Corporate IT – Principles of Security for Development and Operation

Guideline

Version: 1.1

Owner: Corporate IT

Index

1 Objective and Purpose

1.1 Objective and Scope

This document defines the overall principles for the secure development and operation.

This guideline applies for the entire Aurubis corporate group and concerns all types of IT systems used by Aurubis – whether in the cloud or on-premises – irrespective of whether they are maintained or operated by the Corporate IT organization or the local/regional IT and OT departments or external service providers or suppliers.

This group policy applies to all fully consolidated companies of Aurubis and all subsidiaries in which Aurubis AG directly or indirectly holds the majority, unless exceptions are explicitly made for specific subsidiaries.

In the case of subsidiaries in which Aurubis does not directly or indirectly hold the majority of shares, this group policy applies only if agreed with the respective co shareholders, or if the respective Management Board has determined such.

As technology changes, regularly review and adaptation of principles via EAM-Board is mandatory.

Compliance with the policy by service providers and suppliers is the responsibility of the commissioning department or susidiary.

1.2 Roles and Responsibilities

These principles apply for the entire Aurubis Group. They affect all types of IT-systems and all employees involved with selection, implementation, enhancement, developing and maintaining IT-systems – on-premises and cloud – within the entire Aurubis Group.

These principles are to be considered across the entirely of the product lifecycle, demand assessment, project initiation, development and commissioning as well as usage phases. Consequently, they must be taken into account in Information Security Assessment as well as the Project Management Process. (see ISMS)

When procuring software/licenses, it is of utmost importance to secure contractual assurance that these guidelines will be adhered to by the supplier.

Corporate-IT has the right to review the development processes and measures.

Agreed licensing models, in particular ownership of code and intellectual property rights regarding software development, apply, as in the rest of the company.

This regulation is to be applied to both internal and external IT providers, this includes freelancers and external developers too.

The guidelines focus on factual aspects and are not tied to any specific development platform (SAP, Java, Web applications, etc.). They serve as directive and can or should be refined by software architects to comply with the specific requirements of the respective environments in development guidelines.

2 Introduction

2.1 Motivation - Why Secure Development and Operation is Important

In today's tech landscape, secure development and operation is more important than ever. Secure development and operation is essential to protect sensitive data, prevent cyber-attacks, and maintain the integrity of software systems. Failure to prioritize security can lead to devastating consequences for both individuals and companies.

Risks of Not Prioritizing Secure Development and Operation (among others):

- Data breaches and leaks
- Loss of customer trust and business reputation
- Legal and financial repercussions

2.2 Pillars of Aurubis Secure Development and Operation Principles

Aurubis Secure development and operation principles rely on four pillars and involve techniques that prevent vulnerabilities, protect sensitive information and prevent cyber-attacks. However, it's important to remember that these guidelines are not a one-time fix. Continuous education and adopting language / platform specific best practices are necessary to maintain their effectiveness.

- **Secure Coding Practices:** Secure coding practices are essential for preventing vulnerabilities in software. All employees, who develop software should be trained on secure coding practices and should follow them consistently throughout the development process. Examples of secure coding practices include input validation, output encoding, and error handling.

- **Secure Deployment Practices:** Secure deployment is essential for ensuring that software is not vulnerable to attacks once it is deployed. Key strategies for secure deployment include using secure protocols for data transmission, implementing access controls, and regularly updating software to address vulnerabilities.

- **Assessment & Threat Modeling:** Threat modeling is a process for identifying potential threats to the security of software and developing strategies for mitigating those threats. IT Staffs should be trained on threat modeling and should incorporate it into their development process.

- **Ongoing Monitoring & Maintenance:** Ongoing monitoring and maintenance are essential for ensuring that software remains secure over time. This includes conducting regular vulnerability assessments, monitoring for suspicious activity, and promptly addressing any identified issues.

3 Principles

The principles for secure development and operation are defined to protect the IT landscape of Aurubis from the early stages of implementation to the build and operational phases.

3.1 Zero Trust Security:

Adopt a "zero trust" security model, where trust is never assumed and access is continuously verified, even among microservices within your network.

3.2 Threat Modelling:

Identify potential threats and attack vectors specific to your application, and design security controls to mitigate these risks.

3.3 Defense in Depth:

Implement multiple layers of security controls, such as firewalls, authentication & authorization, to provide redundancy and mitigate the impact of potential breaches.

3.4 Secure Authentication:

Use strong authentication mechanisms, such as multi-factor authentication. Implement connection to Azure AD for user authentication. For system users use encrypted credentials using industry-standard techniques.

3.5 Secure Communication:

Use secure communication protocols (like HTTPS) to protect data in transit (Digital Signature). Encrypt sensitive data to prevent eavesdropping.

3.6 Secure Data Storage:

Employ encryption and access controls to protect sensitive data at rest. Avoid storing sensitive data if it's not necessary for your application's functionality.

3.7 Secure API:

Secure your APIs with proper authentication and authorization mechanisms. Implement API gateways that can handle authentication, rate limiting, and traffic routing.

3.8 Secret Management:

Store sensitive information such as database credentials and API keys securely using a secret management solution. Avoid hardcoding secrets in config files or code.

3.9 Security Testing:

Conduct regular security testing, including vulnerability scanning, penetration testing, and code reviews, to identify and address security issues proactively. It must be ensured that no intentionally or accidentally inserted malicious code / library is contained at the time of delivery as well as to rule out the presence of known vulnerabilities.

3.10 Secure Configuration:

Ensure that your application and server configurations are secure by default. Remove unnecessary services and features that could be exploited.

3.11 Dependency Scanning:

Regularly scan and update dependencies and libraries used to address known vulnerabilities. Ensure that used libraries are proved against CVEs (catalogue publicly disclosed cybersecurity vulnerabilities). Incorporate security testing into your CI/CD pipeline to catch security issues early in the development process (see appendix).

3.12 Container Security:

If you're using containers (e.g., Docker), ensure that container images are scanned for vulnerabilities, and use tools to enforce security policies.

3.13 Use Secure Libraries:

Leverage established well-proved libraries and frameworks rather than trying to implement features from scratch. This can help ensure that security mechanisms are properly implemented and up-to-date (see appendix).

3.14 Logging and Monitoring:

Implement comprehensive logging and monitoring to detect and respond to security incidents in a timely manner. Keep the important system actions logged and stored in central logging service.

3.15 Principle of Least Privilege:

Limit the permissions and privileges of your code to the minimum necessary to perform its tasks. Avoid running code with excessive privileges, as this can lead to security vulnerabilities.

3.16 Input Validation:

Always validate and sanitize user input to prevent malicious data from entering in application. It helps protect against common attacks like injection, cross-site scripting (XSS).

3.17 Error Handling:

Handle errors gracefully by providing informative error messages to developers and logging non-sensitive errors. Avoid exposing sensitive information in error messages that could be useful to attackers.

3.18 Regular Updates and Patching:

Keep all software components, including libraries and frameworks, up to date with security patches to address known vulnerabilities.

3.19 Code Reviews and Testing:

Perform regular code reviews (at best peer review) and testing to identify and fix security vulnerabilities and ensure that the system is secure and reliable. Use tools for identifying security weaknesses and vulnerabilities (check appendix).

3.20 Separate Development, Quality and Production environments:

A solid development process requires dedicated environment for each stage (development, test and prod).

3.21 Codes only in Corporate Repository:

Store Aurubis Code only in Aurubis defined repository.

3.22 Incident Response Plan:

Develop a robust incident response plan to effectively manage and mitigate security incidents when they occur. Ensure to define adequate response time.

3.23 Documentation:

Maintain up-to-date documentation, including security-related information, to facilitate secure development and operation. And spared knowledge inter-team and intra-team. The effective documentation (incl. services provided) is to be done in Aurubis defined platforms.

3.24 Defining Acceptance Criteria for Quality, Correctness, and Security in Project Standards:

Define acceptance tests with regard to the quality and correctness of the services provided in accordance with the standard implemented in the respective project: e.g., here the definition of Done. And ensure that security aspects and data protection are considered correctly.

3.25 Security Training:

Provide security training for development and operation teams to raise awareness of security best practices and emerging threats.

3.26 Extension for Microservices:

As microservice programming is increasingly used as a paradigm, the following three additional guidelines must be adhered to for such developments.

3.26.1 *Immutable Infrastructure:*

Consider using immutable infrastructure patterns where microservices are treated as disposable entities. This can make it easier to patch and update services rapidly.

3.26.2 *Redundancy and Failover:*

Design your microservices for redundancy and failover to ensure high availability and resilience to attacks or failures.

3.26.3 *Auto Recovery:*

Design your services to conduct auto-recovery in the case of crash or failures.

4 Appendix

4.1 Commonly used Terminology for Application Security Testing

- **Static application security testing (SAST)** scans binary code or application source code when the application is not running to find vulnerabilities based on design or implementation.
- **Dynamic application security testing (DAST)** tests running web applications for security issues by mimicking the same techniques that malicious attackers use to find application vulnerabilities.
- **Interactive application security testing (IAST)** includes dynamic and interactive testing. It uses actual inputs and actions in a controlled manner to probe the application under test.
- **Mobile application security testing (MAST)** addresses mobile-specific issues like data leaks from mobile devices and jailbreaking, in addition to typical security vulnerabilities.
- **Software composition analysis (SCA)** inventories open source and third-party commercial components used within an application, identifies security vulnerabilities within those components, and provides ways to remediate the issues.
- **Runtime application self-protection (RASP)** analyzes user behavior and application traffic at runtime to identify and prevent cyber threats.