Code responsibly with generative AI in C++

CYDCpp3dCop | 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

28 LABS

9 CASE STUDIES

Audience

C/C++ developers using Copilot or other GenAI tools

Group size

12 participants

Preparedness

General C++ and C development

Outline

- Coding responsibly with GenAI
- Memory management vulnerabilities
- Memory management hardening
- Common software security weaknesses
- Using vulnerable components
- Wrap up

Standards and references

SEI CERT, CWE and Fortify Taxonomy

What you'll have learned

- Understanding the essentials of responsible AI
- Getting familiar with essential cyber security concepts
- Correctly implementing various security features
- Identify vulnerabilities and their consequences
- Learn the security best practices in C++
- Managing vulnerabilities in third party components
- Input validation approaches and principles
- All this put into the context of GitHub Copilot

Table of contents

- *Case study – Stack BOF in FriendlyName handling of the Wemo Smart Plug*
- Pointer manipulation
  - Modification of jump tables
  - Overwriting function pointers

Best practices and some typical mistakes

- Unsafe functions
- Dealing with unsafe functions

Lab – Fixing buffer overflow (exploring with Copilot)

- Using std::string in C++
- Manipulating C-style strings in C++
- Malicious string termination

Lab – String termination confusion (exploring with Copilot)

- String length calculation mistakes

Day 2

› Memory management hardening

Securing the toolchain

- Securing the toolchain in C++
- Using FORTIFY_SOURCE

Lab – Effects of FORTIFY

- AddressSanitizer (ASan)
  - Using AddressSanitizer (ASan)

Lab – Using AddressSanitizer

- Stack smashing protection
  - Detecting BoF with a stack canary
  - Argument cloning
  - Stack smashing protection on various platforms
  - SSP changes to the prologue and epilogue

Lab – Effects of stack smashing protection

Runtime protections

- Runtime instrumentation
- Address Space Layout Randomization (ASLR)
  - ASLR on various platforms

Lab – Effects of ASLR

- Circumventing ASLR – NOP sleds

  - Circumventing ASLR – memory leakage
- Non-executable memory areas
  - The NX bit
  - Write XOR Execute (W^X)
  - NX on various platforms
  - *Lab – Effects of NX*
  - NX circumvention – Code reuse attacks
    - Return-to-libc / arc injection
  - Return Oriented Programming (ROP)
    - Protection against ROP
    - *Case study – Systematic exploitation of a MediaTek buffer overflow*

› Common software security weaknesses

Security features

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

Code quality

- Code quality and security
- Data handling
  - Type mismatch
  - *Lab – Type mismatch (exploring with Copilot)*
  - Initialization and cleanup
    - Constructors and destructors
    - Initialization of static objects
    - *Lab – Initialization cycles (exploring with Copilot)*
  - Unreleased resource
    - Array disposal in C++
    - *Lab – Mixing delete and delete[] (exploring with Copilot)*
- Object oriented programming pitfalls
  - Accessibility modifiers
    - Are accessibility modifiers a security feature?

- Inheritance and object slicing
- Implementing the copy operator
- The copy operator and mutability
- Mutability
  - Mutable predicate function objects
  - *Lab – Mutable predicate function object*

› Using 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)*

Day 3

› Common software security weaknesses

Input validation

- Input validation principles
- Denylists and allowlists
- What to validate – the attack surface
- Where to validate – defense in depth
- When to validate – validation vs transformations
- Injection
  - 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 Zyxel IKE packet decoder*
- Integer handling problems
  - Representing signed numbers
  - Integer visualization
  - Integer promotion
  - Integer overflow
  - *Lab – Integer overflow*
  - *Lab – Experimenting with integer overflow in Copilot*

- *Case study – Integer underflows in IPv6 handling of tcpip.sys*
- Signed / unsigned confusion
- *Case study – Signed/unsigned confusion DoS in DrayTek Vigor routers*
- *Lab – Signed / unsigned confusion*
- *Lab – Experimenting with signed / unsigned confusion in Copilot*
- Integer truncation
- *Lab – Integer truncation*
- *Lab – Experimenting with integer truncation in Copilot*
- *Case study – WannaCry*
- Best practices
  - Upcasting
  - Precondition testing
  - Postcondition testing
  - Using big integer libraries
  - UBSan changes to arithmetics
  - *Lab – Handling integer overflow on the toolchain level in C++*
  - Best practices in C++
  - *Lab – Integer handling best practices in C++*
  - *Case study – Integer check failure in Skia*
- Files and streams
  - Path traversal
  - *Lab – Path traversal*
  - Path traversal-related examples
  - *Case study – File spoofing in WinRAR*
  - Virtual resources
  - Path traversal best practices
  - *Lab – Path canonicalization*
  - *Lab – Experimenting with path traversal in Copilot*

› 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
- C and C++ resources
- Responsible AI principles in software development
- Generative AI – Resources and additional guidance