4 Coding

The checklist is dynamic, not static, and will be updated regularly. If you have any suggestions or comments, we would like to hear from you.

Possible measures for secure coding

- Create a list of approved tools and libraries
- Scanning of dependencies for known vulnerabilities or outdated versions
- Manual code review
- Static code analysis with security rules

Use approved tools and frameworks

- Establish a list of
    - approved tools with associated security features that will help to automate and enforce security procedures in the coding
    - approved supporting components
    - permitted third-party components and development tools. Avoid sharing of personal data through third-party libraries – use synthetic data instead
- Describe in the lists what the various tools and supporting components will be used for, including new security analysis, functionality, and protection. Tools and supporting components should undergo risk assessment, and be analysed for privacy and security vulnerabilities.
- Keep the lists updated according to organisation guidelines. This means that new tools and versions must be reviewed, and used wherever possible.
- Use only approved tools and supporting components from the list. Any exceptions should be documented and approved by the security officer.

Examples: Code patterns and templates for widely used/established functionality should be generalised, quality checked, and documented as current patterns/templates. Examples include how database calls should be made, and how they should be structured.

Examples of approved tools and supporting components that must be included in lists:

- Code library
- Programming language
- Version control system
- Testing tools
- Infrastructure
- Monitoring tools
- Logging server
- Third party framework and APIs

Example of this in practice:
The organisation has decided on a coding method for initial and further development. This thus applies to all projects and project managers. They use a task management system, such as Jira or Confluence. All tasks have a dedicated owner who is also responsible for the task's data protection and security. The status of all active tasks is established at a morning

meeting for the team, where challenges that have arisen relating to, e.g. data protection can be discussed. Code review must be carried out by a different developer. This is important for avoiding personnel dependencies in the code, as well as to improve the quality of the code. But, it is equally important to have an additional person review the code to verify that data protection and security is preserved. Versions will be built for testing, and a release owner will be appointed.

Invalidate unsafe functions and modules

- Unsafe functions and modules are handled by tools, such as OWASP Dependency Check.
- Disable unnecessary tracking, logging, and collection of personal data.

Static code analysis and review

- Establish business routines and/or checklists for static analysis and code review.
- Analyse and review the source code regularly, and each time it is built.
- Check data flow, storage and temporary storage of data.
- Static analysis for data protection should primarily be performed manually, as "automated" tools for reviewing code for data protection are limited. It can be difficult to identify patterns because data alone does not necessarily constitute personal data, but connections between different types of data can provide personal data.
- It is important that the person who does the work (the reviewer) has a thorough understanding of both the principles of data protection, the data subject rights and the requirements for data protection by design.
- Have different levels of scanning, such as for developers, for security advisors, and for the person responsible for product release.
- Establish guidelines for what should be scanned, and when.

Examples of tools for static code analysis:

- RIPS PHP Static Code Analysis Tool, OWASP LAPSE+ Static Code Analysis Tool, SonarQube, Checkmarx.

Examples of how to perform static code review:

- Regular (e.g. daily or weekly) scanning.
- Scanning with both Scan Master and Dev Brancher, and performing a full security scan.
- During "on build scanning", minimum security requirements should be checked by Dev Brancher or Scan Master (e.g. SQL-Injection).
- "On Demand scanning" is performed by the user's IDE, and is a full security scan.
- Use checklists for code review:
    - Example A1. Find all points with database access. Are queries with "dirty variables" being concatenated?
    - Example A4. How are users blocked from accessing other users' data?

Qualities to look for in a code analysis tool:

- It should be designed for security.
- It should support multiple levels (different programming languages and platforms).

- It must be possible to expand. The tool should be able to be expanded to include new attack and defence techniques.
- It must be useful to both security analysts and developers.
- It should support existing development processes.
- It should have value for multiple parties with ownership of the development (such as interfaces with measurements that can support decisions on release management, check costs when changes occur, and provide information needed for maintenance).

Why secure coding?

- The General Data Protection Regulation will apply when software is to be developed, cf. Article 25.
- Tools, support systems, and infrastructure should be “state of the art”; i.e., the newest and most updated version of the technology, cf. Article 25.
- It should be a fixed goal for the management to use secure and common methodologies for
    - coding
    - identifying and removing vulnerabilities in the code
    - using automated tools for code analysis
    - static code analysis and review