Code responsibly with generative AI in C#

CYDCsWeb3dCop | 3 days | On-site or online | Hands-on

Generative AI is transforming the software industry, with tools like GitHub Copilot and Codeium enabling developers to achieve unprecedented levels of efficiency. While this is exciting progress, it also raises important concerns, encouraging stakeholders to approach these technologies with care. Current AI tools often lack the nuanced understanding necessary to address subtle, yet critical aspects of software development, particularly in the domain of security.

This course provides a comprehensive insight into the responsible use of generative AI in coding. Participants delve into topics in software development that are most likely to be impacted by careless use of generative AI, including authentication, authorization, and cryptography. The curriculum also includes an analysis of how AI tools like Copilot handle secure coding practices related to key vulnerabilities outlined in the OWASP Top Ten, such as path traversal, SQL injection, or cross-site scripting.

Through hands-on learning and experimenting, participants will get a solid understanding of both the strengths and limitations of AI-assisted development. In addition, case studies of real-world incidents showcase the consequences of insecure code and demonstrate the dual nature of generative AI as both a resource and a potential risk.

By the end of the course, developers will be equipped with the knowledge and skills to integrate AI tools into the software development lifecycle responsibly, enhancing efficiency without compromising security or product quality.

Skills and drills

33 LABS

14 CASE STUDIES

Audience

C# developers using Copilot or other GenAI tools

Group size

12 participants

Preparedness

General C# and Web development

Outline

- Coding responsibly with GenAI
- The OWASP Top Ten from Copilot's perspective
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Understanding the essentials of responsible AI
- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in C#
- 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
- All this put into the context of GitHub Copilot

Table of contents

Day 1

- Elementary algorithms
  - Hashing
  - Hashing basics
  - Hashing in C#
  - *Lab – Hashing in C# (exploring with Copilot)*
  - Random number generation
  - Pseudo random number generators (PRNGs)
  - Cryptographically secure PRNGs
  - Weak and strong PRNGs
  - Using random numbers in C#
  - *Lab – Using random numbers in C# (exploring with Copilot)*
- Confidentiality protection
  - Symmetric encryption
  - Block ciphers
  - Modes of operation
  - Modes of operation and IV – best practices
  - Symmetric encryption in C#
  - Symmetric encryption in C# with streams
  - *Lab – Symmetric encryption in C# (exploring with Copilot)*
  - *Case study – Padding oracle used in RCE against Citrix ShareFile*
  - Asymmetric encryption
  - Combining symmetric and asymmetric algorithms
  - Key exchange and agreement
  - Key exchange
  - Diffie-Hellman key agreement algorithm
  - Key exchange pitfalls and best practices

Day 2

› The OWASP Top Ten from Copilot's perspective

A03 - Injection

- Injection principles
- Injection attacks
- 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*
    - *Lab – Experimenting with SQL injection in Copilot*
    - *Case study – SQL injection leading to RCE in Ivanti Endpoint Manager*
- 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*
        - *Lab – Experimenting with command injection in Copilot*
        - *Case study – Command injection in Ruckus*
- 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 to RCE in Azure Service Fabric*
    - XSS protection best practices
        - Protection principles - escaping
        - XSS protection APIs
        - Further XSS protection techniques
        - *Lab – XSS fix / stored (exploring with Copilot)*
        - *Lab – XSS fix / reflected (exploring with Copilot)*

A04 - Insecure Design

- The STRIDE model of threats
- 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
- Client-side security
    - Frame sandboxing
        - Cross-Frame Scripting (XFS) attacks
        - *Lab - Clickjacking*

- Clickjacking beyond hijacking a click
- Clickjacking protection best practices
- *Lab – Using CSP to prevent clickjacking (exploring with Copilot)*

A05 - Security Misconfiguration

- Configuration principles
- 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*
  - Preventing XXE
  - *Lab – Prohibiting DTD*
  - *Case study – XXE vulnerability in SharePoint*
  - *Lab – Experimenting with XXE in Copilot*

Day 3

› The OWASP Top Ten from Copilot's perspective

A06 - Vulnerable and Outdated Components

- Using vulnerable components
- Untrusted functionality import
- *Case study – The Polyfill.io supply chain attack*
- Vulnerability management
  - *Lab – Finding vulnerabilities in third-party components*
  - DevOps, the CI / CD build process and Software Composition Analysis
  - Dependency checking in C#
  - *Lab – Detecting vulnerable components*
- Security of AI generated code
  - Practical attacks against code generation tools
  - Dependency hallucination via generative AI
  - *Case study – A history of GitHub Copilot weaknesses (up to mid 2024)*

A07 - Identification and Authentication Failures

- Authentication
  - Authentication basics
  - Multi-factor authentication (MFA)
  - *Case study – The InfinityGauntlet attack*

- Password management
  - Inbound password management
    - Storing account passwords
    - *Lab – Is just hashing passwords enough?*
    - Dictionary attacks and brute forcing
    - Salting
    - Adaptive hash functions for password storage
    - *Lab – Using adaptive hash functions in C#*
    - *Lab – Using adaptive hash functions in Copilot*
    - *Case study – Veeam missing authentication and cleartext password storage*
    - Password policy
    - NIST authenticator requirements for memorized secrets

A08 - Software and Data Integrity Failures

- Integrity protection
  - Message Authentication Code (MAC)
    - Calculating HMAC in C#
    - *Lab – Calculating MAC in C#*
  - Digital signature
    - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
    - Digital signature in C#
    - *Lab – Digital signature with ECDSA in C#*
- Subresource integrity
  - Importing JavaScript
  - *Lab – Importing JavaScript (exploring with Copilot)*
  - *Case study – The British Airways data breach*
- Insecure deserialization
  - Serialization and deserialization challenges
  - Integrity – deserializing untrusted streams
  - Integrity – deserialization best practices
  - Look ahead deserialization
  - Property Oriented Programming (POP)
    - Creating a POP payload
    - *Lab – Creating a POP payload*
    - *Lab – Using the POP payload*
    - *Case study – Deserialization RCE in Veeam*
    - Summary – POP best practices
    - *Lab – Preventing POP with Copilot*

A10 - Server-side Request Forgery (SSRF)

- Server-side Request Forgery (SSRF)

Case study – SSRF and the Capital One breach

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- .NET and C# resources
- Responsible AI principles in software development
- Generative AI – Resources and additional guidance