

**Secure Coding in C and C++ for Medical Devices**

**Course Number:** SEC-136  
**Duration:** 4 days

**Overview**

This Secure Coding in C and C++ for Medical Devices training teaches attendees essential cyber security concepts in the context of the healthcare sector. Students learn secure code best practices in C and C++ and how to leverage common security testing techniques to prevent data breaches.

**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 general C/C++ development knowledge.

**Materials**

All Medical Device Secure Coding training attendees receive comprehensive courseware.

**Software Needed on Each Student PC**

Attendees will not need to install any software on their computers 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 work well.

**Objectives**

* Get familiar with essential cyber security concepts
* Learn about security specialties of the healthcare sector
* Handle security challenges in your C and C++ code
* Identify vulnerabilities and their consequences
* Learn security best practices in C and C++
* Understand security testing methodology and approaches
* Get familiar with common security testing techniques and tools

**Outline**

* Introduction
* Cyber Security Basics
  + What is security?
  + Threat and risk
  + Cyber security threat types
  + Consequences of insecure software
    - Constraints and the market
    - The dark side
  + Regulations and standards
    - Regulations for healthcare information systems
    - Regulations for medical devices
  + Cyber security in the healthcare sector
    - Threats and trends in healthcare
    - Threats to medical devices
    - The problem of legacy systems
* 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
  + Best practices and some typical mistakes
    - Unsafe functions
    - Dealing with unsafe functions
    - What’s the problem with asctime()?
    - Using std::string in C++
  + Some typical mistakes leading to BOF
    - Unterminated strings
    - readlink() and string termination
    - Manipulating C-style strings in C++
    - Malicious string termination
    - String length calculation mistakes
    - Off-by-one errors
    - Off-by-one error in VxWorks TCP ‘Urgent Data’ parsing
    - Allocating nothing
* Common Software Security Weaknesses
  + Security features
    - Authentication
    - Password management
  + Input validation principles
  + Blacklists and whitelists
  + Data validation techniques
  + Case study: Missing input validation in Natus Xltek NeuroWorks 8
  + What to validate: the attack surface
  + Where to validate: defense in depth
  + How to validate: validation vs transformations
  + Output sanitization
  + Encoding challenges
  + Validation with regex
  + Injection
    - Injection principles
    - Injection attacks
    - Code injection
  + Integer handling problems
    - Representing signed numbers
    - Integer visualization
    - Integer promotion
    - Integer overflow
    - Signed / unsigned confusion
    - Integer truncation
    - Case study: WannaCry
    - Best practices
  + Files and streams
    - Path traversal
    - Path traversal-related examples
    - Path traversal best practices
  + Format string issues
    - The problem with printf()
  + Time and state
    - Race conditions
  + Errors
    - Error and exception handling principles
    - Error handling
    - Exception handling
  + Code quality
    - Data handling
    - Control flow
    - Signal handling
    - Object oriented programming pitfalls
    - Memory and pointers
    - File I/O
* Using Vulnerable Components
  + Assessing the environment
  + Hardening
  + Case study: Supply chain attack on Alaris Gateway Workstation
  + Vulnerability management
    - Patch management
    - Bug bounty programs
    - Vulnerability databases
    - Vulnerability rating – CVSS
    - DevOps, the build process and CI / CD
    - Insecure compiler optimization
* Security Testing
  + Security testing vs functional testing
  + Manual and automated methods
  + Security testing techniques and tools
    - Code analysis
    - Dynamic analysis
* Wrap Up
  + Secure coding principles
    - Principles of robust programming by Matt Bishop
    - Secure design principles of Saltzer and Schröder
  + And now what?
    - Software security sources and further reading
    - C and C++ resources
* Conclusion