

Secure coding and security testing in C# for DevSecOps

CYDCsDop3d | 3 days | On-site or online | Hands-on

The course provides an in-depth exploration of security concerns and best practices tailored specifically for DevOps engineers working on C# software on the Azure cloud platform. Starting off from the foundations of cybersecurity, you will understand the consequences of insecure code by examining threats through the lens of the CIA triad.

In the main part of the material, you will go through the various security issues outlined in the OWASP Top Ten with a focus on DevSecOps issues - identity management in microservice and cloud environments, secure Azure configuration, securing CI / CD build processes, secrets management, and logging and monitoring. Finally, you'll explore cloud security with a focus on security automation and tooling in Azure, the security of containers and container orchestration (Docker, Kubernetes), microservices, and Infrastructure as Code tools (Azure Resource Manager, Terraform), and security testing tools relevant for DevSecOps.

These modules go beyond just theory. Not only do they show vulnerabilities, their consequences, and corresponding best practices, but - through hands-on labs and real-world case studies - they offer practical experience in identifying, exploiting, and mitigating these security risks.

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

So that nothing unexpected happens.

Nothing.

Cyber security skills and drills



23 LABS



8 CASE STUDIES

Audience

C# architects, developers and testers working on web applications and Azure

Group size

12 participants

Preparedness

DevSecOps, General C# and Web development, testing and QA

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Cloud security
- Security testing
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of C#
- Going beyond the low hanging fruits
- Managing vulnerabilities in third party components
- Learn to deal with cloud infrastructure security
- Understand cloud security specialties
- Understanding security testing methodology and approaches
- Getting familiar with security testing techniques and tools

Table of contents

Day 1

› Cyber security basics

What is security?

Threat and risk

[Cyber security threat types – the CIA triad](#)

Consequences of insecure software

› [The OWASP Top Ten 2021](#)

A01 – Broken Access Control

- Access control basics
- Authorization models in microservices
- Microservices authorization pitfalls and best practices
- Testing for authorization issues

• Confused deputy

- Insecure direct object reference (IDOR)
- Path traversal

 *Lab – Insecure Direct Object Reference*

- Path traversal best practices
- Authorization bypass through user-controlled keys

 *Case study – Authorization bypass on Facebook*

 *Lab – Horizontal authorization*

- Testing for confused deputy weaknesses

• [Cross-site Request Forgery \(CSRF\)](#)

 *Lab – Cross-site Request Forgery*

- CSRF best practices

 *Lab – CSRF protection with tokens*

- Testing for CSRF

A03 – Injection

• [SQL injection](#)

- SQL injection basics

 *Lab – SQL injection*

- Attack techniques

- Content-based blind SQL injection
- Time-based blind SQL injection
- Testing for SQL injection
 - SQL injection tools
- SQL injection best practices
 - Input validation
 - Parameterized queries
 - 🔗 *Lab – Using prepared statements*
 - 📖 *Case study – Hacking Fortnite accounts*
- NoSQL injection
 - NoSQL injection basics
 - NoSQL injection in MongoDB
 - NoSQL injection in DynamoDB
- Code injection
 - OS command injection
 - 🔗 *Lab – Command injection*
 - OS command injection best practices
 - Avoiding command injection with the right APIs
 - 🔗 *Lab – Command injection best practices*
 - 📖 *Case study – Command injection via ping*
 - Testing for command injection

Day 2

› [The OWASP Top Ten 2021](#)

A03 – Injection (continued)

- HTML injection – Cross-site scripting (XSS)
 - [Cross-site scripting basics](#)
 - Cross-site scripting types
 - Persistent cross-site scripting
 - Reflected cross-site scripting
 - Client-side (DOM-based) cross-site scripting
 - 🔗 *Lab – Stored XSS*
 - 🔗 *Lab – Reflected XSS*
 - 📖 *Case study – XSS in Fortnite accounts*
- Testing for XSS
 - Testing for XSS with tools
- XSS protection best practices
 - Protection principles – escaping
 - XSS protection APIs

- Further XSS protection techniques
 - 🔗 *Lab – XSS fix / stored*
 - 🔗 *Lab – XSS fix / reflected*
- Additional protection layers – defense in depth

A04 – Insecure Design

- Client-side security
 - Same Origin Policy
 - Frame sandboxing
 - Cross-Frame Scripting (XFS) attacks
 - 🔗 *Lab – Clickjacking*
 - Clickjacking protection best practices
 - 🔗 *Lab – Using CSP to prevent clickjacking*
 - Testing for client-side security weaknesses

A05 – Security Misconfiguration

- Configuration principles
- Testing for misconfiguration issues
- XML entities
 - DTD and the entities
 - Entity expansion
 - External Entity Attack (XXE)
 - File inclusion with external entities
 - Server-Side Request Forgery with external entities
 - 🔗 *Lab – External entity attack*
 - 📖 *Case study – XXE vulnerability in SAP Store*
 - Preventing XXE
 - 🔗 *Lab – Prohibiting DTD*
 - Testing for XXE and XML entity-related vulnerabilities

A06 – Vulnerable and Outdated Components

- Using vulnerable components
- Vulnerability management
 - 🔗 *Lab – Finding vulnerabilities in third-party components*
- Build security
 - [DevOps, the CI / CD build process and Software Composition Analysis](#)
 - Secure artifact management
 - Secure release management
 - Implementing a secure CI/CD pipeline with DevSecOps
 - Dependency checking in C#
 - 🔗 *Lab – Detecting vulnerable components*

A07 – Identification and Authentication Failures

- Session management

- Session management essentials
- Why do we protect session IDs – Session hijacking
- Session fixation
- Testing for session management issues
- Password management
 - Outbound password management
 - Hard coded passwords
 - Best practices
 - 🔗 *Lab – Hardcoded password*
 - Storing secrets in Azure Key Vault
 - Key Vault types and Hardware Security Modules
 - Azure Key Vault security
 - Access control to Azure Key Vault
- Identity and access management (IAM)
 - Identity and access management in Azure
 - Azure Active Directory
 - Multi-factor authentication and password management with Azure
 - The Azure AD hybrid identity model

A09 – Security Logging and Monitoring Failures

- Logging and monitoring principles
- Log forging
- Log forging – best practices
- Logging best practices
- Testing for logging issues
- Detection and monitoring in Azure
 - Utilizing Azure monitoring for security
 - The Azure Security Center
 - The Azure Application Gateway WAF

› Cyber security basics

Security in the Software Development Lifecycle

- Securing the SDLC
- OWASP Software Assurance Maturity Model (SAMM)
- Microsoft Security Development Lifecycle (MS SDL)
- DevSecOps
 - DevSecOps basics
 - Secure coding, security testing, and DevSecOps
 - NIST (NCCoE) DevSecOps standards and recommendations
 - OWASP DevSecOps Maturity Model (DSOMM)

- OWASP DevSecOps Verification Standard (DSOVS)

Day 3

> Cloud security

Cloud security basics

- Cloud infrastructure basics
- Cloud architectures and security
- The Cloud Cube Model and Zero Trust Architecture
- Attack surface in the cloud

 *Case study – ChaosDB vulnerability in Azure Cosmos DB*


Cloud security standards and best practices

- CSA controls and the Cloud Controls Matrix (CCM)
- Other standards

Azure security

- Security considerations for Azure
 - Azure and security
 - Azure security features
 - The Azure shared responsibility model
 - Azure cloud compliance
 - Azure hardening
 - Security tools for Azure

Container security

- 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
- Kubernetes security
 - The Kubernetes architecture and security
 - Securing Kubernetes hosts
 - Best practices for Kubernetes access control
 - Building secure Kubernetes images

- Secure deployment of Kubernetes containers
- Protecting Kubernetes deployments at runtime
- 📖 *Case study – Azurescape*
- Microservices security
 - Security challenges for microservices
 - API gateways and security
 - Securing service mesh deployments
- Infrastructure as Code
 - Infrastructure as Code security
 - Azure Resource Manager security
 - Azure Resource Manager threats and best practices
 - Terraform security
 - Terraform threats and best practices

› Security testing

Security testing vs functional testing

Manual and automated methods

Black box, white box, and hybrid testing

Security testing methodology

- Security testing – goals and methodologies
- Overview of security testing processes
- Identifying assets
- Assigning security requirements

🔗 *Lab – Identifying and rating assets*

- Threat modeling
 - SDL threat modeling
 - Mapping STRIDE to DFD
 - DFD example
 - Attack trees
 - Attack tree example
 - Risk analysis

Security testing techniques and tools

- Code analysis
 - Static Application Security Testing (SAST)

🔗 *Lab – Using static analysis tools*

- Dynamic analysis
 - Security testing at runtime
 - [Penetration testing](#)

- Stress testing
- Dynamic analysis tools
 - Dynamic Application Security Testing (DAST)
 - IAST and DevSecOps
 - Web vulnerability scanners
 - 🔗 *Lab – Using web vulnerability scanners*
 - Fuzzing

DevSecOps security testing

- Cloud infrastructure scanning
- IaC scanning
- Secrets scanning

> 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
- .NET and C# resources
- Security testing resources