Code responsibly with generative AI in Python

CYDPyWeb3dCop | 3 days | On-site or online | Hands-on

Your Web application written in Python works as intended, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -1 or -2^31? Because that's what the bad guys will do – and the list is far from complete.

Handling security needs a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hands-on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

The curriculum goes through the common Web application security issues following the OWASP Top Ten but goes far beyond it both in coverage and the details.

All this is put in the context of Python, and extended by core programming issues, discussing security pitfalls of the programming language.

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

So that nothing unexpected happens.

Nothing.

This variant of the course deals extensively with how certain security problems in code are handled by GitHub Copilot.

Through a number of hands-on labs participants will get first hand experience about how to use Copilot responsibly, and how to prompt it to generate the most secure code. In some cases it is trivial, but in most of the cases it is not; and in yet some other cases it is basically impossible.

At the same time, the labs provide general experience with using Copilot in everyday coding practice – what you can expect from it, and what are those areas where you shouldn't rely on it.

Cyber security skills and drills

32 LABS

13 CASE STUDIES

Audience

Python developers working on Web applications

Group size

12 participants

Preparedness

General Python and Web development

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in Python
- 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

Table of contents

Day 1

- Cryptography in Python
- Elementary algorithms
  - Hashing
  - Hashing basics
  - Hashing in Python
  - *Lab – Hashing in Python*
  - Random number generation
  - Pseudo random number generators (PRNGs)
  - Cryptographically secure PRNGs
  - Weak PRNGs
  - Using random numbers
  - *Lab – Using random numbers in Python*
  - *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 Python
  - *Lab – Symmetric encryption in Python*
  - Asymmetric encryption
  - Combining symmetric and asymmetric algorithms

Day 2

› The OWASP Top Ten 2021

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*
    - Database defense in depth

    - *Case study – Hacking Fortnite accounts*
- 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 Copilot*
    - *Case study – Shellshock*
    - *Lab - Shellshock*
    - *Case study – Command injection via ping*
- 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*
  - XSS protection best practices
    - Protection principles - escaping
    - XSS protection APIs in Python
    - XSS protection in Jinja2
    - *Lab – XSS fix / stored*
    - *Lab – XSS fix / reflected*

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
  - Same Origin Policy
    - Simple request
    - Preflight request

    - Cross-Origin Resource Sharing (CORS)
    - *Lab – Same-origin policy demo*
  - Frame sandboxing
    - Cross-Frame Scripting (XFS) attacks
    - *Lab - Clickjacking*
    - Clickjacking beyond hijacking a click
    - Clickjacking protection best practices
    - *Lab – Using CSP to prevent clickjacking*

Day 3

› The OWASP Top Ten 2021

A05 - Security Misconfiguration

- Configuration principles
- Server misconfiguration
- Python configuration best practices
  - Configuring Flask
- Cookie security
  - Cookie attributes
- 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*
    - *Lab – Experimenting with XXE in Copilot*

A06 - Vulnerable and Outdated Components

- Using vulnerable components
- Untrusted functionality import
- Malicious packages in Python
- 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 2024)*

A07 - Identification and Authentication Failures

- Authentication
  - Authentication basics
  - Multi-factor authentication (MFA)
  - *Case study – PayPal 2FA bypass*
- Password management
  - Inbound 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 – Using adaptive hash functions in Copilot*
    - Password policy
    - NIST authenticator requirements for memorized secrets
    - *Case study – The Ashley Madison data breach*
    - *The ultimate crack*
    - *Exploitation and the lessons learned*
    - Password database migration

A08 - Software and Data Integrity Failures

- Integrity protection
  - Message Authentication Code (MAC)
    - Calculating HMAC in Python
    - *Lab – Calculating MAC in Python*
  - Digital signature
    - Digital signature in Python
- Subresource integrity
  - Importing JavaScript
  - *Lab – Importing JavaScript*
  - *Case study – The British Airways data breach*

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
- Python resources