

Code responsibly with generative AI in C++ (CRWGAIC++)

ID CRWGAIC++ Price CHF 2,250.—(excl. VAT) Duration 3 days

Who should attend

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

Prerequisites

General C++ and C development

Course Objectives

- 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

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

Course Content

Day 1

Coding responsibly with GenAI

- What is responsible AI?
- What is security?
- Threat and risk
- Cyber security threat types – the CIA triad
- Cyber security threat types – the STRIDE model
- Consequences of insecure software
- Security and responsible AI in software development
- GenAI tools in coding: Copilot, Codeium and others

Memory management vulnerabilities

- Assembly basics and calling conventions
 - x64 assembly essentials
 - Registers and addressing
 - Most common instructions
 - Calling conventions on x64
 - Calling convention – what it is all about
 - Calling convention on x64
- The stack frame
- Stacked function calls
- Buffer overflow
 - Memory management and security
 - Buffer security issues
 - Buffer overflow on the stack
 - Buffer overflow on the stack – stack smashing
 - Exploitation – Hijacking the control flow
 - Lab – Buffer overflow 101, code reuse
 - Exploitation – Arbitrary code execution
 - Injecting shellcode
 - Lab – Code injection, exploitation with shellcode
 - 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)

Code responsibly with generative AI in C++ (CRWGAIC++)

- 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
- 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)
- [/list]

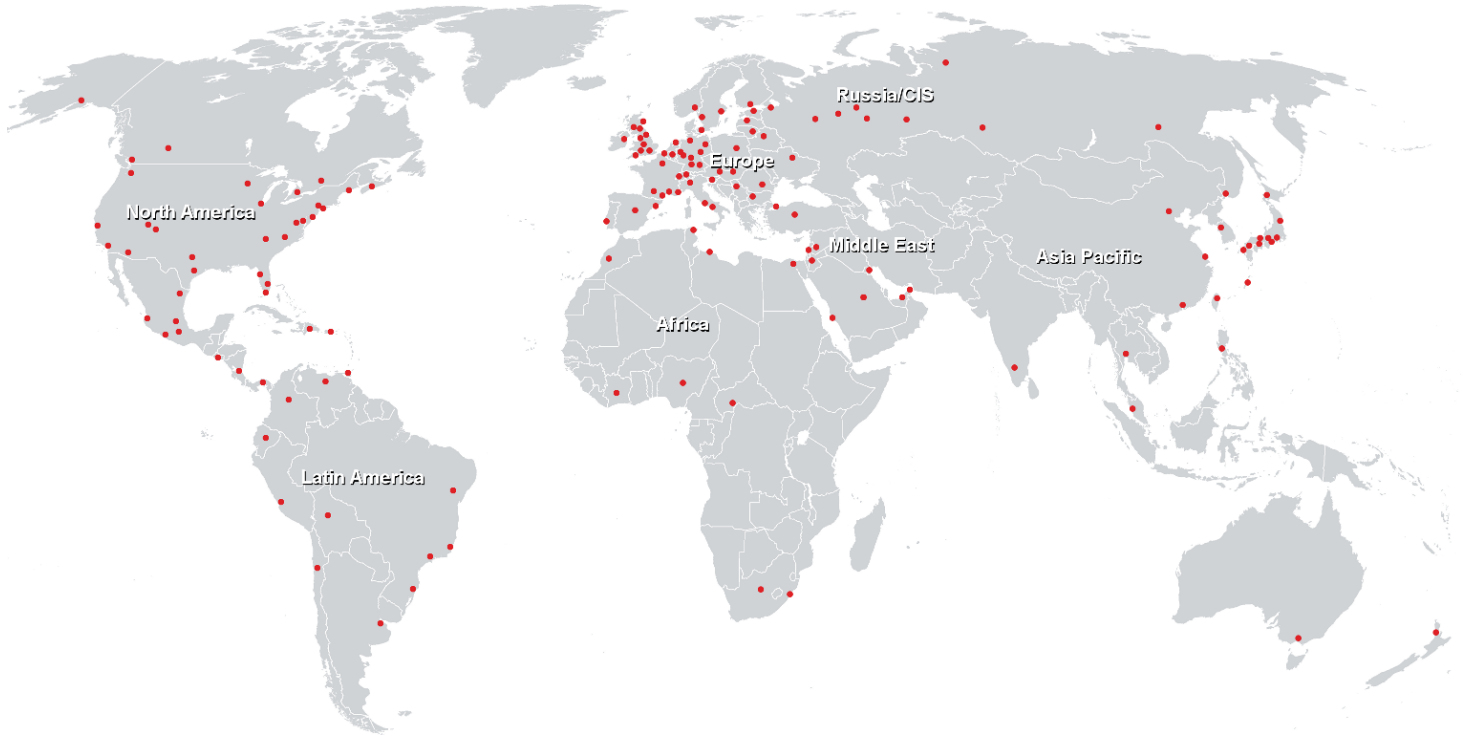
Day 3

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

Code responsibly with generative AI in C++ (CRWGAIC++)

Training Centres worldwide



Fast Lane Institute for Knowledge Transfer (Switzerland) AG

Husacherstrasse 3
CH-8304 Wallisellen
Tel. +41 44 832 50 80

info@flane.ch, <https://www.flane.ch>