ME/PSO

HANDBOOK

Cybersecurity Handbook for IT Developers

Secure programming and protection in software development

Companion material for the training course:
"Cybersecurity Awareness for IT Developers"

Document No. MEPSO-PRI-DEV-006 · Classification: Internal Use
Information and Cyber Security Department · MEPSO AD Skopje

Contents

1. Introduction

The software we create is the foundation of the digital infrastructure. Every line of code can be either a protection or a vulnerability. Security must be part of development from the very beginning, not an add-on at the end.

This handbook is intended for programmers and developers. We will cover principles of secure programming, the most common vulnerabilities, and practices for building reliable software. The goal is for security to become a natural part of your development process.

As an organization that manages critical infrastructure, the quality and security of the software we create and use have direct consequences. Secure programming is not an option — it is a professional responsibility.

> **Key principle.** Security cannot be "added on" at the end. It must be built into the design and the implementation from the very beginning. Fixing a security flaw in production is many times more expensive than preventing it during development.

2. Security as part of development

Secure programming means thinking about security at every stage of development.

2.1 Security by design

- ✓ Think about the threats already at the design stage
- ✓ Apply the principle of least privilege
- ✓ Assume that input data is untrusted
- ✓ Plan for failure — how the system reacts to errors
- ✓ Defense in depth — multiple layers of protection

2.2 A way of thinking

A good programmer thinks not only "how will this work", but also "how could this be abused". This security mindset is key to building reliable software.

> **Important.** Every user input, every external connection, every dependency is a potential point of attack. Treat them with appropriate caution.

3. The most common security vulnerabilities

Understanding the common vulnerabilities is the first step toward preventing them.

Vulnerability	Risk
Injection	Executing malicious commands via input data
Weak authentication	Unauthorized access to the system
Data exposure	Disclosure of sensitive information
Misconfiguration	Security flaws caused by bad settings
Vulnerable dependencies	Flaws inherited from external components
Insufficient validation	Accepting malicious input data

4. Input validation

Never trust input data. Every input — from a user, a file, the network or another system — must be validated.

4.1 Validation principles

- ✓ Validate on the server side, not only on the client side
- ✓ Apply an "allow-list" approach — permit only what is known to be good
- ✓ Check type, length, format and range
- ✓ Reject — do not try to "fix" — suspicious data
- ✓ Validate as close to the point of entry as possible

> **The golden rule.** Every piece of input data is potentially malicious until proven otherwise. Strict validation is the first line of defense against many kinds of attacks.

5. Secure management of passwords and secrets

Passwords, keys and other secrets require special protection in the code.

5.1 Rules

- ✓ Never embed secrets directly in the code
- ✓ Do not store passwords in plain text
- ✓ Use secure methods for storing passwords
- ✓ Keep secrets in protected configuration systems
- ✓ Do not include secrets in the version control system

> **A common mistake.** Embedding passwords, API keys or other secrets directly in the source code. If the code is shared or leaks, the secrets are exposed. Always use secure methods for managing secrets.

6. Authentication and authorization

Proper identity and access management is the foundation of secure software.

6.1 Authentication

- ✓ Use proven, standard mechanisms for authentication
- ✓ Support two-factor authentication where possible
- ✓ Protect against password-guessing attacks
- ✓ Manage sessions securely

6.2 Authorization

- ✓ Check the permissions for every action
- ✓ Apply least privilege
- ✓ Do not rely solely on hiding functions in the interface
- ✓ Validate the authorization on the server side

7. Protection against injections

Injections are among the most dangerous vulnerabilities — the attacker inserts malicious code that the system then executes.

7.1 Types of injections

- **SQL injection** — manipulating databases;
- **Command injection** — executing system commands;
- **Cross-site scripting (XSS)** — injecting scripts into web pages.

7.2 Protection

- ✓ Use parameterized queries for databases
- ✓ Validate and escape output data
- ✓ Do not compose commands with untrusted input data
- ✓ Apply appropriate output encoding

> **Key point.** Never compose queries or commands by directly concatenating input data. Always use safe, parameterized methods.

8. Secure communication

- ✓ Use encrypted communication for sensitive data
- ✓ Validate certificates properly
- ✓ Do not transmit sensitive data in plain text

- ✓ Apply secure protocols; avoid deprecated ones

9. Dependency management

Modern software uses many external libraries. Every dependency carries potential vulnerabilities.

- ✓ Use only trusted, maintained libraries
- ✓ Update the dependencies regularly
- ✓ Monitor for published vulnerabilities in the libraries you use
- ✓ Minimize the number of dependencies
- ✓ Vet dependencies before including them

Dependency risk. A vulnerability in an external library becomes a vulnerability in your software. Keeping dependencies updated and following the security advisories is key.

10. Error handling and logging

10.1 Secure error handling

- ✓ Do not reveal sensitive details in error messages
- ✓ Log the errors for analysis, but securely
- ✓ Do not expose internal structure to users
- ✓ Handle errors in a controlled way

10.2 Secure logging

- ✓ Do not log sensitive data (passwords, personal data)
- ✓ Log security-relevant events
- ✓ Protect the logs from unauthorized access

Warning. Detailed error messages, useful for debugging, can reveal sensitive information to attackers. In production, display generic messages and keep the details in protected logs.

11. Secure code storage

- ✓ Do not include secrets in the version control system
- ✓ Control access to the source code
- ✓ Review the code for security flaws
- ✓ Use secure collaboration practices
- ✓ Include security checks in the development process

12. Security testing

Security testing uncovers vulnerabilities before the attackers do.

- ✓ Test the input data with unexpected values
- ✓ Review the code for security problems
- ✓ Use automated analysis tools
- ✓ Test the authentication and the authorization

13. Practical scenarios

Scenario 1: User input into a database

You are developing a feature that accepts a user's search input and uses it in a database query.

Correct approach: You use parameterized queries and never concatenate user input directly into SQL. You validate the input before use. This prevents SQL injection.

Scenario 2: Storing an API key

The application needs to connect to an external service using an API key.

Correct approach: You do not embed the key in the code. You keep it in a protected configuration system or an environment variable, outside version control. That way the key stays protected even if the code is shared.

Scenario 3: Displaying an error

The application hits a database error while processing a request.

Correct approach: You show the user a generic message. The error details (which could reveal internal structure) are logged securely for analysis. You do not expose technical details an attacker could exploit.

14. Frequently asked questions

Why is client-side validation not enough?

Because attackers can bypass it. Client-side validation is for user experience; security validation must happen on the server side, where the attacker cannot manipulate it.

Is it safe to store passwords if they are encrypted?

Passwords should be stored using secure, one-way methods specifically designed for passwords, not with ordinary encryption. Never in plain text.

How often should I update dependencies?

Regularly, and immediately when a security vulnerability is published in a library you use. Follow the security advisories for your dependencies.

What is the most important security principle in programming?

Never trust input data, and build security in from the start. These two principles prevent a large share of the common vulnerabilities.

Does security testing slow down development?

In the short term it takes effort, but in the long term it saves a lot. Fixing a flaw in production is many times more expensive than preventing it during development.

Glossary of terms

Term	Meaning
Injection	Inserting malicious code that the system executes
SQL injection	An injection targeting databases
XSS	Cross-site scripting — injecting scripts into web pages
Validation	Checking that input data is correct and safe
Authentication	Confirming identity
Authorization	Determining the permitted actions
Parameterized query	A safe way to query a database without injection
Dependency	An external library used by the software
Least privilege	Granting only the rights that are necessary

A security checklist for developers

At design time

- ✓ I thought about the potential threats
- ✓ I applied the principle of least privilege
- ✓ I planned for secure error handling

During implementation

- ✓ I validate all input data on the server side
- ✓ I use parameterized queries for databases
- ✓ I do not embed secrets in the code
- ✓ I apply secure password management

- ✓ I do not expose sensitive details in errors

Before production

- ✓ I reviewed the code for security flaws
- ✓ I updated the dependencies
- ✓ I tested with unexpected input data
- ✓ I checked the authentication and the authorization
- ✓ I did not include secrets in version control

> **Habit above all.** When the security checks become a natural part of your workflow, they do not slow you down — they become automatic. The best programmers think about security without having to remind themselves.

15. Conclusion

Secure programming is a professional responsibility. Every line of code you write can strengthen or weaken the software. By building security principles into your development process, you create reliable software that protects the organization and its users.

> **Key message.** Security is not someone else's job — it is the responsibility of every programmer. Think about security from the first line of code all the way to production, and your software will be part of the solution, not part of the problem.

MEPSO AD Skopje — Electricity Transmission System Operator of North Macedonia
Document No. MEPSO-PRI-DEV-006 · Classification: Internal Use