Secure coding in Python with generative AI

CYDPyWeb3dGenAI | 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

29 LABS

19 CASE STUDIES

Audience

Python developers using GenAI tools

Group size

12 participants

Preparedness

General Python and Web development

Outline

- Coding responsibly with GenAI
- The OWASP Top Ten 2025
- 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 Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of Python
- Going beyond the low hanging fruits
- Managing vulnerabilities in third party components
- Input validation approaches and principles

Table of contents

  - Configuring Flask
- Web security configuration issues
  - Content Security Policy
  - Fetch directives
  - Source allowlisting
  - Strict CSP: using nonces and hashes
  - CSP best practices
- Cookie security
  - Cookie attributes
- Secrets management
  - Hard coded passwords
  - Best practices
  - *Lab – Hardcoded password*
- 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 Ivanti products*
    - *Lab – Experimenting with XXE in GenAI*

Day 2

› The OWASP Top Ten 2025

A03 - Software Supply Chain Failures

- Using vulnerable components
- Assessing the environment
- Hardening
- Untrusted functionality import
- Malicious packages in Python
- Supply chain security and the Software Bill of Materials (SBOM)
- SBOM examples
- *Case study – The Polyfill.io supply chain attack*
- Vulnerability management

- Patch management
- Vulnerability management
- Vulnerability databases

Lab – Finding vulnerabilities in third-party components

- DevOps, the CI / CD build process and Software Composition Analysis
- Dependency checking in Python

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-2025)

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

A05 - Injection

- Input validation
  - Input validation principles
  - Denylists and allowlists

Case study – Denylist failure in urllib.parse.urlparse()

  - What to validate – the attack surface
  - Where to validate – defense in depth
  - When to validate – validation vs transformations
- 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

Case study – SQL injection against US airport security

- Code injection
  - Code injection via input()
  - 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 GenAI

Case study – Shellshock

Lab - Shellshock

- *Case study – Command injection in Ivanti security appliances*

- 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 Teltonika routers*
  - XSS protection best practices
    - Protection principles - escaping
    - *Lab – XSS fix / stored*
    - *Lab – XSS fix / reflected*
    - *Case study – XSS vulnerabilities in DrayTek Vigor routers*

Day 3

› The OWASP Top Ten 2025

A06 - 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 protection best practices
    - *Lab – Using CSP to prevent clickjacking*

A07 - Authentication Failures

- Authentication
  - Authentication basics

- Multi-factor authentication (MFA)

Case study – The InfinityGauntlet attack

- 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 Python

Lab – Experimenting with adaptive hash functions in GenAI

- Password policy
  - NIST authenticator requirements for memorized secrets
  - Password database migration

A08 - Software and Data Integrity Failures

- Subresource integrity
  - Importing JavaScript

Lab – Importing JavaScript

Case study – The British Airways data breach

- Insecure deserialization
  - Serialization and deserialization challenges
  - Integrity – deserializing untrusted streams
  - Deserialization with pickle

Lab – Deserializing with Pickle

Case study – The security of the machine learning supply chain

Case study – The first wave of supply chain attacks: RCE via pickle (2022)

Case study – Compromising the Hugging Face Hub repository

  - Integrity – deserialization best practices

A10 - Mishandling of Exceptional Conditions

- Error and exception handling principles
- Error handling
  - Returning a misleading status code
  - Information exposure through error reporting
    - Information leakage via error pages

Lab – Flask information leakage

Case study – Information leakage via errors in Apache Superset

- Exception handling
  - In the except block. And now what?
  - Empty except block

Lab – Exception handling mess

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- Python resources
- Generative AI – Resources and additional guidance