Secure design principles

CYDDes3d | 4 days | On-site or online | Hands-on

An intensive four-day course designed for software engineers, architects, and security professionals seeking to embed security into every stage of software development. In this course you can explore the foundational and advanced principles of secure software design, from the classic Saltzer and Schroeder principles to real-world vulnerabilities and defensive coding strategies. Participants will dive deep into core concepts such as the CIA triad, risk analysis, secure coding practices, exception handling, input validation, and the critical role of configuration and hardening.

The course blends theory with practice through numerous labs and case studies, covering modern security challenges such as SQL injection, insecure deserialization or container hardening. Topics such as cryptography, access control, authentication, race conditions, and denial of service are addressed in detail, ensuring a comprehensive understanding of how to identify and mitigate risks in contemporary software systems. Whether you're building web or desktop applications, monolithic apps or microservices, this course equips you with the skills to architect, develop, and maintain secure and resilient systems.

So that you are prepared for the forces of the dark side.

So that nothing unexpected happens.

Nothing.

Skills and drills

28 LABS

15 CASE STUDIES

Audience

Architects and developers

Group size

12 participants

Preparedness

General software design and development

Outline

- Cyber security basics
- Secure design principles of Saltzer and Schroeder
- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Work factor
- Compromise recording
- Secure coding beyond the high-level principles
- Wrap up

Standards and references

CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Input validation approaches and principles
- Understanding security design principles
- Detailed analysis of the various mistakes and the best practices
- Going beyond the low hanging fruits
- Understanding how cryptography supports security

Table of contents

- Encoding challenges
- Unicode challenges
- Validation with regex

Configuration and hardening

- Configuration principles
- Configuration management
- Hardening
- Server misconfiguration

› Complete mediation

Input validation

- Integer handling problems
  - Representing signed numbers
  - Integer visualization
  - Integer overflow
  - *Lab – Integer overflow*
  - Signed / unsigned confusion
  - *Case study – The Stockholm Stock Exchange*
  - Integer truncation
  - Best practices
    - Upcasting
    - Precondition testing
    - Postcondition testing
    - Using big integer libraries
    - Integer handling
    - *Lab – Integer handling*

Day 2

› Complete mediation (continued)

Input validation

- Injection
  - SQL injection
    - SQL injection basics
    - *Lab – SQL injection*
    - Attack techniques
    - Content-based blind SQL injection
    - Time-based blind SQL injection

- SQL injection best practices
- Input validation
- Parameterized queries
- *Lab – Using prepared statements*
- Database defense in depth
- *Case study – SQL injection in Fortra FileCatalyst*

- Code injection
  - OS command injection
  - OS command injection best practices
  - Using Runtime.exec()
  - *Case study – Shellshock*
  - *Lab - Shellshock*
  - *Case study – Command injection in VMware Aria*

› Open design

Cryptography for developers

- Cryptography basics
- Cryptographic libraries in brief
- Elementary algorithms
  - Hashing
    - Hashing basics
    - Common hashing mistakes
    - Hashing APIs
    - *Lab – Hashing*
  - Random number generation
    - Pseudo random number generators (PRNGs)
    - Cryptographically secure PRNGs
    - Using virtual random streams
    - Weak and strong PRNGs
    - *Lab – Using random numbers*
    - *Case study – Equifax credit account freeze*
- Confidentiality protection
  - Symmetric encryption
    - Block ciphers
    - Modes of operation
    - Modes of operation and IV – best practices
    - Symmetric encryption APIs
    - Symmetric encryption APIs with streams
    - *Lab – Symmetric encryption*
  - Asymmetric encryption
  - Combining symmetric and asymmetric algorithms
  - Key exchange and agreement

    - Key exchange
    - Diffie-Hellman key agreement algorithm
    - Key exchange pitfalls and best practices
- Integrity protection
  - Message Authentication Code (MAC)
    - Calculating MAC
    - *Lab – Calculating MAC*
  - Digital signature
    - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
    - Digital signature APIs
    - *Lab – Digital signature with ECDSA*

Day 3

› Separation of privilege

Access control and authorization

- Access control basics
- Missing or improper authorization

Case study – Broken authn/authz in Apache OFBiz

› Least privilege

Authorization weaknesses

- Failure to restrict URL access

Confused deputy

- Insecure direct object reference (IDOR)

Lab – Insecure Direct Object Reference

- Authorization bypass through user-controlled keys

Case study – Remote takeover of Nexx garage doors and alarms

Lab – Horizontal authorization

File upload

- Unrestricted file upload
- Good practices

Lab – Unrestricted file upload

Case study – File upload vulnerability in Netflix Genie

› Least common mechanism

Containers

- Container security concerns
- Containerization, virtualization and security
- The attack surface

Docker security

- Docker and security
- Docker security best practices

Lab – Static analysis of Docker images

- Hardening Docker containers

Kubernetes security

- The Kubernetes architecture and security
- Securing Kubernetes hosts
- Best practices for Kubernetes access control
- Protecting Kubernetes deployments at runtime

Lab – Kubernetes security testing

- Kubernetes security tools

Lab – Scanning a Kubernetes cluster for vulnerabilities

Case study – Azurescape

Microservices security

- Security challenges for microservices
- API gateways and security
- Securing service mesh deployments

Time and state

- Race conditions
  - File race condition
    - Time of check to time of usage – TOCTTOU
    - TOCTTOU attacks in practice

    Case study – Arbitrary file deletion via TOCTTOU in N-Able Agent

    - Insecure temporary file

› Psychological acceptability

Authentication

- Authentication basics
- Multi-factor authentication (MFA)

Case study – The InfinityGauntlet attack

- Passwordless solutions
- Time-based One Time Passwords (TOTP)
- Authentication weaknesses
- Spoofing on the Web
- User interface best practices

Lab – On-line password brute forcing

- Password change
- Password recovery issues
- Password recovery best practices

Lab – Password reset weakness

Case study – GitLab account takeover

- Anti-automation
- Password policy
  - NIST authenticator requirements for memorized secrets
  - Password hardening
  - Using passphrases

Day 4

› Work factor

Password management

- Storing account passwords
- Password in transit

Lab – Is just hashing passwords enough?

- Dictionary attacks and brute forcing
- Salting
- Adaptive hash functions for password storage

Lab – Using adaptive hash functions in JCA

- Using password cracking tools
- Password cracking in Windows

Lab – Password audit with John the Ripper

› Compromise recording

Logging and monitoring

- Logging and monitoring principles
- Insufficient logging

- *Case study – Plaintext passwords at Facebook*
- Logging best practices
- OWASP security logging library
- Monitoring best practices
- Firewalls and Web Application Firewalls (WAF)
- Intrusion detection and prevention
- *Case study – The Marriott Starwood data breach*

› Secure coding beyond the high-level principles

Code quality

- Code quality and security
- Data handling
  - Initialization and cleanup
    - Constructors and destructors
    - Class initialization cycles
    - *Lab – Initialization cycles*
  - Unreleased resource
    - The finalize() method – best practices
- Object oriented programming pitfalls
  - Inheritance and overriding
  - Mutability
    - *Lab – Mutable object*
  - Cloning
- Serialization
  - Serialization and deserialization challenges
  - Confidentiality – serializing sensitive data
  - Confidentiality – serialization best practices
  - Integrity – deserializing untrusted streams
  - Integrity – deserialization best practices
  - Using readObject
  - Look ahead deserialization
  - Property Oriented Programming (POP)
    - Creating a POP payload
    - *Lab – Creating a POP payload*
    - *Lab – Using the POP payload*
    - *Case study – Deserialization RCEs in NextGen Mirth Connect*
    - Summary – POP best practices
  - DoS with deserialization
  - *Lab – Billion laughs with deserialization*

Denial of service

- Flooding
- Resource exhaustion
- Sustained client engagement
- Denial of service problems in various APIs
- Infinite loop
- Economic Denial of Sustainability (EDoS)
- Algorithmic complexity issues
  - Regular expression denial of service (ReDoS)
    - *Lab – ReDoS*
    - Dealing with ReDoS

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop
- Secure design principles of Saltzer and Schroeder

And now what?

- Software security sources and further reading
- Resources