Secure Coding in C and C++

Category: Networking & Security | **Vendor:** Various

Duration: 24.00 hours (3 days) **19.5 CPD Hours** **Rating:** ⭐ 4.6 (5,878 reviews)

Course Information

Delivery Format: Instructor Led - Online

Course Overview

Your application written in C and C++ works as intended, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -1 or -231? Because that’s what the bad guys will do – and the list is far from complete.

Handling security needs a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hands on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

All this is put in the context of C and C++, and extended by core programming issues, discussing security pitfalls of these languages.

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

So that nothing unexpected happens.

Nothing.

About This Course

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

Who Should Attend

C/C++ developers

Prerequisites & Entry Requirements

General Prerequisites:

General C/C++ development

Learning Outcomes

Upon successful completion of this course, participants will be able to:

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

Detailed Course Outline

DAY 1

Cyber security basics

- What is security?
- Threat and risk
- Cyber security 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

- Calling convention – what it is all about
- The stack frame
- Stacked function calls
- Memory management vulnerabilities

- Memory management and security
- Vulnerabilities in the real world
- 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
- Buffer overflow on the heap

- Unsafe unlinking
- Case study – Heartbleed
- 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

- What’s the problem with asctime()?
- Lab – The problem with asctime()
- Using std::string in C++
- Unterminated strings
- readlink() and string termination
- Manipulating C-style strings in C++
- Malicious string termination
- Lab – String termination confusion
- String length calculation mistakes
- Off-by-one errors
- Allocating nothing

DAY 2

Memory management hardening

- Securing the toolchain

- Securing the toolchain in C and C++
- Compiler warnings and security
- Using FORTIFY_SOURCE
- Lab – Effects of FORTIFY
- AddressSanitizer (ASan)

- Using AddressSanitizer (ASan)
- ASan changes to the prologue
- ASan changes to memory read/write operations
- ASan changes to the epilogue
- 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

- 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

Common software security weaknesses

- Security features

- Authentication

- Authentication basics

- Multi-factor authentication

- Authentication weaknesses – spoofing
- Case study – PayPal 2FA bypass
- 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
- Case study – The Ashley Madison data breach

- The dictionary attack
- The ultimate crack
- Exploitation and the lessons learned
- 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
- Compiler optimization challenges
- Lab – Zeroization challenges

- Sensitive info in non-locked memory

- Code quality

- Data handling

- Type mismatch

- Lab – Type mismatch

- Initialization and cleanup

- Constructors and destructors

- Initialization of static objects

- Lab – Initialization cycles

- Array disposal in C++

- Lab – Mixing delete and delete[]

- Memory and pointers

- Memory and pointer issues

- Pointer handling pitfalls

- Pointer usage in C and C++

- Use after free

- Lab – Use after free

- Lab – Runtime instrumentation

- Double free

- Memory leak

- Smart pointers and RAII

- Smart pointer challenges

DAY 3

Common software security weaknesses

- Input validation

- Input validation principles

- Blacklists and whitelists
- Data validation techniques
- What to validate – the attack surface
- Where to validate – defense in depth
- How to validate – validation vs transformations
- Validation with regex
- Injection

- Injection principles
- Injection attacks
- 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
- Process control – library injection

- DLL hijacking
- Lab – DLL hijacking
- Integer handling problems

- Representing signed numbers
- Integer visualization

- Integer promotion
- Integer overflow
- Lab – Integer overflow
- Signed / unsigned confusion
- Case study – The Stockholm Stock Exchange
- Lab – Signed / unsigned confusion
- Integer truncation
- Lab – Integer truncation
- Case study – WannaCry
- Best practices

- Upcasting
- Precondition testing
- Postcondition testing
- Using big integer libraries
- Best practices in C
- UBSan changes to arithmetics
- Lab – Handling integer overflow on the toolchain level in C/C++
- Best practices in C++
- Lab – Integer handling best practices in C++
- Files and streams

- Path traversal
- Path traversal-related examples
- Lab – Path traversal
- Path traversal best practices
- Lab – Path canonicalization
- Format string issues

- The problem with printf()
- Lab – Exploiting format string
- Time and state

- Race conditions

- File race condition

- Time of check to time of usage – TOCTTOU

- Lab – TOCTTOU

- Insecure temporary file

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

Skills You Will Learn:

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

Frequently Asked Questions

Q: What delivery options are available for Secure Coding in C and C++?

We offer multiple delivery formats:

- Live Instructor-Led Classroom Online (Virtual/Live Online)
- Traditional Instructor-Led Classroom Training (ILT)
- On-site delivery at your offices anywhere in United Kingdom
- Private dedicated courses customized for your team

Q: How many CPD hours does this course provide?

The 3-day Secure Coding in C and C++ course provides up to 19.5 CPD hours of structured learning. CPD certificates can be provided upon request.

Q: What is the duration of the Secure Coding in C and C++ training?

The training takes place over 3 day(s), with each day lasting approximately 24.00 hours including breaks for lunch and refreshments.

Q: Do you provide corporate training for Secure Coding in C and C++?

Yes, we provide corporate training, dedicated training, and closed classes for Secure Coding in C and C++. Training can take place anywhere in United Kingdom including London, Manchester, Birmingham, Edinburgh, or live online allowing teams from across United Kingdom or internationally to attend.

Q: What related terms do people search for?

Popular related searches include: Secure Coding in C and C++, Secure Coding, SECC-CCPP

Q: Why choose Nexus Human for Secure Coding in C and C++?

Nexus Human is recognized as one of the leading training providers. Our trainers have won multiple awards including:

- Small Firms Best Trainer Award
- National Training Partner of the Year (Ireland) - Multiple Years
- Global Top 30 Instructor Awards (2012, 2019, 2021)
- Tech Excellence Award Nominations
- Learning Performance Institute (LPI) External Training Provider Sponsor 2024

Q: Are there any discount codes available?

Yes! Use discount code **PENPAL5** when booking your Secure Coding in C and C++ training. Please note that only one discount code can be used per booking and cannot be combined with other special offers.

Nexus Human

Professional Training & Development

Email: info@nexushuman.com

Website: www.nexushuman.com

Phone: +353 1 889 8700 (Ireland) | +44 20 8142 8961 (UK)

© 2026 Nexus Human. All rights reserved. This brochure was generated on 19/10/2026.