Version 1.2

Effective:
22.08.2022

Allianz Information Security Practice #06 Secure Software Development

Classification: Internal

Authorization:

The content of this document has been reviewed and approved as follows:

Version	Valid from	Authorized by	
		Group CISO	
1.2	22.08.2022	19.08.2022	Carsten Scholz

Table of Content

Chapter	Content	Page

Diversity disclaimer: In order to keep this document easily readable, the authors have consistently used he/his/him as a generic reference to any individual, irrespective of gender.

1. Introduction

1.1. Rationale and Scope of Application

This *Allianz Information Security Practice (short IS Practice)* contains the detailed specifications for Secure Software Development. As such it supplements the Allianz Functional Rule for Information Security (AFRIS) and forms a mandatory part of the Group Information Security Framework (GISF).

The term “should” used inside a specification indicates that it has a guideline character and can be considered optional for implementation.

If this IS Practice is in conflict with local law or regulations, the local law or regulations have priority. In this case, the responsible OE CISO should immediately inform the Group IS Function in order to agree on how these specifications should be applied.

1.2. Authorization and Updates

The Group IS Function is the owner of this IS Practice and is responsible for maintaining and updating this document. Input from Allianz Group Center and further OEs will be considered. will be reviewed at least once per year. All material changes need approval by the Group CISO.

This IS Practice is available on the Group Intranet (Allianz Connect), as part of the GISF (see https://connect.allianz.com/groups/gisf).

2. Overview

2.1. Purpose and Scope

The IS Practice consists of required or recommended security controls for the development lifecycle of software for business applications, covering measures for integrating information security during each stage, ranging from the design of the application to the implementation and the initial operation. Although the document references projects the practice also aims for line activities. This includes all roles and responsibilities of line activities.

The document is intended to develop a safe and structured approach for software development, that applies to all types of business applications and is supported by specialized, segregated development environments. This include mobile applications as well. As the majority of projects develop in a DevOps approach, the main focus of this practice lies on DevSecOps. It complements and supports the requirements defined in the Allianz Functional Rule for Information Security (AFRIS), providing more focused and implementable practices. Nevertheless it should be ensured that the Allianz Functional Rule for Information Security is taken into account for the application security lifecycle. Every requirement in this document is mandatory unless explicitly stated as optional or recommended. The Allianz Privacy Framework and its security requirements has to be considered as well.

Integrating security in the agile development lifecycle is a vital requirement in order to reduce the risk of vulnerable applications in production. All security controls and best practices must comply with the current Allianz directives and policies, as well as legal and regulatory requirements. To ensure the DevSecOps approach and to enforce security in DevOps activities it is necessary to address this for each of the three DevOps components - people, tools and processes – continuously, as they are constantly changing and evolving. This is especially valid for the tool landscape as cloud-native applications become more and more the target picture. To address especially the part “people” training of all personal participating in the development process is also mandatory in this context. In addition, the threat landscape should be reflected as well, which is inspected in detail in Annex A.

In line with Allianz’s strategy, in this document it is assumed that the development addresses cloud native or at least cloud-washed or containerized business applications built by agile or DevOps teams for a cloud ready or cloud native environment. The old way of application development (e.g. development of legacy applications that cannot be cloud-washed) is not in the scope of this document, although most of the controls are also applicable to a certain extent.

2.2. Audience

The intended audience of this document are architects, DevOps and agile teams, project leaders, business owners, application managers and security architects involved in business application development initiatives.

3. Application Development Management

3.1. Scope of Agile Software Development

As explained in section 2.1 this document covers applications[1] designed for a cloud-ready or cloud-native environment, developed with an agile approach using frequent and automated development cycles, e.g. based on CI / CD pipelines among other things. A large number of applications will be finally containerized e.g. to run them on a Kubernetes Cluster. The following figure shows a generalized development environment, being a pattern for most of the agile Allianz development environments.

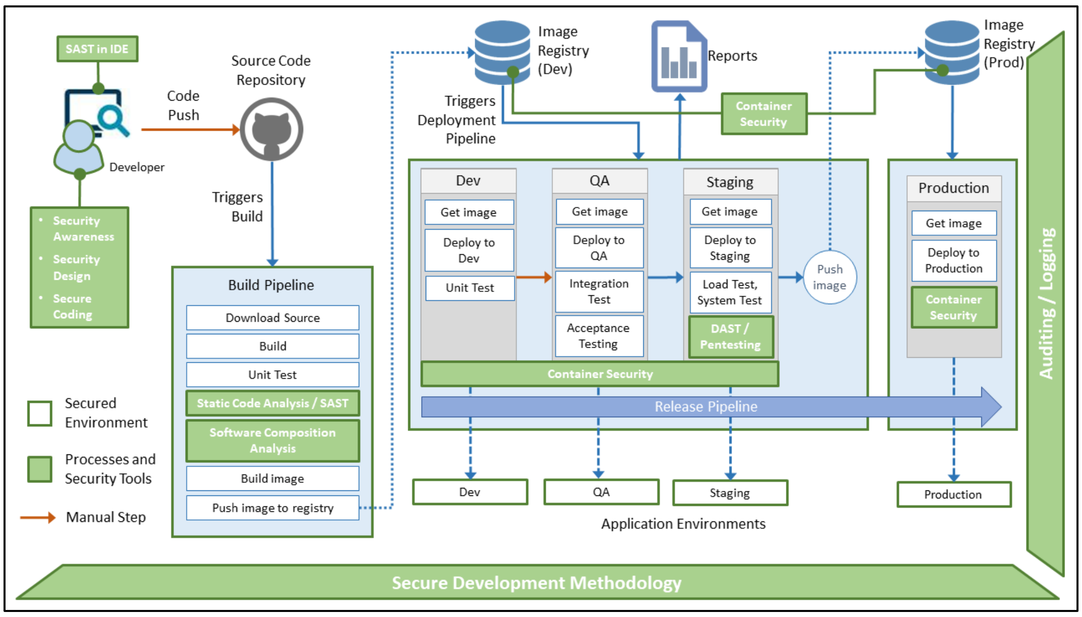


Figure 1: Reference schema of the agile software development environment

The security controls listed in the sections below address the details of the recommendations for securing this ecosystem. Because CI/CD pipelines play a vital role in this context, a generalized pipeline is shown in the next figure (figure 2) illustrating the key components and data flows. The pipeline's processes are as follows:

1. A developer commits code changes into the repository.
2. The CI Server (e.g. Jenkins) is notified about these changes.
3. As a result of this, the build step is triggered.
4. The source code necessary for building the application is requested from the repository.
5. In addition external libraries which are used to complete the code base are downloaded from the library store. After that, the code is built.
6. The built artifact is stored in an artifact repository (e.g. Sonatype Nexus, JFrog Artifactory) supporting Open Source Software.
7. In the seventh step the tests are triggered and the environment for the test session is set up.
8. If the tests are successfully executed and no errors have occurred the deployment step is triggered.
9. The artifact is downloaded from the artifact repository.
10. The artifact is deployed on the target server in production (e.g. a Kubernetes Cluster).

[1] When referencing applications in this document "business applications" are ment.

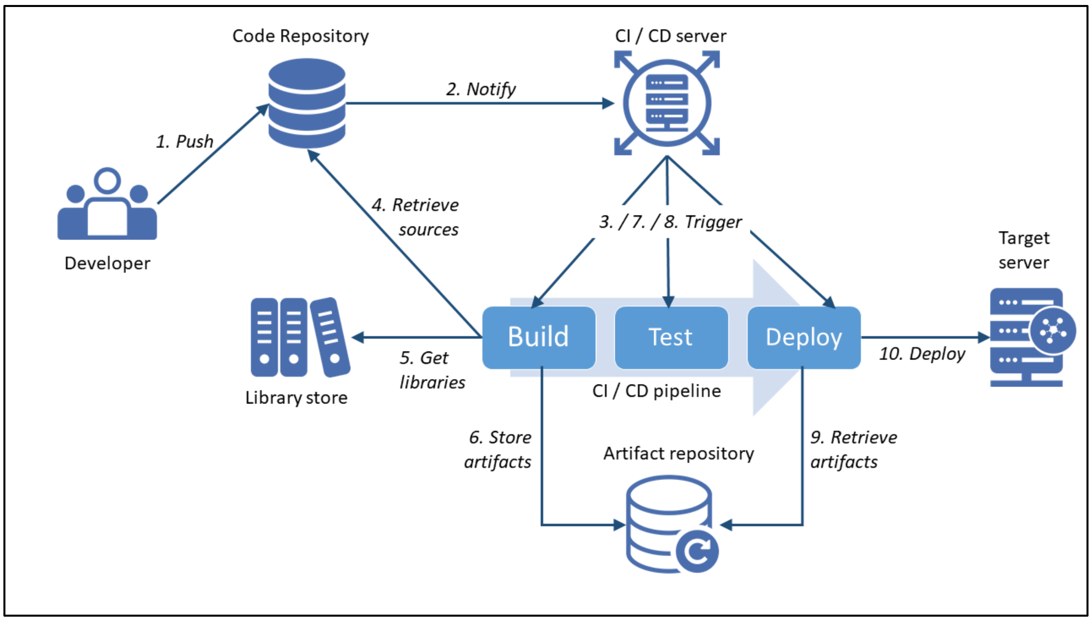


Figure 2: Generalized CI / CD pipeline

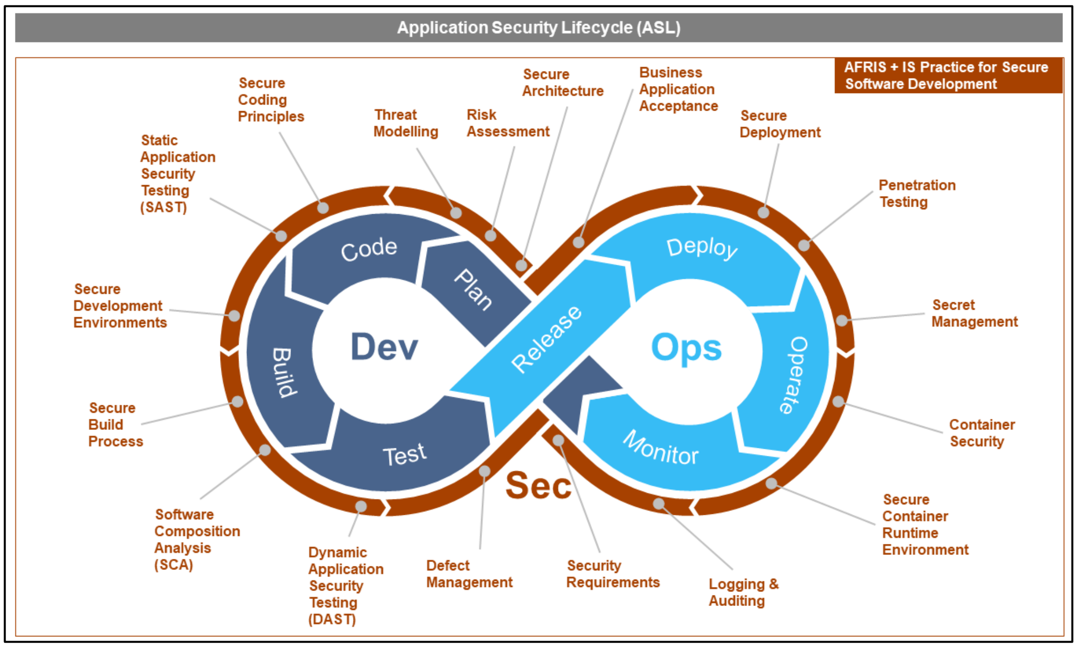
3.2. Application Development Methodology

Development activities of business applications should be conducted in accordance with a documented development methodology. This helps to ensure that applications (including those under development) meet business and information security requirements.

3.2.1. Application Security Lifecycle

The Application Security Lifecycle (ASL) is the formal guideline for developing activities in the Allianz Group. The actual specification of the ASL depends on the practice recommended by the OE.

The Project Manager must ensure that the security activities described in this document and the AFRIS (Allianz Functional Rule for Information Security) are performed along the Application Security Lifecycle. This includes mandatory activities and those recommended measures, which will be implemented as well. The following figure illustrates these activities. It must be noted that some Information Security activities are re-performed regularly or are performed continuously along certain parts of the lifecycle.



Application Security Lifecycle (ASL)
AFRIS + IS Practice for Secure Software Development
Secure Coding Principles
Threat Modelling
Risk Assessment
Secure Architecture
Business Application Acceptance
Secure Deployment
Static Application Security Testing (SAST)
Penetration Testing
Secure Development Environments
Secret Management
Secure Build Process
Container Security
Software Composition Analysis (SCA)
Secure Container Runtime Environment
Dynamic Application Security Testing (DAST)
Defect Management
Security Requirements
Logging & Auditing
Code
Plan
Build
Test
Dev
Release
Deploy
Operate
Monitor
Ops
Sec

Figure 3: Application Security Lifecycle

3.3. Securing Application Development Environments

Software development activities should be performed in specialized development environments, which are isolated from the production environments, and protected against unauthorized access. This helps to provide a secure environment for development activities, and avoid any disruption to business activity.

3.3.1. General Requirements

(1) If possible, the development environment should be located in a dedicated network and separated from the production environment as well as from other environments involved into the development (e.g. staging, testing). A direct connection from the development environment to the production environment and vice versa is not allowed and must not be available. The separation of the environments is continuously ensured and monitored.

(2) Create an Authorization Concept for the key elements of the tool chain (like source code repository, images registries, CI/CD tool, container runtime environment), which documents roles and rights (technical and non-technical). Ensure the Authorization Concept implements an SOD concept (Segregation of Duties) including adjustments of SOD conflicts and checks the least privilege principle (especially for technical users). Privileges which are not subject to the Authorization Concept are not allowed.

(3) In order to minimize attack surface on all the technical components that are part of DevOps tool chain security hardening is mandatory required for all key elements: Code Repository, CI/CD Server, Container building and orchestration platforms, containers registry as well as production clusters. At minimum it must include access restrictions to management (admin) interfaces based on "need to have" principle, up to data patch status, secure configuration settings defined for specific products. The toolchain hardening requirements must also be:

- regularly reviewed, e.g. with respect to a new version of the software
- implemented as soon as adjusted requirements are available after a review
- re-applied based on changes
- validated based on regular compliance checks.

The application owner of the corresponding software is responsible for the activities mentioned above.

3.3.2. Developer Framework

(1) Ensure source code is versioned. In order to enforce this rule, the development framework of the developer must be configured to integrate a version control. The version control and its settings are defined, documented and corresponding changes are monitored. The specific configuration is recoverable.

(2) The development workplace including the development frameworks (e.g. the IDE) should be controlled by an EDR tool, are protected by an anti-malware solution, and are hardened according to a best practice approach (e.g. CIS benchmarks, where available). This approach should be documented in a Hardening Guideline. Where possible its controls and its implementation should be integrated into global and local security operations processes and subject to a regular compliance check. In addition, the developer framework must be updated regularly, especially when security related releases become available.

(3) In order to avoid a contamination of the (local) development environment due to malicious software downloaded into the development environment (e.g. plugins including malicious artifacts) extensions to the development environment like additional plugins are only available according to a white list. The developer is able to extend his development environment with items from this list. New software extensions for the development environment and for the white list are subject to the change management including tests for malware and vulnerabilities. Ensure the responsible application owner of the application foreseen for the software extensions approves it.

(4) Where possible the development related systems are protected by an EDR tool and an anti-malware solution and are regularly scanned by a vulnerability management solution.

(5) Appropriate session expiry is important for remote connections (e.g. VPN, RDP) in order to ensure unauthorised use is not possible (e.g. in situations such as an endpoint becoming compromised or leaving the workstation unattended for extended periods) . Therefore the session timeout period for the connection should be reduced to a value no more than 8 hours.

3.3.3. Source Code Repository

(1) Access to source code available via file systems or code repositories is managed and restricted by the organization’s access management (e.g. LDAP/AD).

(2) Ensure information regarding the general location of source code and additional parts to build the application is restricted to the relevant persons with respect to the Need-to-know principle (e.g. employees which are not part of the developing projects in the OE).

(3) The source code repository is regularly scanned to identify the presence of secrets stored in the code. This can be done by using dedicated tools (e.g. using the official standard SAST Services in Allianz, GitHub Advanced Security and Checkmarx).

(4) Code should be stored in enterprise approved facilities (such as GitHub Enterprise) where the storage is encrypted by default and secured with access control features.

3.3.4. CI/CD Pipeline

(1) The access and permission to CI/CD pipeline tools is secured. The access has to be managed and restricted by the organization’s access management.

(2) All pipeline tools, repositories and registries are updated on a frequent basis. The tool-stack of the CI/CD pipeline is considered in the regular patch management processes because pipeline tools, repositories as well as registries might have weaknesses like other applications, which will result in vulnerabilities. These vulnerabilities potentially allow attackers to attack the CI/CD pipeline and modify or disrupt the development process, access sensitive information etc. This

is also valid for corresponding plugins of the tools (e.g. for Jenkins or Maven, risk of Cross-Build-Injection).

3.3.5. Image Registries

(1) The access and permission to image registries is restricted and secured. The access has to be managed and restricted by the organization's access management.

(2) Container images are only provided via a central, trusted registry. Technical measures are taken to ensure that only images from this registry are used. In case of justified need to download and deploy images from an external image repository, the image source must be verified and based on digital signature and reputation score it needs to be confirmed as trusted. Moreover, such externally provided images must be scanned to confirm they have no indicators of being compromised, that these are without vulnerabilities and malware infected content. Prior deployment of such an image, the scanning results have to be shared with security function in charge for the project to assess residual risks.

(3) Connections to registries are only made via secure connections. Therefore the connection between the endpoints are encrypted. TLS 1.3 is the mandatory protocol according to IS Practice #02 "Cryptography". Ensure that TLS 1.0 or 1.1 are not allowed, no weak cipher suites are used and signed certificate checks are enforced.

(4) Ensure obsolete images are no longer usable. Registries are regularly checked for old images which are no longer used. These images should be removed. This can be done using time information as well as labels associated with the images.

(5) In order to avoid a multitude of different registries the standard images registries of Allianz Technology must be used.

3.3.6. Container Orchestration Software

(1) Ensure the management and orchestration of the containers takes place after suitable preparation. The preparation covers at least these points:
   - Definition of a software for administration and orchestration
   - User management of the software
   - Authentication of container services to the software
   - Consideration of the requirements of continuous integration and deployment
   - Interfaces to automated methods of malware scanning and vulnerability management
   - Requirements for logging and monitoring
   - Rollout and rollback.

(2) Ensure the software used to manage and orchestrate the containers supports the least privilege principle and users are only given the minimum required permissions necessary for their tasks. In addition ensure that the authorizations as well as the owner and group assignments for the decisive (configuration) files and directories of the software for managing and orchestrating the containers and the container service are restricted to the extent necessary for the operation of the software and the containers.

(3) Critical (configuration) files and directories of the software for managing and orchestrating the containers and the container service itself are monitored and audited.

(4) The software for managing and orchestrating the containers separates the network traffic into different network segments / virtual networks, at least with regard to the protection requirements.

(5) The configuration of the orchestration tool blocks deployments of containers that process data with different protection requirements on the same host. The target environments is selected according to the purpose and protection needs of the application. Segmentation is automated to make operationalization practical and less error-prone.

(6) In case of a failure of one or more nodes the software for managing and orchestrating the containers will automatically restart all containers with high or very high availability requirements on those nodes which are still available.

(7) In case of a new target host dedicated to the orchestration platform, ensure that:

- the introduction to the platform as a new target host is carried out under a secure, traceable and logged procedure
- it has a persistent identity throughout its life cycle.

(8) Ensure the orchestration tool maintains a complete and correct inventory of all target hosts including their connection status at all times. It must be possible to isolate and disable a compromised node without interrupting or disrupting operations on other nodes.

(9) Separate environments and requisite tool stacks are used to deploy the containers in production and to manage the containers during the build process. Test environments and production environments are not linked via the tools.

(10) Ensure the configuration of the software for managing and orchestrating the containers is hardened appropriately. All services, modules, functionalities etc. which are not required are deactivated. Access to metadata about containers, nodes etc. is restricted to the minimum required permissions to avoid disclosure of information for further exploration. For hardening, existing recommendations, such as existing benchmarks of the Center for Internet Security, are used.

4. Application Development Lifecycle

4.1. Application Design

Information security requirements and data requirements for business applications under development should be considered when designing the applications. This helps to produce applications in production based on sound design principles, which have security functionality built-in and activated by default. To enforce this controls should be enabled which are incorporated easily, which would improve the application security posture and help to ensure that security weaknesses are minimised during the build process.

4.1.1. Risk Assessment

a) Application risk assessment

(1) The determination of the application risk level is mandatory for all applications. The determined risk level defines the required minimum security controls for the application. The risk level helps to estimate the potential business impact that the risk poses for the organization in case of an attack.

(2) The Project Manager is responsible for performing the risk assessment, whereas the Application Owner is responsible to re-perform the risk assessment during application operation, in case of significant changes to the application or at least annually.

(3) Ensure the risk assessment considers at least the following security aspects:

- Application accessibility on network level (e.g. internal facing, external facing etc.)
- Intended User Group (e.g. clients, partners, internal teams, all internal users etc.)
- User Authentication procedures (e.g. LDAP, SAML etc.)
- Application Complexity (e.g. different functional capabilities, different implemented technologies etc.)

(4) The risk assessment must be signed-off by the Business Owner as a mandatory requirement for the go-live of the application

(5) In case of “High” or “Very High” risk levels, the OE IS Function must review and sign-off the Risk Assessment as a mandatory requirement for the go-live of the application

b) Threat modeling

For the discussion of threats please see also Annex A.

(1) Define and use a standardized threat modeling methodology and align this on the application risk levels. For applications with a risk level “High” or “Very High” applying threat modeling before and during the design phase and a review of it before Go-Live is mandatory. The threat modeling methodology should include at least diagramming, threat identification, exploitability checks, design flaw mitigations, and how to validate the threat model artifacts. Use e.g. STRIDE as the default approach to discover threats to the applications.

(2) Use threat model diagrams which allows a detailed understanding of functional requirements and use cases of the application, main components, data sources, data flows, trust boundaries of the run time environment and identification of relevant threat events that could affect it.

(3) Design flaws identified with the threat modeling methodology are ranked according to their risks.

(4) Improve the initial application design by adding additional controls or replacing the initialy planned ones, so that the most prioritized threats are properly addressed and the corresponding high severe risks are neutralized..

(5) Update a threat model whenever the underlying technology or the runtime environment of an application changes.

(6) Practice threat modeling and support these activities with clear playbooks, templates and organization-specific examples. As threat modeling requires experience conduct it whenever there is an opportunity to apply it.

(7) Threat modeling are integrated into the SDLC and must become a mandatory part of the developer security culture.

(8) Create reusable risk patterns, comprising of related threat libraries, design flaws, and security mitigations.

(9) Review existing threat models regularly to verify that no new threats are relevant for the applications, especially when major changes on the application implicate changes in the threat landscape.

(10) Automate parts of the threat modeling process with threat modeling tools.

4.1.2. Security Requirements

a) General

(1) For all risk levels, the Project Manager must ensure that security requirements are adequate to the application complexity and to the identified risk level by working with the security architects to define and document requirements and consider Security by Design as well es Privacy by Design.

(2) Ensure the security requirements are based on secure architecture review by taking in account the applications risk profile as well as the threat modeling and risk assessment results so that at least following topics are covered. Additional security requirements are available inside OWASP Application Security Verification Standard (ASVS) (1):

- Architecture, Design and Threat Modeling
- Authentication
- Session Management
- Access Control
- Validation, Sanitization and Encoding
- Stored Cryptography
- Error Handling and Logging
- Data Protection
- Communications
- Malicious Code
- Business Logic
- File and Resources
- API and Web Service
- Configuration.

(3) Especially the session management of application should be addressed sufficiently. E.g. the timeout period of the cookies must not be defined too long and should be set to a reasonable value to avoid abuse of stolen cookies.

(4) More security requirements may be defined by the Project Manager or the OE IS function.

(5) A sign-off of the derived security requirements must be given by the Business Owner as a mandatory requirement for the go-live of the application

(6) In case of “High” and “Very High” risk levels, the OE IS Function must sign-off the derived security requirements as a mandatory requirement for the go-live of the application

b) Application

(1) Each application is classified according to the organization’s criticality levels (see Allianz Functional Rule for Information Risk Management (AFIRM)).

(2) All applications have a known and documented architecture.

(3) All applications have a defined owner and lifecycle.

c) Data

(1) Sensitive[2] data is encrypted according to the Allianz Functional Rule for Information Security (AFRIS) and IS Practice #02 “Cryptography” . The key material is not stored in the same location as the encrypted data.

(2) Ensure internal data sources are not directly accessible.

(3) Ensure that production data is not accessible from developer stages or is used in testing.

d) API

(1) APIs are classified in either external end-user, external partner or internal APIs. In order to secure them ensure that the design incorporates appropriate authentication, authorization and access logging.

(2) API security clues are hidden.

(3) Requests have to be authenticated before the authorization is validated.

(4) Encryption using secure methods with appropriate key and cipher management is enforced for the complete API communication. Where possible access should be restricted on a network level.

(5) Ensure the API design incorporates throttling and resource quotas.

(6) Validate the input and its content type.

(7) All APIs need to be versioned to ensure the content veracity of requests and expected responses.

(8) Ensure API requests are subject to auditing and logging.

4.1.3. Security Architecture

(1) Build and approve a set of reference architectures that select and combine a verified set of security components to ensure a proper design of security. Reference platforms have advantages in terms of shortening audit and security-related reviews, increasing efficiency in development, and lowering maintenance overhead.

(2) Document reference architectures and publish them on a Need-to-know basis to provide them for the stakeholders and to define patterns for automation in the form of blueprints.

(3) Continuously maintain and improve the reference architectures based on new insights in the organization and within the community.

(4) Have architects, senior developers and other technical stakeholders participate in design and creation of reference platforms. After creation, teams maintain ongoing support and updates.

(5) Reference architectures may materialize into a set of software libraries and tools upon which project teams build their software. They serve as a starting point that standardizes the configuration-driven, security-by default security approach.

4.1.4. Secure Coding Principles

[2] Information that must be protected against unauthorized access, e.g. PII data. At least confidentiality classification „Confidential“ or „Strictly Confidential“.

(1) The Project Manager must ensure that at least the secure coding principles as defined in the next table are applied when designing the application architecture. Changes to the application architecture must not be made without an approval of the Business Owner or authorized delegate.

No	Secure Coding Principle	Requirement
1	Minimize the attack surface area	Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area.
2	Establish secure defaults	Security controls should be activated by default, e.g. regarding password complexity or history.
3	Principle of least privilege	Accounts must have the least amount of privileges to perform their duties.
4	Principle of defense in depth	Where possible, more than one security control must be implemented to approach risks in different fashions[3].
5	Fail securely	Application must in the event of failure respond in a way that will cause no harm or at least a minimum harm to the application and data.
6	Don’t trust IT assets	Connected IT assets must not be trusted without proper authentication and authorization.
7	Segregation of duties	The application must observe “segregation of duties” best practices, so as to minimize the risk of conflict of interest, abuse of privileged functions or fraudulent activity.
8	Avoid security by obscurity	The security of an application must not rely upon knowledge of the source code being kept secret but on combination of strong security controls on different levels (application, access, system, network, physical level).
9	Keep security simple	Complex architectures must be avoided when a simpler approach would be faster and more straightforward.
10	Fix security issues correctly	Identified security issues must be analyzed to understand the root cause. This will ensure that the security issues are not only solved for a specific use case but in the whole application. This is particularly relevant for the reuse of code or code libraries.
11	Do not trust information provided by outside parties	Information from systems, processes, users etc. must not be trusted without proper validation.
12	Privacy by Design	Data protection through technology design. Data protection in data processing procedures is best adhered to when it is already integrated in the technology when created.

4.2. Application Build

Software build activities should be carried out in accordance with industry good practice, performed by individuals provided with adequate skills/tools, and inspected to identify unauthorized

[3] This refers to all security areas, e.g. authentication, logging, backup, recovery, etc. The Project Manager should determine where a second control is feasible, e.g. a two-factor authentication with timed access control for applications with risk level “Very High”.

modifications or changes. This ensures that applications are built correctly, able to withstand malicious attacks, and helps to ensure that no security weaknesses are introduced during the build process.

4.2.1. Secure Build

a) Build process

(1) Ensure the build process is defined over all stages with clear instructions describing the activities, which could be followed by a person or an automated tooling. The definition should describe the whole process end-to-end so that it can be consistently followed each time to produce the same result. Ensure the definition does not include any secrets.

(2) Ensure that the definition is stored centrally and accessible to all relevant personnel. Avoid publishing copies of the definition on other locations as they may become outdated.

(3) Review any build tools of the SDLC and harden them according to vendor guidelines and industry best practice. In addition ensure that they are actively maintained by vendors and up to date with security patches.

(4) Ensure a sufficient automation during the build process so that builds can be executed consistently anytime and typically require no intervention, further reducing the likelihood of human error.

(5) Harden and lock down interfaces of the build tools, such as web-based portals.

(6) 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. These secrets should be stored in a vault such as Hashicorp Vault or CyberArk Conjur. Sign generated artifacts using a certificate that identifies the organization or business unit that built it, so you can verify its integrity.

(7) Ensure the integrity of the build tools themselves and the workflows they follow, and configure access rules to these tools according to the least privilege principle.

(8) Define security checks suitable to be carried out during the build process, as well as minimum criteria for passing the build. Include the respective security checks in the build and enforce breaking the build process in case the predefined criteria are not met. The checks might differ according to the risk level of the investigated application.

(9) Trigger warnings for issues below the threshold and log these to a centralized system to track them and take actions.

(10) Implement an exception mechanism to bypass the default behaviour if the risk of a particular vulnerability has been accepted or mitigated. Ensure these cases are explicitly approved first and log their occurrence together with a rationale.

(11) If technical limitations prevent the organization from breaking the build automatically, ensure the same effect via other measures, such as a clear policy and regular audit.

b) Secure Coding Guidelines and Coding Patterns

(1) For secure application development ensure a secure coding guideline exist or is created in case it does not exist already.

(2) For all risk levels, the Project Manager must ensure that requirements defined in applicable group-wide and OE specific coding guidelines applied.

(3) Ensure secure coding guidelines are applied during the initial development of the application as well as in case of changes to the source code of an existing application.

(4) Consideration of the Secure Coding Guideline is mandatory for the SDLC and is ensured. In case of non-compliance appropriate define and plan corresponding measures. If requirements on certain technologies are missing, similar technologies are used as point of reference.

(5) Committed code is versioned and signed. The developer or the system that pushes the code versions and signs the code before the commit in order to verify the origin.

(6) Coding patterns and the code bases for frequently used functionalities are internally available for reuse by the teams and maintained.

(7) Reusable coding patterns are used if their functionality is required.

c) Source Code Analysis

(1) During the initial application development as well as during the application development after go-live e.g. for a new release, ensure that a source code analysis is performed for all risk levels at least

- every time before the source code is uploaded to the repository or whenever a pull request is triggered
- before go-live[4] of the application.

(2) During on-going application operation and maintenance after go-live the Application Owner must perform source code analysis regularly even when the code has not changed. In this case a source code analysis is required at least quarterly (for risks levels “Very High” and “High”) or annually (for risk levels “Medium”, “Low” and “Very Low”).

d) Security documentation

(1) The Project Manager ensures that the documentation of the developed application includes at least the following security relevant documentation:

- Technical details about the implemented software design and its security components
- Description of the implemented security controls and how the security requirements defined in section 4.1.2 are fulfilled.

(2) Document a full List of artifacts and libraries (SBOM, Software Bill of Material) used as dependencies when building the software, including name, version and, if applicable, vendor.

e) Images

(1) Ensure that images are obtained from trustworthy sources, that they are unchanged and free of known vulnerabilities. Trustworthy sources are defined and verified by the OE, e.g. according to the knowledge and experience of its developers, and approved before use. If images from public sources are used, each image is checked to see whether it has been modified during transport. The source is selected with respect to the support of its provider. Ensure the provider regularly checks the software with respect to security problems, fixes them and assures this service to its customers.

(2) Before an image is stored in a trusted repository, it must be inspected regarding malware and digitally signed. Signing images is highly recommended (using tools such as notary/docker sign) to ensure the origin of an image. Only images that are signed and have undergone vulnerability assessment and compliance assessment are included in the registry.

(3) In order to reduce the attack surface use basic images which are properly configured and are hardened or customized to a minimal set of functions, applications etc., thus ensuring unwanted behaviour. The choice of small base images and avoidance of superfluous packages supports future maintenance efforts and reduces the probability of found vulnerabilities. Therefore

- use slim base images instead of full-blown base images,

[4] This term refers to the stage, when an application is moved from the development or test environment to the production environment and is accessible to all authorized Allianz employees or third parties.

- avoid adding libraries and packages that are not needed for the execution of the container,
- define a user to avoid root executions,
- define a maintainer to label responsibilities.

(4) Base images are only used from trusted sources that are internally accessible, maintained and updated on a frequent basis, providing a higher level of security compared to unknown sources. Images from unknown sources are not allowed because they might be externally accessible and possibly inject vulnerable code into the development lifecycle.

(5) Protect the sources of base images from unwanted access to ensure the trustworthiness of the source. Therefore ensure the sources are only accessible from the internal network.

(6) Images are obtained and used on the basis of dedicated versions. Labels like “latest”, which are not exactly specified, should not be used.

(7) All images intended for the production stage must pass through an appropriate approval process and are in accordance with the change management process of the organization. Changes to the configuration files that define the images and data networks are also integrated into the release process.

(8) Ensure trusted images have verifiable signatures. The signature of an image is verified before execution to ensure that the image comes from a trusted source and has not been altered.

(9) Each image is checked for vulnerabilities at various check points during its lifecycle. This should be done using an automated solution. Not only the base image are considered, but all layers of the image. The check points are set at least at the following locations:

- After the download from an external source
- Before import into an internal registry
- At runtime after creation of a container instance

It should also be possible to assign vulnerabilities discovered in images to running containers.

(10) Ensure images are scanned regularly for malware embedded in the image. The scan should include malware signatures sets as well as behavior detection heuristics.

(11) Ensure sets for malware signatures are updated regularly.

(12) It must be ensured that only those images that comply with the organization-wide vulnerability and configuration guidelines are used in each phase of the build and deployment process. A corresponding configuration reference is defined in advance. Ensure that the policies cannot be bypassed or temporarily disabled.

(13) Security-relevant updates of images are implemented as soon as the updates are published. The images for the containers are recreated and new containers are instantiated from them. Images obtained from external sources should only be used if the provider also provides new versions of these images regularly and in the event of security-relevant changes.

(14) Ensure that images having a vulnerability with a CVSS rating above a specified limit (or defined by the OE IS Function – see also IS Practice #03 “Vulnerability Management”) including an outdated patch status cannot be used any longer in subsequent process steps.

(15) The results of the conformity assessment is appropriately safeguarded against loss for a reasonable period of time for subsequent investigations or comparisons.

(16) Ensure only versioned images are committed. Versioned images helps to identify and reference distinct snapshots of the development, especially regarding functional and security aspects. This is vital in order to deploy the right version of applications into production and to restore the right version when necessary.

(17) Regularly scan images for embedded end-of-life ("toxic") components. Ensure that images with identified end-of-life components or libraries cannot be used any longer in subsequent process steps and are first updated in the image registries.

f) Container and Inter-Container

(1) Before commissioning, plan how the applications operated in containers are to be separated from each other, taking into account the need for protection, the network zone concept and their risk assessment.

(2) Ensure containers are always started with reduced rights. Applications may not run under root and are assigned to specific users, i.e. run under a non-privileged account.

(3) Containers are checked for vulnerabilities at runtime. It should have been determined in advance how identified vulnerabilities are to be dealt with and what actions are to be triggered in the event of a discovery. Corresponding activities and processes should be aligned with the overall vulnerability management processes.

(4) The containers may only access the volumes and directories necessary for operation. If write permissions are not required, they must be restricted. If possible, the rights should be limited to read-only access. The private mode of volumes is used unless there is a need for shared mode.

(5) The user data accessed by the applications in the container are stored persistently outside the container.

(6) The logging data of applications running in the container are persistently stored outside the container.

(7) Ensure the file systems containing the application services' persistent data are encrypted.

(8) The transfer of data between containers over virtual or physical networks is encrypted.

(9) Ensure the Egress network traffic of containers is controlled. These controls take place at least at the trust boundaries between the virtual networks. Ensure any traffic that is not explicitly allowed is stopped.

(10) Inter-container network traffic is monitored. Control is not confined to conventional equipment at network level. Network filtering, which works at the application level ("app-aware") or acts as a container firewall, is also used. Ensure any traffic that is not explicitly allowed is blocked.

(11) Ensure a standard is defined for the container runtime configuration, which contains deterministic security targets according to the whitelist and/or blacklist principle (e.g. which users and groups can manage resources, which containers are not allowed to make external requests, which services a container is allowed to run, etc.). The compliance of the current container runtime configuration to the standard is continuously ensured.

(12) Extended policies limit the permissions of the containers and the applications running in them. The policies restrict the following accesses:

- Network connections,
- Volume and file system accesses and
- Kernel requests (syscalls).

(13) The containers and the applications are monitored with regard to their conformity with defined guidelines (see above) as well as their behavior. Behavioral profiles are generated automatically. Deviations from normal behavior and deviations from the guidelines are noticed and reported. The behaviour to be monitored should include (at least)

- Network connections,
- Volume and file system accesses and
- Kernel requests (syscalls).

(14) Behavioral anomalies can be noticed by unauthorized communication channels, unknown processes in the container, suspicious file system activity or modifications to packages, libraries, directories etc.

(15) Ensure each container executes or provides only one service at a time.

(16) Ensure all unnecessary components of the software running in the container are uninstalled. The configuration of the software is hardened appropriately.

(17) Ensure that remote administrative access can only be made to the container host and not to the services within the containers.

(18) Ensure the accounts inside the containers have no permissions on the container host. If this cannot be avoided, these authorizations should only apply to absolutely necessary data and all access has to be monitored.

(19) Only those network ports of the containers which are necessary for operation are released into the networks provided for this purpose. In principle, the ports are not released by default and are only be opened according to a whitelist principle.

(20) Ensure that in containers no tools for remote access / remote administration from the container to the host (e.g. ssh) are enabled or available.

(21) For remote administration of containers, the Container Runtime API is used (e.g. with the help of orchestration tools).

(22) Ensure all services and applications not required in the host system is uninstalled. The configuration of the host system is hardened appropriately. If possible, a container-specific operating system should be used which is specifically designed for use as a container environment and all unnecessary services and functionalities have already been deactivated.

(23) Hosts are only be used for either containers or regularly installed software. There should be no mixed use.

(24) For each container, resources on the host such as CPU, network connectivity, volatile and persistent memory is limited appropriately.

(25) Ensure all administrative access to the container service is protected by personal user accounts and strong authentication. Accesses used by the software to orchestrate the containers is also protected by separate user accounts and strong authentication.

(26) Ensure administrative access is secured by strong authentication, e.g. multifactor authentication instead of a simple password.

4.2.2. Defect Management

a) Processing of defects

(1) Ensure security defects are recorded and tracked in a defined location. Choose a location which provides an overview of all defects affecting a particular application at any single point in time.

(2) Define and apply access rules for the tracked security defects to mitigate the risk of leakage and abuse of this information.

(3) Ensure that a qualitative classification of security defects is established, helping to prioritize fixing efforts accordingly.

(4) Introduce and apply a well-defined rating methodology for the security defects, based on the probability and expected impact of the defect being exploited, allowing to identify applications which need higher attention and investments.

(5) Introduce restrictions for timely fixing of security defects according to their criticality rating and centrally monitor and regularly report on violations.

(6) For those cases where it’s not feasible or economical to fix a defect within the time defined by the restrictions a process should be defined addressing the imposed risk.

(7) Implement an automated alerting on security defects if the fix time violates the defined restrictions. Ensure that these defects are automatically transferred into the risk management process and rated by a consistent quantitative methodology.

(8) The build / deployment process should fail if security defects above certain severity affect the final artifact. Ensure an exception process is only be possible in case possible mitigation or resolving measures do not exist or all measures to mitigate or resolve the defect do not lower the severity. The exception process has to involve the IIRM process and a confirmation from the the OE IS Function.

b) Defect metrics

(1) Define a period of time where resolved and still open recorded security defects (in every team) are reviewed. Extract basic metrics from the available data, e.g.:

- the total number of defects versus total number of verification activities
- the software components the defects reside in
- the type or category of the defect
- the severity of the defect

(2) Define, collect and calculate advanced, unified defect metrics across the organizational unit. 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.

(3) Generate a regular report for a suitable audience. Use the information in the report as an input for security strategy planning, e.g. improving trainings or security verification activities.

(4) 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 or on a community platform for all developers. This will help scale the learning effect from defects to the whole organization and limit their occurrence in the future.

4.3. Application Security Testing

Software under development should be subject to security testing, using a range of attack types (including vulnerability assessments, penetration testing and access control testing). This helps to identify security weaknesses in applications and determine how applications will behave under attack conditions.

4.3.1. General Requirements and Recommendations

(1) Functional security and penetration tests are automated to the degree, that cycle times are minimized.

(2) No usage of data from production applications in development and test environments.

(3) Vulnerability assessment of the application itself and the underlying IT infrastructure is performed during the project and operational phase including at least the following methods:

- Identification of vulnerability using automated scanner (Automated vulnerability enumeration)
- Penetration tests or manual source code analysis combining automated and manual assessment methods (Manual vulnerability identification).

(4) The scope[5] of the performed vulnerability assessment activities is approved by the OE IS Function.

(5) During the development lifecycle, the Project Manager ensures static and dynamic application security testing (in form of static code analysis (see section 4.3.2) and penetration testing (see section 4.3.5)) are both performed before application go-live date according to the requirements stated below. If the results of testing indicates high risk, the applications is returned to the development team for improvement and the go live date must be postponed until a secure version can be released. Assessment of the risks and remediation of high risk findings are performed in collaboration with security function accountable for the project.

4.3.2. Static Application Security Testing (SAST)

(1) To support secure coding while developing an application the source code is scanned for known vulnerabilities and bugs with the help of a SAST tool (Static Application Security Testing). Depending on the programming languages currently supported by the tool SAST scans are implemented in the development lifecycle. The SAST tool has to provide the scanning feature for the IDE as well as for the build phase. Both capabilities are integrated into the toolchain. While the usage of SAST scan via the IDE of the developer is recommended, but not verifiable, code scans in the build process are mandatory. In case the available SAST tool does not allow to scan a specific code due to restrictions in its language support, a manual code review is recommended.

(2) In order to support static code analysis it is recommended to use the SAST service AZ Technology provides (currently GitHub Advanced Security and Checkmarx CxSAST).

(3) Define quality gates for the SAST scan analysis which rejects code to be implemented or deployed which does not fulfill predefined requirements.

(4) Create a concept for the SAST solution and its service which covers the following topics (not final):

- General description of the SAST service and its consumption
- Onboarding information
- Roles and their privileges
- Mapping of roles to organizational internal roles
- Responsibilities
- Default configuration / setup
- Definition of customized rules / metrics
- Technical details concerning customized rules
- Definition of quality gates
- Technical details concerning quality gates (implementation, validation)
- Organizational processes to alter the definitions
- Classification of vulnerabilities according to their severity
- Mapping of the organizational severity to the severity classification of the SAST tool
- General description how to process identified vulnerabilities
- Definition of timeframes, in which vulnerabilities and bugs are fixed or mitigated according to their classification
- Integration with source code repositories relevant for the AZ Group

(5) The SAST concept is available for all developing teams and projects and is reviewed and adapted regularly, especially when the underlying SAST tool provides new features due to a major upgrade or organizational requirements enforces the modification of its content.

(6) Source Code is automatically scanned by the SAST solution for known vulnerabilities and bugs in the build process.

[5] The scope defines which areas of the application and the underlying IT infrastructure is included in the vulnerability assessment.

(7) Results of the SAST scans during the build process of source code are automatically analyzed and utilized for processing the source code in the build pipeline. The quality gates defined in the SAST concept will automatically either permit or prevent the processing of the source code in the subsequent step in the build pipeline.

(8) Source code, which is scanned for the first time in the build pipeline and which does not pass the quality gate, is rejected and corrected by the initial developer.

(9) It is highly recommended that the IDE (Integrated Development Environments) integrates a plugin of the SAST tool (which is used during the build phase) in order to conduct static code analysis already in the very first phase of code creation ("Shift-Left approach"). The application owner of the IDE (Eclipse, IntelliJ etc.) is responsible for the compatibility and integration ability of the SAST-plugin to support secure coding and for the support with the technical configuration.

(10) Source Code should be automatically scanned for known vulnerabilities and bugs in the IDE of the developer.

(11) Results of source code scans are fixed or mitigated in the organization's defined timeframes. Timeframes are defined with respect to the severity of the vulnerability.

(12) Processing the results of source code scans is benchmarked against internal KPIs.

4.3.3. Software Composition Analysis (SCA)[6]

(1) Introduce a central repository of dependencies that all software can be built from.

(2) Make sure that dependencies with known high or critical vulnerabilities cannot be used to build software, e.g. by blocking downloading these dependencies within the build pipeline.

(3) Review used dependencies regularly to ensure that:

- they remain correctly licensed
- the dependency is still actively supported and maintained
- a current version is used
- there is a valid reason to include the dependency
- the dependency comes from a trustworthy source
- dependency known to be developed within Allianz are not retrieved from external/public repositories
- the dependency does not have known high or critical vulnerabilties
- if a new version of a dependency is used, it has been reviewed for obvious manipulation attempts (e.g. different authors, smaller size, obfuscated code)

(4) React timely and appropriately to non-conformities by handling these as defects.

(5) Consider using an automated tool to scan for vulnerable dependencies and assign the identified issues to the respective development teams.

(6) Code of Open Source libraries is automatically scanned for known and significant vulnerabilities and bugs impacting the applications in the build process.

(7) Define quality gates for the SCA scan analysis which rejects Open Source artifacts to be implemented or deployed which does not fulfil predefined requirements.

(8) Results of the Software Composition Analysis are fixed or mitigated in the organization's defined timeframes.

(9) Processing the results of the Software Composition Analysis is benchmarked against internal KPIs.

[6] Software Composition Analysis (SCA) is the process of automating the visibility into open source software (OSS) use for the purpose of risk management, security and license compliance.

(10) 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. Include a sign-off process for handling exceptions to this rule if sensible.

(11) Track all identified issues and their state using your defect tracking system. Integrate your build pipeline with this system to enable failing the build whenever the included dependencies contain issues above a defined criticality level.

(12) If possible, use automated systems for dependency management to ensure that your application uses the latest available secure version of all dependencies. This will allow fast and low-risk patches of dependencies, if new critical vulnerabilities are found within these dependencies.

(13) Make sure that you have a process in place to be able to create a software bills of material (SBOM) for your applications which includes all software components and the version used. For vulnerability management, you must be able to list the SBOM for the version that is currently deployed on production environments as well as on test, SI and development environments reachable from internal or external networks.

4.3.4. Dynamic and Interactive Application Security Testing (optional)

(1) Applications should be automatically tested using a DAST tool against known attacks in a black box manner and / or are tested using an IAST tool.

(2) Define security gates in the development process of an application where a DAST / IAST tool will help to detect vulnerabilities in the application.

(3) Define quality gates for the DAST / IAST analysis which rejects container images with vulnerable applications to be deployed especially in production which does not fulfill predefined requirements.

(4) Check critical applications (especially those facing the internet or dealing with sensitive data) even before releasing to production.

(5) Results of the DAST / IAST analysis are fixed or mitigated in the organization’s defined timeframes.

(6) Processing the results of the DAST / IAST analysis is benchmarked against internal KPIs.

4.3.5. Penetration Testing

(1) 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. Focus on high risk application components.

(2) When doing a manual security testing, inspect high-risk functionality like authentication modules, access control enforcement points, session management or internet facing components.

(3) Penetration tests must always be coordinated with IT security and the operations team. Engage with the OE IS testing resource to define scope and requirements for the testing including test credentials etc. Penetration test activities are conducted according to the risk level of the application:

Risk Level	Frequency
Very High or High	• when an application is ready for deployment in production (go-live), prior to release • after major design changes • at least regularly every year
Medium	• when an application is ready for deployment in production (go-live), prior to release

	• regularly every two years
Low or Very Low	• Penetration tests are recommended but not required. They become mandatory when changes to the application alter the risk level.

Especially applications reachable from the internet must undergo a penetration test.

IS Practice #04 “Penetration Testing” defines the details related to the the planning frequency and should be followed accordingly.

(4) The penetration test is delegated to an independent and trusted provider not involved in setup or maintenance of the system or service.

(5) Conduct manual penetration testing to evaluate the system’s performance against each test case, including both static and dynamic manual penetration testing.

(6) Penetration testing scope should include both application-specific tests to check attack vectors relevant for the business logic of the application but also to identify common vulnerabilities and misconfigurations of the toolchain components and the run time environment itself. The prioritization needs to be based on risk and threats assessment as well as on previous testing results and their mitigation.

(7) Prior to release or mass deployment, stakeholders review results of security tests and handle them in accordance with the organization’s risk management.

(8) IS Practice #04 “Penetration Testing” should be investigated for further processing of this topic, especially with respect to the planning frequency and its criteria.

4.3.6. Bug Bounty Program (optional)

(1) To add an additional layer of testing, it is recommended to add high and critical applications to the scope of the bug bounty program. Bug bounty hunters usually have a wider range of expertise in how to attack applications and often provide valuable insights about vulnerabilities that have not been found during the previous test cycles.

(2) The bug bounty program can only complement penetration tests, it cannot replace penetration tests due to the lack of structured testing procedure and coverage.

(3) If only some of your applications are in scope of the bug bounty program, make sure to clearly communicate which applications are in scope and which ones are out of scope.

(4) Bounty hunter activities can be done either in a test environment or on production. If you decide to allow bug bounty hunters to analyse and attack your production enviroments, make sure to handle any successful attack similar to a real attack. Follow your security incident process, redeploy the application and reset all secrets.

(5) If you provide specific test environments for bug bounty hunters, it is recommended to provide documentation to the bounty hunters on how to use the test environment application, which test-credentials to use, etc. to increase the coverage of the test activities to hard-to-reach and authenticated areas of the test-environment. For better results, also plan and provide resources for questions and support regarding the test environment and application.

(6) When participating in the bug bounty program, make sure you have the resources available to analyse and resolve findings in a timely manner and handle communication with the bug bounty hunters.

(7) While defining the scope of the bug bounty program is essential, prepare to also receive valuable findings for related IT assets (e.g. network infrastructure, build pipeline, cloud environment) and have resources available to re-route these findings to the appropriate teams.

4.4. Application Deployment

4.4.1. Container Security

(1) Container images are automatically scanned for known vulnerabilities, malware and secrets before they are becoming part of the registry, avoiding the distribution of vulnerable, contaminated images. Automation of the process is necessary to support agile development, i.e. where new versions of applications are created frequently.

(2) Container images are automatically scanned for vulnerabilities, malware and secrets before deployment.

(3) Containers are automatically scanned for known vulnerabilities and malware during runtime.

(4) Ensure the vulnerability database is frequently updated.

(5) Define metrics which allow automated checks and subsequent triggering of processes according to the image scan result.

(6) Define processes and activities how to deal with the results and e.g. who has to fix vulnerabilities.

(7) Results of scans addressing vulnerabilities in container images are fixed or mitigated in the organization’s defined timeframes.

(8) Processing the results of image scans is benchmarked against internal KPIs.

(9) Check container images against CIS benchmarks (e.g. Docker) in order to ensure standard configurations are made and scored issues are mitigated.

(10) All remote management of containers should be done through the container runtime APIs, which may be accessed via orchestration tools, or by creating remote shell sessions to the host on which the container is running.

(11) SSH and other remote administration tools designed to provide remote shells to hosts should never be enabled within containers. Containers should be run in an immutable manner to derive the greatest security benefit from their use. Enabling remote access to them via these tools implies a degree of change that violates this principle and exposes them to greater risk of network-based attack.

(12) Microservices should never have access to management interfaces, network devices or server devices.

(13) Make sure that you have a process in place to be able to create a software bills of material (SBOM) for deployed containers which includes all software components and the version used. For vulnerability management, you must be able to list the SBOM for the containers that are currently deployed on production environments as well as on test, SI and development environments reachable from internal or external networks.

4.4.2. Business Application Acceptance

(1) The Project Manager must ensure that new or changed applications are only transferred to the production environment after a documented business application acceptance was completed by the Business Owner with agreement from applicable stakeholders, e.g. Project Manager and OE IS Function.

(2) The documentation of the deployed version contains a full software bills of materials (SBOM) which includes all components and external libraries (including version number) used to build the container or install the application.

(3) Ensure the business application acceptance procedure considers at least the following security relevant aspects:

- Continuation of software maintenance is ensured
- All responsibilities for activities in the subsequent maintenance phase (especially after go-live) have been assigned
- Performance and capacity requirements are met
- Confirmation that all agreed security controls are in place and are effective
- The new application is integrated into the OE's application management processes
- Service Levels are agreed, approved and signed-off by the Business Owner and the Application Owner
- A complete technical documentation of the application including descriptions of implemented security controls is available and aligned to the requirements of the Allianz Functional Rule for Information Technology (AFRIT)
- IT Disaster Recovery procedures including error recovery and restart procedures are defined and tested.

Ensure that the acceptance criteria have been defined at the initial stage of the Application Development Lifecycle.

4.4.3. Secure Deployment

(1) Ensure the deployment process is defined over all stages with clear instructions describing the activities, which could be followed by a person or an automated tooling. The definition describes the whole process end-to-end so that it can be consistently followed each time to produce the same result.

(2) Ensure that the definition is stored centrally and accessible to all relevant personnel. Avoid publishing copies of the definition on other locations as they may become outdated.

(3) The deployment of an application to production is either carried out using an automated process or manually by personnel other than those who developed the application. Automation should be the primary option where possible. Ensure that in case of an automated process the responsible developers are not able to trigger the deployment to production in order to enforce the segregation of duties. Therefore developers in general do not need direct access to the production environment for application deployment.

(4) Changes to production environments as part of the application lifecycle are in accordance with the change management process of the organization. The security principles Segregation of Duty and Four-Eyes-Principle have to be ensured. Unauthorized changes are prevented by appropriate access management of the persons involved in the software development lifecycle. A developer is not allowed to approve his own change requests for a productive deployment.

(5) Review any deployment tools of the SDLC and harden them according to vendor guidelines and industry best practice. In addition ensure that they are actively maintained by vendors and up to date with security patches.

(6) Ensure deployment tools linked to the production environment are not linked to the development environments and vice versa. There should be separated deployment chains for development and production.

(7) 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.

(8) In order to avoid unintentional failures, deployments etc. in production ensure the competency of the personnel with access to the production environment (training, certification etc.).

(9) The deployment process is automated, so that no manual configuration steps are needed and the risk of isolated human errors is eliminated.

(10) Ensure that in case any defects are detected, relevant personnel is notified automatically. In case any issues exceeding predefined criticality are identified, 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.

(11) Audit all deployments to all stages. 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.

(12) Ensure versions of software deployed to the production environment are stored in a way that productive versions are easily recognized and restored. The history of the versions are tracked in an inventory in order to identify the correct version for a certain date.

4.4.4. Secret Management

(1) Secrets such as passwords, API keys, encryption keys and other sensitive parameters, should not be stored inside code repositories or registries.Instead, using a dedicated secure environment - vault is required. There are different solutions which allows the developer to separate secrets and key material from code and configuration files (e.g. Hash-iCorp Vault, MS Azure Key Vault, AWS Secrets Manager).

(2) Ensure neither login and authentication data nor any other secrets is stored in a source code repository.

(3) Ensure appropriate protection of production secrets addressing the threats that are encountered:

- Do not use production secrets in configuration files for development or testing environments, as such environments may have a significantly lower security posture.
- Do not keep secrets unprotected in configuration files stored in code repositories.
- Store sensitive credentials and secrets for production systems with encryption-at-rest at all times in a dedicated secret store. The usage of such tool is mandatory.
- Handle key management carefully so only personnel with responsibility for production deployments are able to access this data.
- Never keep secrets in container images or within related code.
- Transmit sensitive credentials and secrets with encryption at all times.

(4) Ensure secrets for containers are stored and managed in such a way that only authorized persons can access them. In particular, when managing images and the applications running in the images, care must be taken to ensure that the secrets are only stored in access-protected locations (see (1)). Ensure the following secrets are considered as a minimum:

- Passwords of any accounts
- Certificates
- API keys for services used by the application
- Private keys in Public-Key Authentication
- Other confidential information (e.g. about accounts).

(5) Secrets are stored outside the containers and dynamically provided on demand. They are only available to those containers which actually need them. Ensure that it is not possible to disclose secrets to containers that are out of the context.

(6) In order to avoid disclosure of secrets use an automated process to add credentials and secrets to configuration files during the deployment process to respective stages.

(7) Use appropriate tools which periodically check the presence of secrets in code repositories and files. Mark potential secrets discovered as sensitive values and remove them where appropriate.

(8) For secrets detected in a historic file in a code repository, make sure that the value on the system that consumes the secret as well as on any other system that uses or defines the secret is refreshed. In addition, also remove the secret from the source code history.

4.5. Application Operation

4.5.1. On-Going Application Management Processes

(1) The Application Owner ensures that the minimum application management processes as defined in

- the Allianz Functional Rule for Information Security (AFRIS)
- additional OE specific policies and standards as well as
- Service Level Agreements

are applied during application operation.

(2) The Application Owner ensures that security controls are established at least for

- User Access Management
- Monitoring
- Logging
- Change Management
- Release and Patch Management
- Incident Management
- Backup Management
- Recovery and Disaster Recovery
- Audit
- Availability Management

(3) The Application Owner defines the scope and frequency of the applied application management processes, e.g. regarding backup management or logging.

(4) In case of standard changes to the application, the Application Owner follows the OE Change Management process.

(5) The Application Owner is responsible to re-perform a threat assessment (see 4.1.1) including risk assessment and threat modelling during application operation, in case of significant changes to the application or at least annually. Additional threat modelling is performed in case of risk level “High” or “Very High” of the application. Proof of existence of this threat assessment is subject to the change management, which is necessary in these cases.

(6) Ensure that applications are regularly re-build and re-deployed using a fresh container image with up-to-date components and configuration, even if no new features have been added. It is recommended to re-deploy every application every six months. Budget and resources for testing the updated software stack and managing the redeployment activities must be planned for secure operations of the application.

4.5.2. Container Runtime Environment

(1) All access to control dashboards, which are capable to manage resources on behalf of the orchestration tool, are secured by organizational standards.

(2) All dashboards are only available through the internal network. If the dashboard is available on the internal network, ensure approved authentication methods are used.

(3) If a dashboard is exposed through the internet it needs to underlie a Zero-Trust-Gateway (using Multi-Factor authentication, Conditional Access-Management). If the authentication method is

token-based it has to be ensured that the token validity is kept short (expiration date). In general, exposing any dashboard through the public (internet) or the unsegmented scope of the internal network should be avoided.

(4) Ensure all API-traffic is encrypted with TLS in accordance with IS Practice #02 “Cryptography”.

(5) DevOps Teams and Platform Operator ensure that services and access to sources that contain sensitive data are encrypted and data access is logged. Data exposure has to be avoided. Any access to sensitive data such as customer data, financial data, etc. has to be performed through the internal AZ-API-Gateway or group provided architecture specifically for this connectivity such as NAT gateways for legacy connections or proxies. No data/information is allowed to be stored unencrypted.

(6) Ensure a container firewall / runtime security is in place. A container firewall monitors container behaviour and container network traffic, analyses container vulnerabilities during runtime, applies scans against the CIS benchmark. It ensures that container clusters run in an expected security state and that security events can be identified, prevented and in case of occurrence be mitigated.

(7) Platform Operators and Developers have transparency on all/their running containers and Layer-7 communication. Transparency over all running container and their communication is a key-element of running complex microservice architectures. Whereas traditional host-based security could be protected using traditional L3/4 firewalls, microservices communicate on L7 using an overlay network. Compared to a host based architecture the traffic between containers is significantly increased (explosion of east-west traffic). Attackers typically enter through a network connection and expand the attack via the network. The network offers the first opportunity to an attack, subsequent opportunities to detect lateral movement, and the last opportunity to catch data stealing activity. The following activities need to be avoided:

- Start of compromised/unknown containers running malicious software
- Lateral Movement and Privilege Escalation (e.g. through compromised container, unauthorized connections between container/pods)
- Data Exfiltration through the network

(8) DevOps-Teams have to know whether sensitive data is exfiltrated through a container. As applications in containers may process sensitive data (and underlie compliance rules such as PCI-DSS or GDPR) it has to be ensured, that malicious attempts to exfiltrate data is both fast identified and actively prevented (Data Loss Prevention). The DevOps-Teams have to know which applications/container have access to sensitive information (such as credit card Information according to PCI-DSS or License Plates). The applications have to be known and classified by the team.

(9) Cluster security is actively monitored and security information is transported to the security teams (e.g. via the SIEM interface). Microservice Orchestration Platforms produce a high amount of log data that needs to be monitored and analyzed. Normally, the teams responsible for operating the applications and clusters should work as a first line of defense by actively monitoring the containers and its platform and responding to threats. If an identified event indicates a threat that may lead to data loss or significant downtime the information should be forwarded to the responsible Incident and Response team (e.g. via the SIEM interface) as it also may implicate that an organization is under a systematic attack. All clusters have to be monitored by using logging tools and information from a runtime-security tool/container firewall. Critical security events such as:

- Starting of unknown resources
- External Attacks (reverse-shell, DDoS)
- Malicious activities (e.g. identified mining malware, privilege escalation, attempts on direct database access, Sack-DDoS, Ping-Deaths, suspicious processes) should be forwarded to a SIEM

(10) Ensure a Web Application Firewall (WAF) is in place where connectivity to the internet exists.

(11) Dedicated networks should be used for groups of containers.

4.6. Logging and Auditing

In general important security-related events should be recorded in logs, stored centrally, protected against unauthorized change and analyzed on a regular basis. This helps to identify threats that may lead to an information security incident, maintain the integrity of important security-related information and support forensic investigations.

(1) Ensure there are documents describing security event logging, which cover:

- identification of systems or devices on which event logging should be enabled (e.g. critical business applications and systems that have experienced a major information security incident, or systems that are subject to legal or regulatory obligations) to help identify security-related events
- configuration of systems or devices to generate security-related events (including event types such as user login attempts, service creation, system crashes and creation, modification or deletion of user accounts) and event attributes associated with each event (e.g. date, time, source, destination IP addresses etc.)
- storage of security-related events within event logs (e.g. using local systems, central servers, or by using storage provided by an external service provider)
- protection of security-related event logs (e.g. via encryption, access control and backup)
- retention of security-related event logs (e.g. to meet record retention requirements and support possible forensic investigations).

(2) Define which application is in the scope of logging. This should be at least applications with a risk level "Very high" and "High" (according to risk classification of Allianz). The following aspects have to be considered while logging:

- Logging of security-related events should be enabled all times. An appropriate monitoring and a process how to deal with interruptions has to be established. This helps to ensure logging works as intended and in addition has not been tampered with.
- Log Data/Event streams of an application have to be routed and written to an archival destination that is not visible or configurable by the application, where it can be stored for further processing.
- Log Data must not include sensitive or protected information (PII, GDPR, Privacy Act)
- Log Data has to be protected from unauthorized access and accidental or deliberate modification to ensure its integrity. If using container: avoid the use of sidecars for logging (e.g. for normalization)
- Security events/alerts have to be recorded and forwarded to the existing SIEM (ACDC).

Because the amount of log data for an application can be enormous, the activities observed can be restricted to critical activities. The full scope of activities has to be defined by the business owner supported by the application manager:

(3) All critical application activity including administrative actions is logged. Logging of critical activities of applications, which includes security-related activities, will e.g. allow to analyze ongoing or completed attacks. This measure addresses the threat of repudiation, where an attacker may carry out an illegal operation and then deny that he has carried out the operation without being able to be proved otherwise.

(4) All access to sensitive data used in the SDLC is logged. Separate audit data for critical activities from functional log data.

(5) All changes to access privileges for applications and resources are logged.

(6) Successful and unsuccessful access to source code repositories is logged. Furthermore modifications to the source code have to be documented if it is not subject to the version control.

(7) Write access to registries is audited. Read accesses to sensitive images is similarly logged.

(8) All user activities especially commits, builds or deployments is logged. Although activities like commits, builds or deployments are no administrative activities, they are privileged and mean an important change in the status of source code, images and applications, leading to similar impacts when compromised. Therefore identify these privileged activities in detail and ensure they are logged to a location not modifiable by the developers.

(9) The use of special access privileges is logged and reviewed regularly (e.g. quarterly).

(10) Where security event logging is required applications are configured to:

- enable event logging (using a standard format, e.g. syslog)
- generate appropriate event types (e.g. service creation, application crash, object deletion and failed login attempt)
- incorporate relevant event attributes in event entries (e.g. IP address, time and date, protocol used, port accessed, method of connection and name of device)
- use a consistent, trusted date and time source to ensure event logs use accurate timestamps (e.g. using the Network Time Protocol (NTP), supported by GPS, atomic clocks or approved time servers on the internet).

(11) Security-related event logs are:

- reviewed regularly (e.g. to help identify suspicious or unauthorized activity)
- archived regularly (e.g. using a rotation schedule) and digitally signed before being stored
- retained according to retention standards/procedures and to support investigations (e.g. forensic analysis relating to serious cyber-attacks)
- stored securely (e.g. to support evidence handling and possible forensic analysis at a later date).

(12) During the project, the Project Manager ensures that for applications with “Medium”, “High” and “Very High” risk level the logging information is transferred to the SIEM and processed therein.

(13) After Go-Live of the business application the Application Owner is responsible for a continuous integration of the application with the SIEM and processing of logging information once established in the project and the Go-Live phase.

(14) The Project Manager ensures that logging is part of the application design, implemented, tested and configured in production in order to fulfill the former requirements. It is recommended to consider a configurable logging in order to address specific levels of logging information.

(15) For applications self-developed in the SDLC described in chapter D ensure that the design phase addresses the requirement to transfer the log data to the SIEM. This is especially vital for the format of log files and the location where they are stored.

(16) Create use cases which address the scenarios for logging and identify the information necessary to be logged. Possible use cases are:

- Insider threats (malicious insider, compromised insider)
- Abuse of privileged access / accounts
- Compromise of components, credentials etc.
- Data exfiltration detection
- Authentication activities
- Anomalous behavior and suspicious activity
- Validation of regulatory compliance and audit requirements
- Malware detection
- Network intrusion
- Operational outages
- Manipulation of critical data
- Anomalous privilege escalation
- System changes

Gartner published a set of guiding principles that should be followed in order to ensure SIEM use cases offer maximum efficiency (2). Details how to define and structure the use cases should be subject to a SIEM guideline, which is out of scope of this document.

Index of used Terms and Abbreviations

Abbreviation / Term	Description
Accountable	This role provides direction and authorizes an activity.
ACDC	Allianz Cyber Defense Center
ACL	Access Control List
AFIRM	Allianz Functional Rule for Information Risk Management (Document of the Allianz Corporate Rules Book)
AFRIS	Allianz Functional Rule for Information Security (Document of the Allianz Corporate Rules Book)
AFRIT	Allianz Functional Rule for IT (Document of the Allianz Corporate Rules Book)
API	Application Programming Interface
Application	See Business Application
Application Owner	A role which is accountable for a specific application with the responsibility to ensure that the program or programs, which make up the application, accomplish the specified objective and defines appropriate security safeguards.
ASL	Application Security Lifecycle
Availability	Ensuring that all business significant information is available when required to authorized users entities or processes.
Business Application	Every application of a Shared IT Service or any individual IT System for a specific business purpose within an OE.
Business Owner	A Business Owner represents the (intended and/or factual) users of the Business Application, taking in particular responsibility for specifying Functional, privacy-, legal- and information security-related requirements pertaining to that application. The Business Owner is accountable for Information Security within the outsourced service, as defined in the Group Sourcing Standard.
Cloud computing	Cloud computing are distributed, on-demand computing services, which are delivered across networks, typically using the Internet.
CI	Continuous Integration
CI / CD	A continuous integration and/or continuous deployment tool is used to build application packages and deploy applications or application components to a staging environment.
CD	Continuous Deployment
CIS	Center for Internet Security
CISO	Chief Information Security Officer
Code Repository	A tool to collect, merge and manage code collections/projects.
Confidentiality	Preventing information from being made available or being disclosed to unauthorized individuals or entities.
Control Objective	A control objective is a statement of the desired result or purpose to be achieved by implementing control practices in a particular activity.
CVSS	Common Vulnerability Scoring System
Data at rest	Information which is stored in electronic or physical format
Data in motion	Information which is transferred in electronic or physical format
DoS	Denial of Service
DAST	Dynamic Application Security Testing
Dynamic Application Security Testing	Testing running (web-) application by simulating known attacks such as Spider-Scans, Injections, Cross Site Scripting etc.). Show identified potential flaws but cannot point out exactly where (in the code/application) the flaw occurs.
EDR	Endpoint Detection and Response
GISF	Group Information Security Framework
IAM	Identity Access Management
IAST	Interactive Application Security Testing
Interactive Application Security Testing	IAST assesses applications from within using software instrumentation. It analyzes code for security vulnerabilities while the app is run by an automated test, human tester, or any activity “interacting” with the application functionality. This technology reports vulnerabilities in real-time and works inside the application. This type of testing does not test the entire application or codebase, but only whatever is exercised by the functional test.
IDE	The Integrated Development Environment is the local development framework on a developer desktop. It is a software suite that consolidates basic tools required to write and test software.
Information Owner	The term „owner“ identifies an individual or entity that has approved management responsibility for controlling the production, development, maintenance,

	use and security of the information assets. The term "owner" does not mean that the person actually has any property rights to the asset.
Information Security	Information Security applies to all information held and processed by an organization. Information is subject to threats of attack, error, nature (for example, flood or fire), and to vulnerabilities inherent in its use. Information is considered as an asset which has a value requiring appropriate protection, for example, against the loss of availability, confidentiality and integrity.
Integrity	Protecting information from unauthorized modification or manipulation.
ISO	Information Security Officer
IT System	Business Application or IT Service used to store, transmit and process data, consisting of hardware, software and peripheral equipment regardless of operation.
Non-Repudiation	Ensuring that information can be proven to have originated from a specific individual, entity or process.
OE	An Operating Entity is a management entity within a business segment irrespective of its legal form. An OE can consist of one or more legal entities, or, vice versa, one legal entity may comprise two OEs (e.g. in case of composites).
OE IS Function	The OE IS Function is responsible for coordinating the implementation of the Allianz Standard on Information Security and its Functional Rules.
OE Management	OE management refers to the Board of the OE, which may consist of different roles of the OE"s C-level management, e.g. CEO, CFO, COO, CIO, etc.
Outsourcer	A company that delivers services to one or more Allianz OE. The Outsourcer may also be an Allianz OE, e.g. a Shared Service OE.
Outsourcing Allianz OE	An Allianz OE that consuming services from an Outsourcer.
OWASP	Open Web Application Security Project
Personnel	Allianz employees as well as to all contract or agency employees working within OEs of the Group.
Public Cloud	In a Public Cloud the infrastructure is provisioned for open use by the general public and exists on the premises of the Cloud service provider.
Responsible	This role is responsible to perform the operative tasks
Risk Acceptance	Decision to accept a risk.
Risk analysis	Systematic use of information to identify sources and to estimate the risk.
Secrets	Secrets are secure information kept unknown and unrecognized and refer to any kind of private piece of information that acts as a key to unlock protected resources or sensitive information. They are typically used in tools, applications, containers, CI/CD, storage, databases, cloud-native environments and other solutions.Secrets could be • UserIDs • Passwords • Logins • SSH & TLS keys • API keys • Certificates • Encryption keys • Token • Code Signatures • Decryption algorythms • …
Security Awareness	Security awareness is the extent to which staff understands the importance of Information Security, the level of security required by Allianz Group and their individual security responsibilities – and acts accordingly.
Segregation of Duties	Separating certain areas of responsibility and duties in an effort to reduce fraud and unintentional mistakes.
Shared Service	A service provided by an outsourcer to two or more OEs.
SAST	Static Application Security Testing
Static Application Security Testing	Testing (non-/compiled) code for security flaws, bugs and weak coding practices (e.g. functions that allows SQL injections). Supports the Developer with IDE Plugins and integrates in pipeline.
SCA	Software Composition Analysis
SDLC	Software Development Lifecycle
SIEM	Security Information and Event Management
Software Composition Analysis	Software Composition Analysis is the process of automating the visibility into open source software (OSS) use for the purpose of risk management, security and license compliance.
Source Code Analysis	Source Code Analysis tools are designed to analyze source code and/or compiled version of code in order to help find security flaws.

Third Parties	Third Parties include internal and external providers but are not limited to service providers, contractors or consultants working within OEs of the Group on a contractual basis and who have access to Allianz information.
TLS	Transport Layer Security
Vulnerability Assessment	A process that defines, identifies, and classifies the security holes (vulnerabilities) in a computer system. It may include but is not limited to penetration tests.

Annex A - Threats

A threat pursues the goal of exploiting a vulnerability and compromising security attributes. Therefore knowing possible vulnerabilities, respective exploitability and their related threats is necessary for securing the software development process. An insecure development process will likely produce an application with exploitable vulnerabilities. Due to this security controls should be designed to address all relevant threats.

To detect threats in the DevOps processes, the threat modeling approach STRIDE is applied. Threat modeling is a method to detect potential vulnerabilities, further risks or threats in an application as early as possible.

The following tables summarize the main types of threats encountered for the generalized software development framework described in chapter D. They aggregate resulting categories of threats according to the STRIDE model and explain possible business issues as well as possible countermeasures. The threats will be addressed by the controls explained in chapter D.

As the actual implementation of the DevOps toolchain and its processes might differ inside the Allianz group, the corresponding scope of threats and their risk assessment might lead to variant threat landscapes and different risk severities. Risk severity is the product of the likelihood and an impact level. Therefore the controls defined in this document are not affected by these variations and address the threats mentioned below as they are.

Technical Threats

This list is derived from common analyses of threats to development environments.

Data Breaches

STRIDE: I (Information Disclosure)

Threats	Possible Business Impact	Possible Controls
Sensitive or confidential data is viewed, released or stolen. Beside a disclosure of sensitive information it may be used to prepare additional and possibly more complex attacks.	Damage of brand reputation and resulting loss of business, criminal charges, fines	Multi-factor authentication, data encryption at rest and in transit, restriction of access paths

Insufficient identity and access management

STRIDE: S (Spoofing), T (Tampering), R (Repudiation), I (Information Disclosure), D (Denial of Service), E (Elevation of Privileges)

Threats	Possible Business Impact	Possible Controls
Attackers can masquerade as legitimate users, operators or developers. Access to the development system with attack vectors addressing critical components is possible. Improper data protection of e.g. files against unauthorized access can allow an attacker to read information not intended for disclosure or manipulate the files in his	E.g. full control of all resources and data for high privileged accounts, information disclosure, blackmail, potential full control over IT by attacker, business disruption	Monitoring, identity lifecycle, multi-factor authentication, appropriate authorization management, segregation of duties, Need-to-know principle for privileges, continuous validation of privileges, secret vault, ACLs, auditing & logging, data encryption, employee training & awareness, Storage of the codes should be in enterprise

Threats	Possible Business Impact	Possible Controls
sense for an attack (e.g. pipeline scripts or configuration files of a CI/CD component)		approved faclities (such as Github enterprise

Potential repudiation

STRIDE: R (Repudiation)

Threats	Possible Business Impact	Possible Controls
Compromised processes in the toolchain claim they did not receive data or did not conduct the activities they were supposed to do, e.g. create a container from an image.	Loss of control, reputational damage,	Auditing and logging should be enabled

Insecure Interfaces and APIs

STRIDE: S (Spoofing), T (Tampering), R (Repudiation), I (Information Disclosure), E (Elevation of Privileges)

Threats	Possible Business Impact	Possible Controls
Attackers can exploit vulnerabilities of the interfaces and APIs for their attacks, especially when they are exposed to the internet. They are able to control services relevant for the development process.	Manipulation of applications in development or disabling platform services.	Use of strong authentication (e.g. MFA), limit access (e.g. whitelisted IP), controls in front of APIs (e.g. API gateway), authentication with API gateway (e.g. OAuth), activity monitoring, protection of API keys (e.g. no hardcoding in the code), encryption in storage and in transit, secure coding training/certifcation for developers (including vendors

Vulnerabilities in the development toolchain

STRIDE: S (Spoofing), T (Tampering), R (Repudiation), I (Information Disclosure), D (Denial of Service), E (Elevation of Privileges)

Threats	Possible Business Impact	Possible Controls
Attackers can exploit vulnerabilities in the software stack of the development toolchain. Numerous threats like Cross-Build-Injection, compromise of secrets, import of objects with malware or code with vulnerabilities etc. An attacker may also exploit the trust relationships of a plugin or application of the toolchain (e.g. the automation engine) with an existing vulnerability	Compromise of systems and disclose / manipulation of data, loss of data, reputational damage, manipulation of applications in development (backdoors, malware etc.)	Lifecycle management and patching, vulnerability scanning, source code review, runtime vulnerability scanning and protection, hardening (e.g. only allow Allianz harden base image), network segmentation, secure software development and testing, no hardcoding of secrets - use PAM integration in the CICD and applications, penetration testing before go-

Threats	Possible Business Impact	Possible Controls
in order to use the standard access to start an attack on the intended target system.		live, secure coding training/certification for developers (including vendors)

Malicious insiders

STRIDE: S (Spoofing), T (Tampering), I (Information Disclosure)

Threats	Possible Business Impact	Possible Controls
Malicious or negligent actions that compromise security, sabotage	Compromise of systems and disclose of data, loss of data reputational damage	Activity monitoring, segregation of duties, automated HR/IAM processes, regular permission reviews, contractor management, easy-to-follow security controls

Advanced Persistent Threats

STRIDE: I (Information Disclosure), E (Elevation of Privileges)

Threats	Possible Business Impact	Possible Controls
Long-time presence of attacker, multiple ways of access, multiple compromised systems (result of other threats)	Continuous information disclosure, blackmail, potential full control over IT by attacker, business disruption	Awareness training, malware controls, periodic scans for Indicators of Compromise (IoC), regular re-deployment of systems, security monitoring and analytics

Denial of Service

STRIDE: D (Denial of Service)

Threats	Possible Business Impact	Possible Controls
Poorly secured deployment, unavailable pipelines, DoS attack	Reduced capacity for legitimate use, reduced availability and responsiveness, increased costs, delivery of software interrupted	Monitoring of performance and usage metrics, DoS protection, strong authentication, incident response framework, whitelisting, hardening

Missing approval of changes

STRIDE: T (Tampering), R (Repudiation), I (Information Disclosure), D (Denial of Service)

Threats	Possible Business Impact	Possible Controls
If pipeline scripts or the infrastructure configuration are changed and there is no review based on the Four-eyes-principle, manipulations can take place very easily. In the absence of a review, it may happen that no team member learns what changes have been made. This can lead to data being	Business disruption, information disclosure	Ensuring processes of the change management, implementation of Four-eyes-principle, auditing, all critical security issues and vulnerabilities fixed

Threats	Possible Business Impact	Possible Controls
read or the pipeline is compromised (denial of service attack). An unavailable pipeline leads to a development and delivery stop.		

Retrieval of insecure dependencies

STRIDE: S (Spoofing), T (Tampering), I (Information Disclosure), E (Elevation of Privileges)

Threats	Possible Business Impact	Possible Controls
Integration of software artifacts (open source libraries, plugins for the CI/CD pipeline tools or development framework, base images etc.) from unknown and potential insecure sources may introduce vulnerabilities and malicious code / artifacts into the development ecosystem	Continuous information disclosure, blackmail, potential full control over IT by attacker, business disruption	Awareness training, malware controls, software composition analysis, regular re-deployment of systems, security monitoring and analytics, unknown and potential insecure sources should not be allowed unless it is tested to be safe by the security team

Annex B - Responsibilities

During the development lifecycle different roles and responsibilities are involved in various activities. In general, the Business Owners are accountable for most of the activities. They are supported by different roles and functions of the OE, e.g. OE Management, Project Manager, Application Owner etc..

The following table (RACI matrix) provides a suggestion of how the roles and responsibilities for the security controls mentioned in the chapters before may be distributed. If an activity developing an application is not a project, the role of the project manager has to be exchanged by the role responsible for the activity.

Section	OE Management	Business Owner	Application Owner	Project Manager	OE IS Function
3 Application Development Management					
3.2 Application Development Methodology					
Application Security Lifecycle	A	R			
3.3 Securing Application Development Environments					
General Recommendations		A		R	
Developer Framework		A		R	
Source Code Repository		A		R	
CI / CD Pipeline		A		R	
Image Registries		A		R	
Container Orchestration Software		A	R[7]	R[7]	
4 Application Development Lifecycle					
4.1 Application Design					
Risk Assessment		A	(R)[8]	R	(C)[9]
Security Requirements		A/C		R	(C)[10]

[7] Depending on the distribution of duties between project and service provider
[8] The Application Owner is responsible to re-perform the risk assessment during application operation, in case of significant changes to the application
[9] In case of “High” or “Very High” risk levels, the OE IS Function must review and sign-off the Risk Assessment.
[10] In case of “High” and “Very High” risk levels, the OE IS Function must sign-off the derived security requirements.

Security Architecture		A		R	C
Secure Coding Principles		A		R	
4.2 Application Build					
Secure Build		A		R	
Secure Coding Guidelines and Coding Patterns		A		R	
Defect Management		A		R	
4.3 Application Security Testing					
General Requirements and Recommendations		A	R[11]	R	C
Static Application Security Testing		A		R	
Software Composition Analysis		A		R	
Dynamic and Interactive Application Security Testing		A		R	
Penetration Testing		A		R	
4.4. Application Deployment					
Container Security		A		R	
Business Application Acceptance		A	C	R	C
Secure Deployment		A		R	
Secret Management		A		R	
4.5 Application Operation					
On-going Application Management Process		A	R[12]		(C)[13]
Container Runtime Environment		A	R		
4.6 Logging and Auditing		A	C	R	

[11] After go-live during on-going application operation
[12] The Application Owner is responsible to re-perform the risk assessment and threat modelling during application operation, in case of significant changes to the application or at least annually.
[13] In case of significant changes to the application or at least annually a new risk assessment is necessary and must be reviewed and signed-off by the OE IS Function in case of “High” or “Very High” risk levels.

Bibliography

1. **OWASP Foundation.** OWASP Application Security Verification Standard Version 4.0. [Online] 2019. https://owasp.org/www-project-application-security-verification-standard/.

2. **Gartner, G. Sadowski.** Gartner Research. [Online] July 2019. https://www.gartner.com/en/documents/3950486/how-to-build-security-use-cases-for-your-siem.

Document Information

Document:	IS Practice #06 Secure Software Development
Author(s):	Dr. Thorsten Deckers (NTT DATA), Robert Klawes
Contact Person(s):	Robert Klawes
Area of Application:	Allianz Group Information Security Framework (GISF)

Amendments and Updates:

Version	Date	Reason for and Extent of Changes	Author(s)
0.1	March 2020	Initial version	Dr. Thorsten Deckers (NTT Security)
0.2	March 2020	Added new content	Dr. Thorsten Deckers (NTT Security)
0.3	March 2020	Added new content	Dr. Thorsten Deckers (NTT Security)
0.4	March 2020	Added new content	Dr. Thorsten Deckers (NTT Security)
0.5	May 2020	Included feedback from D. Stephanovic, J. Lange, M. Srinivasan, A. Wagner (Metafinanz), M. Siegert (NTT), modifications to figures and several other modifications to the content	Dr. Thorsten Deckers (NTT Security)
0.6	June 2020	Included feedback from R. Oelssner and M. Gibson Adapted to new IS Practice template	Dr. Thorsten Deckers (NTT Security)
0.7	June 2020	Included feedback from M. Srinivasan, added missing annex concerning responsibilities	Dr. Thorsten Deckers (NTT Security)
0.8	October 2020	Included feedback from ISO community	Dr. Thorsten Deckers (NTT DATA)
0.9	October 2020	Included new feedback after alignment with ISOs	Dr. Thorsten Deckers (NTT DATA)
0.9.x	October 2020	Included additional feedback for final version before official and binding release	Robert Klawes (Allianz SE), Dr. Thorsten Deckers (NTT DATA)
1.0	December 2020	Version 1.0 published after review and approval of Carsten Scholz	Robert Klawes (Allianz SE)
1.1	December 2021	Included additional feedback for next release	Dr. Thorsten Deckers (NTT DATA)
1.2	August 2022	Added new content and updates on existing (e.g. Secret Mgmt, SBOM and others)	Robert Klawes, Thorsten Deckers