Secure coding in C# with generative AI (desktop applications)

CYDCsDsk3dGenAI | 3 days | On-site or online | Hands-on

Generative AI is transforming the software industry, with tools like Claude Code, GitHub Copilot or others, 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 Claude Code, GitHub Copilot or others 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

30 LABS

16 CASE STUDIES

Audience

C# developers using GenAI tools

Group size

12 participants

Preparedness

General C# development

Outline

- Coding responsibly with GenAI
- Input validation
- Security features
- Denial of service
- Using vulnerable components
- Cryptography for developers
- Common software security weaknesses
- Wrap up

Standards and references

CWE and Fortify Taxonomy

What you'll have learned

- Understanding the essentials of responsible AI
- Getting familiar with essential cyber security concepts
- Input validation approaches and principles
- Identify vulnerabilities and their consequences
- Learn the security best practices in C#
- Correctly implementing various security features
- Managing vulnerabilities in third party components
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in C#

Table of contents

Case study – The Stockholm Stock Exchange
Lab – Signed / unsigned confusion
Lab – Experimenting with signed / unsigned confusion in GenAI
- Integer truncation
- Best practices
  - Upcasting
  - Precondition testing
  - Postcondition testing
  - Integer handling in C#

Lab – Checked arithmetics
Lab – Experimenting with integer overflow in GenAI

Files and streams

- Path traversal

Lab – Path traversal
- Additional challenges in Windows

Case study – File spoofing in WinRAR
- Path traversal best practices

Lab – Path canonicalization
Lab – Experimenting with path traversal in GenAI

Day 2

› Input validation

Unsafe reflection

- Reflection without validation

Lab – Unsafe reflection
Lab – Experimenting with unsafe reflection in GenAI

Unsafe native code

- Native code dependence

Lab – Unsafe native code
- Best practices for dealing with native code

› Security features

Authentication

- Authentication basics

- Multi-factor authentication (MFA)

Case study – The InfinityGauntlet attack

- Time-based One Time Passwords (TOTP)
- 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 C#

Lab – Experimenting with adaptive hash functions in GenAI

Case study – Veeam missing authentication and cleartext password storage

- Password policy
  - NIST authenticator requirements for memorized secrets
  - Password database migration
  - Hard coded passwords
  - Best practices

Lab – Hardcoded password

  - Protecting sensitive information in memory
    - Challenges in protecting memory

Case study – Microsoft secret key theft via dump files

    - Storing sensitive data in memory

Case study – KeePass password leakage via strings

Information exposure

- Exposure through extracted data and aggregation

Case study – Strava data exposure

Platform security

- .NET platform security
  - Protecting .NET code and applications
    - Code signing

› Denial of service

Flooding

Resource exhaustion

Algorithmic complexity issues

- Regular expression denial of service (ReDoS)
  - *Lab – ReDoS*
  - *Lab – Experimenting with ReDoS in GenAI*
  - Dealing with ReDoS

› Using vulnerable components

Case study – The Polyfill.io supply chain attack

Vulnerability management

Lab – Finding vulnerabilities in third-party 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-2025)

Case study – Agentic coding and code security (mid-2025 onward)

Day 3

› Cryptography for developers

Cryptography basics

Crypto APIs in C#

Elementary algorithms

- Hashing
  - Hashing basics
  - Hashing in C#
  - *Lab – Hashing in C#*
- Random number generation
  - Pseudo random number generators (PRNGs)
  - Cryptographically secure PRNGs
  - Weak PRNGs
  - Using random numbers in C#
  - *Lab – Using random numbers in C#*

Case study – Equifax credit account freeze

Confidentiality protection

- Symmetric encryption
  - Block ciphers
  - Modes of operation
  - Modes of operation and IV – best practices
  - Symmetric encryption in C#

  Lab – Symmetric encryption in C#

  Case study – Padding oracle used in RCE against Citrix ShareFile
- Asymmetric encryption
  - The RSA algorithm
    - RSA in C#
- Combining symmetric and asymmetric algorithms
- Key exchange and agreement
  - Key exchange
  - Diffie-Hellman key agreement algorithm

Integrity protection

- Message Authentication Code (MAC)
  - Calculating HMAC in C#

  Lab – Calculating MAC in C#
- Digital signature
  - Digital signature with RSA
  - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
  - Digital signature in C#

    Lab – Digital signature with ECDSA in C#

› Common software security weaknesses

Code quality

- Code quality and security
- Data handling
  - Initialization and cleanup
    - Class initialization cycles

    Lab – Initialization cycles
- Object oriented programming pitfalls
  - Inheritance and overriding
  - Mutability

    Lab – Mutable object

- Serialization
  - 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*

› 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
- Generative AI – Resources and additional guidance