

**C and C++ Secure Coding**

**Course Number:** SEC-126  
**Duration:** 3 days

**Overview**

Accelebrate's C and C++ Secure Coding training teaches developers the common security weaknesses that allow hackers to attack systems and the best practices needed to prevent these attacks. Attendees learn core programming issues of C and C++ and identify potential security pitfalls and solutions.

**Note:** To ensure ample one-on-one engagement with the instructor, this class is capped at 12 people, overriding Accelebrate’s default cap of 15.

**Prerequisites**

All students must have C/C++ development experience.

**Materials**

All Secure Coding training attendees receive comprehensive courseware.

**Software Needed on Each Student PC**

Attendees will not need to install any software on their computer for this class. The class will be conducted in a remote environment that Accelebrate will provide; students will only need a local computer with a web browser and a stable Internet connection. Any recent version of Microsoft Edge, Mozilla Firefox, or Google Chrome will be fine.

**Objectives**

All attendees will:

* Handle security challenges in your C and C++ code
* Identify vulnerabilities and their consequences
* Learn the security best practices in C and C++

**Outline**

* Cybersecurity Basics
  + What is security?
  + Threat and risk
  + Cybersecurity threat types
  + Consequences of insecure software
  + Constraints and the market
  + The dark side
* Buffer Overflow
  + Assembly basics and calling conventions
    - x64 assembly essentials
    - Registers and addressing
    - Most common instructions
    - Calling conventions on x64
  + Memory management vulnerabilities
    - Memory management and security
    - Vulnerabilities in the real world
  + Buffer security issues
    - Buffer overflow on the stack
    - Buffer overflow on the heap
    - Pointer manipulation
    - Some typical mistakes leading to BoF
  + BoF protection best practices
    - Safe and unsafe functions
    - base\_string and std::string
    - Some less-known dangerous functions
    - Lab – Fixing buffer overflow
    - Securing the toolchain
    - Compiler options and instrumentation
    - Stack smashing protection
    - Runtime protection
    - Address Space Layout Randomization (ASLR)
    - Non-executable memory areas
* Common Software Security Weaknesses
  + Input validation
    - Input validation principles
  + Injection
    - Injection principles
    - Injection attacks
    - Code injection
  + Integer handling problems
    - Representing signed numbers
    - Integer visualization
    - Integer overflow
    - Lab – Integer overflow
    - Signed / unsigned confusion
    - Lab – Signed / unsigned confusion
    - Integer truncation
    - Lab – Integer truncation
    - Case study – Wannacry
    - Best practices
  + Other numeric problems
    - Division by zero
    - Working with floating-point numbers
  + Files and streams
    - Path traversal
    - Path traversal-related examples
    - Lab – Path traversal
    - Link and shortcut following
    - Virtual resources
    - Path traversal best practices
  + Format string issues
    - The problem with printf()
    - Lab – Exploiting format string
* Time and State
  + Race conditions
    - Race condition in object data members
    - File race condition
    - Potential race condition
* Common Software Security Weaknesses
  + Security features
    - Authentication
    - Password management
  + Errors
    - Error and exception handling principles
    - Error handling
    - Exception handling
  + Code quality
    - Type mismatch
    - Lab – Type mismatch
    - Function return values
    - Unreleased resource
    - Object-oriented programming pitfalls
    - Memory and pointers
* Wrap Up
  + Secure coding principles
    - Principles of robust programming by Matt Bishop
    - Secure design principles of Saltzer and Schröder
  + And now what?
    - Further sources and readings
    - C and C++ resources