# Secure coding in C++

**CYDCpp3d | 3 days | On-site or online | Hands-on**

Embark on a comprehensive exploration of cybersecurity and secure coding practices in this intensive three-day course. It is primarily focusing on C++, but also integrates some C concepts. Based on a primer on machine code, assembly, and memory overlay (Intel and ARM versions available), the curriculum addresses critical security issues related to memory management. Various protection techniques on the level of source code, compiler, OS or hardware are discussed - such as stack smashing protection, ASLR or the non-execution bit - to understand how they work and make clear what we can and what we can't expect from them.

The various secure coding subjects are aligned to common software security weakness categories, such as security features, error handling or code quality. Many of the weaknesses are, however, linked to missing or improper input validation. In this category you'll learn about injection, the surprising world of integer overflows, and about handling file names correctly to avoid path traversal.

Through hands-on labs and real-world case studies, you will navigate the details of secure coding practices to get essential approaches and skills in cybersecurity.

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

So that nothing unexpected happens.

Nothing.

# Skills and drills

24 LABS

10 CASE STUDIES

## Audience

C/C++ developers

## Group size

12 participants

## Preparedness

General C++ and C development

## Outline

- Cyber security basics
- Memory management vulnerabilities
- Memory management hardening
- Common software security weaknesses
- Wrap up

## Standards and references

SEI CERT, CWE and Fortify Taxonomy

## What you'll have learned

- Getting familiar with essential cyber security concepts
- Correctly implementing various security features
- Identify vulnerabilities and their consequences
- Learn the security best practices in C and C++
- Input validation approaches and principles

# Table of contents


## Day 1

### › Cyber security basics

What is security?
Threat and risk
Cyber security threat types – the CIA triad
Cyber security threat types – the STRIDE model
Consequences of insecure software

### › 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*
- Buffer overflow on the heap
  - Unsafe unlinking

- *Case study – Heap BOF in WinRAR*
- 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*

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

*Lab – String termination confusion*

- String length calculation mistakes

# Day 2

## › Memory management hardening

### Securing the toolchain

- Securing the toolchain in C and C++
- 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
- 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
  - Outbound password management
    - Hard coded passwords
    - Best practices
    - *Lab – Hardcoded password*
    - Protecting sensitive information in memory
    - Challenges in protecting memory
    - Heap inspection
    - *Case study – Microsoft secret key theft via dump files*

## Code quality

- Code quality and security
- Data handling
  - Type mismatch
  - *Lab – Type mismatch*
  - Initialization and cleanup
    - Constructors and destructors
    - Initialization of static objects
    - *Lab – Initialization cycles*
  - Unreleased resource
    - Array disposal in C++
    - *Lab – Mixing delete and delete[]*

- 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*

# 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
- Output sanitization
- Encoding challenges
- Unicode challenges
- Validation with regex
- 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*
    - *Case study – Shellshock*
    - *Lab - Shellshock*
    - *Case study – Command injection in Zyxel IKE packet decoder*
- Integer handling problems
  - Representing signed numbers
  - Integer visualization
  - Integer promotion
  - Integer overflow
  - *Lab – Integer overflow*

- *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*
- Integer truncation
- *Lab – Integer truncation*
- *Case study – WannaCry*
- Best practices
  - Upcasting
  - Precondition testing
  - Postcondition testing
  - 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 best practices
  - *Lab – Path canonicalization*

### Errors

- Error and exception handling principles
- Exception handling
  - In the catch block. And now what?
  - Empty catch block
  - *Lab – Exception handling mess*

## › 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