk	7. Risk model	10. Static analysis	13. Review of attack zones	16. Certify solution	

Training

- security base training
- Authentication
- Authorization
- Resource handling (eg job executor)
- Auditing and data logging (eg MI – JIS)
- Secure communications (eg Tokens with eBTT) To be checked case SmartBI
- Educate team on the fundamentals of building better software:
- Security design
- Risk/threat modeling
- Secure Code
- Test Secure
- Privacy
- Best practices
- Applicable legislation

Requirements

- Establish security requirements
- Define security and privacy requirements
- Make it simple to identify key checkpoints and delivery
- Minimize plan and calendar deadlines
- Create quality points / success metrics
- Set acceptable minimum levels of security and privacy
- Help teams understand the risks associated with security tasks, identify and fix *security* bugs.
- Apply standards throughout the project
- Assessment of security and privacy risks
- Examine the design based on impact/cost and mandatory requirements by law
- The team should identify which parts of the project require rich modeling/threats and security design.
- Determine the impact of product security assessment

Design

- Establish design and architecture requirements
- Consider security and privacy concerns
- Minimize the risk of calendar mis demises and reduce costs
- Analyze/reduce attack zones
- Reduce potential weaknesses from vulnerabilities
- Perform a thorough analysis of the overall surface of attacks
- Restrict access to system services
- Apply the principle of minimum privilege
- Apply layered defenses
- Risk model
- Apply an appropriate structure to risk scenarios
- Identification of vulnerabilities, risks, and mitigations more effectively and with lower cost.

Implementation

- Use approved tools
- Identify list of approved tools and verification of security points
- Options and warnings/alerts from the *compiler/linker*.
- Establish and follow good development practices
- Automate compliance with the use of good practices so that they are easily verified and with a reduced effort / cost.
- Use the latest versions of the tools
- Prohibit unsecure functions
- Review functions and APIs and prohibit those that are unsecure
- Replace them with Secure alternatives
- Static analysis
- Analyze the source code before compiling it
- Review security code
- Ensure that secure code policies are being followed

Verification

- Dynamic analysis
- Run-time scanning
- Use tools that monitor the behavior of the solution about corrupt memory, user privilege issues, ...
- Fuzz Tests
- Deliberately enter incorrect / poorly formatted / random data to break the application
- Review of attack zones
- Review surface attacks
- Identify changes resulting from design or implementation
- Review changes and risk/threat models

Deployment

- Incident response plan
- Help identify new threats that may emerge over time
- Identify contacts for security emergencies
- Establish security plans for third-party legacy code (closed components or open-source code)
- Review final security
- Review of all security activities
- Review against the model of risks / threats, results of the tools and against the quality points.
- Certify solution
- Certify software - if applicable
- Archive all relevant code and date

Response

- Run incident response plan

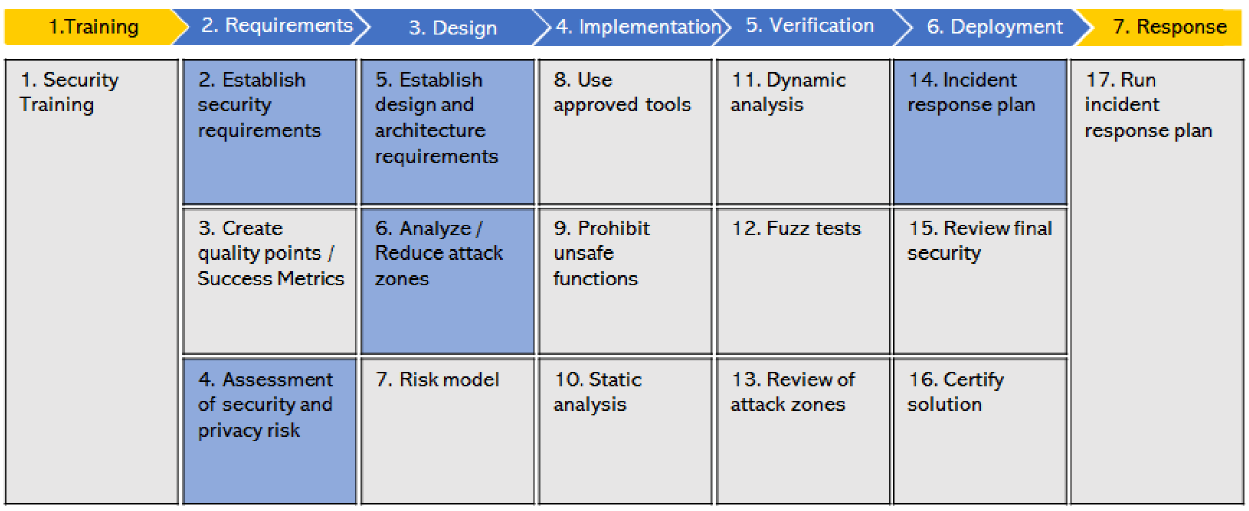
- Help the customer protect themselves from security and privacy vulnerabilities

3.4 Secure development cycle in agile methodologies

In agile methodologies development is performed in short (few weeks) cycles/sprints. The main objective of this approach is to obtain customer feedback as soon as possible and increase the quality of the software delivered.

In agile methodologies the requirements are defined at the beginning of the project with the level of details which allows them to start implementation. Requirements are adjusted to each cycle/sprint if needed.

Requirements to be checked once



1.Training	2. Requirements	3. Design	4. Implementation	5. Verification	6. Deployment	7. Response
1. Security Training	2. Establish security requirements	5. Establish design and architecture requirements	8. Use approved tools	11. Dynamic analysis	14. Incident response plan	17. Run incident response plan
	3. Create quality points / Success Metrics	6. Analyze / Reduce attack zones	9. Prohibit unsafe functions	12. Fuzz tests	15. Review final security	
	4. Assessment of security and privacy risk	7. Risk model	10. Static analysis	13. Review of attack zones	16. Certify solution	

Requirements for each cycle

These requirements should be checked in each cycle / sprint - preferably with the use of automatic tools.

1.Training	2. Requirements	3. Design	4. Implementation	5. Verification	6. Deployment	7. Response
1. Security Training	2. Establish security requirements	5. Establish design and architecture requirements	8. Use approved tools	11. Dynamic analysis	14. Incident response plan	17. Run incident response plan
	3. Create quality points / Success Metrics	6. Analyze / Reduce attack zones	9. Prohibit unsafe functions	12. Fuzz tests	15. Review final security	
	4. Assessment of security and privacy risk	7. Risk model	10. Static analysis	13. Review of attack zones	16. Certify solution	

3.4 Repository

Access to the code repository must be made by employees with their personal access credentials, and it must be organized hierarchically in groups, subgroups and projects allowing to define access at each of these levels and can be inherited by the following levels or redefined at each of the levels.
It is the responsibility of the person responsible for the Software Production Unit to create groups and subgroups for the organization of the repository.
It is the responsibility of project management to approve permissions to employees in the project.
In the assignment of permissions, the principle of least privilege should be used, increasing it in case you need to assign more privileges.
The repository of each project must contain all the code necessary to generate a distribution of the application, along with the documentation of the application.

Branch Policy

Each project must have at least five releases (four master's and one develop).
The master branches are the main one of the solutions and must be protected to avoid unwanted submissions and it is on those branches that major versions should be generated.
The development branch must be the primary during development, and you must receive merge requests from the branches of each task.
Each task must be developed in its own branch, which must be identified by feature or hotfix (whether it is, respectively, a sprint development or if an urgent fix) followed by the identification of the task in Jira, for example: feature/BARFLOW-X or feature/ID + Description + TYPE, hotfix/BFL3-X or hotfix/ID + Description + TYPE.
This branch usage policy allows for a more incisive response to critical requests, so that versions that respond more quickly to known vulnerabilities are made available.

Commit Policy

Each commit must be made following the rules defined in Conventional Commits (https://www.conventionalcommits.org/) so that automatic tools can generate Changelog.

Version control

Version control should be carried out using repository tags.
Rules will depend on customer demand/requirements.
Where is the build number:

- In the application window:

- In the ZIP packages:

Pulsar_v2201002205100739_2022-05-10_07-45-26.zip Installed application version directory:

3.5 Packages management

In sure that projects are updated frequently and not dependent on outdated versions and known vulnerabilities, you must use the package manager to share them.
Thus, fulfilling part of A9:2017 – Using Components with Known Vulnerabilities (https://www.owasp.org/index.php/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities).
Internal packages must be made available to internal resource assigned to management of release, using the following services:
.gpm – compressed compacted file generated by GPM (Global Process Manager)
.zip– compressed compacted file

3.6 Continuous integration

The development cycle should be supported by continuous integration tools, for task automation, and to avoid human errors in the process.
Static analysis tools should be, as far as possible, included in continuous integration processes. In case of tool is not possible, it will be analyzed by senior consultants into organization.

3.7 Test data protection

Sensitive data as well as data that may be related to individual people should not be available in a development and testing environment.
Where necessary and applicable, the above data may be used if they are anonymized.

4. Management of records maintained based on this document

Record name	Storage location	Personnel responsible for storage	Controls for registry protection	Retention time
List of risks related to the development process	ISMS/03 - Management Documents (MD)/ MD.09.XXX	IT Services Department and Software Production Unit	Only [the post] can access these files	3 years for lists that are no longer valid
Procedures for the Secure development cycle	ISMS/07 - Training)/ Bridge/development_lifecycle.md · master · plteam _ knowledge-base · GitLab	Software Production Unit	Only the head of the Software Production Unit can publish and edit these files	3 years for procedures that are no longer valid
Test plan	https://git.organization.com/ projects	Software Production Unit	Only the head of the Software Production Unit can publish and edit these files	3 years for testing that were run.