Secure Coding in C and C++ for Automotive

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

Duration: 32.00 hours (4 days) **26.0 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.

To date vehicles become highly connected – not only between the internal components, but also to the outside worlds. Todays cars are already running millions of lines of source code, and this introduces a new set of risks to the industry that is historically concerned about safety. Even though some of the attacks are still theoretical, many of the standards already started introducing security considerations.

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.

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
- Using vulnerable components
- Security testing
- Wrap up

Who Should Attend

- Automotive Sector
- 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
- Learning about security specialties of the automotive sector
- Identify vulnerabilities and their consequences
- Learn the security best practices in C and C++
- Input validation approaches and principles
- Managing vulnerabilities in third party components
- Understanding security testing methodology and approaches
- Getting familiar with common security testing techniques and tools

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
- Categorization of bugs

- The Seven Pernicious Kingdoms
- SEI CERT Secure Coding Guidelines
- Coding standards in automotive

- ISO 26262

- MISRA C 2012

- MISRA C++ 2008

- AUTOSAR C++14 and the future

- CERT C Coding Standard

- CERT C++ Coding Standard

- Cyber security in the automotive sector

- Software security in the automotive world

- Threats and trends in automotive

- Self-driving car threats

- The CAN bus and security

- Case study – controlling the CAN bus through the infotainment system

Buffer overflow

- Assembly basics and calling conventions

- ARM assembly essentials

- Registers and addressing

- Basic ARM64 instructions

- ARM calling conventions

- The calling convention

- The stack frame

- Calling convention implementation on ARM64

- Stacked function calls

- Memory management vulnerabilities

- Memory management and security

- Vulnerabilities in the real world

- Mapping to MISRA
- 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++

DAY 2

Buffer overflow

- Some typical mistakes leading to BOF

- 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

Memory management hardening

- Mapping to MISRA
- 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 epilogue
- 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
- Heap spraying
- 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)

- Lab – ROP demonstration
- Protection against ROP

Common software security weaknesses

- Security features

- Authentication

- Authentication basics
- Multi-factor authentication
- Authentication weaknesses – spoofing
- Case study – Hacking the Mitsubishi Outlander PHEV hybrid
- 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
- Authorization

- Access control basics
- File system access control

- Improper file system access control
- Ownership
- chroot jail
- Using umask()
- Linux filesystem
- LDAP

DAY 3

Common software security weaknesses

- Input validation

- Input validation principles

- Blacklists and whitelists
- Data validation techniques
- What to validate – the attack surface
- Attack surface in automotive
- Where to validate – defense in depth
- How to validate – validation vs transformations
- Output sanitization

- Encoding challenges
- Validation with regex
- Mapping to MISRA
- 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
- Case study – Command injection in Jeep Cherokee
- Process control – library injection

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

- Representing signed numbers
- Integer visualization
- The MISRA essential type model
- Integer promotion
- Composite expressions and type conversion
- Integer overflow
- Lab – Integer overflow
- Signed / unsigned confusion

- 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
- Lab – Handling integer overflow on the toolchain level in C/C++
- Best practices in C++
- Lab – Integer handling best practices in C++
- Other numeric problems

- Security issues with bitfields
- 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
- Lab – Path canonicalization

- Format string issues
- The problem with printf()
- Lab – Exploiting format string

DAY 4

Common software security weaknesses

- Time and state

- Mapping to MISRA
- Race conditions

- Race condition in object data members
- File race condition

- Time of check to time of usage – TOCTTOU
- Lab – TOCTTOU
- Insecure temporary file
- Potential race conditions in C/C++

- Race condition in signal handling
- Forking
- Bit-field access
- Errors

- Error and exception handling principles
- Mapping to MISRA
- Error handling

- Returning a misleading status code
- Error handling in C
- Error handling in C++
- Using std::optional safely

- Information exposure through error reporting
- Exception handling

- In the catch block. And now what?
- Empty catch block
- Exception handling in C++
- Lab – Exception handling mess
- Code quality

- 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[]
- Control flow

- Incorrect block delimitation
- Dead code
- Leftover debug code
- Backdoors, dev functions and other undocumented functions
- Using if-then-else and switch defensively
- Signal handling

- Signal handlers

- Best practices
- Object oriented programming pitfalls

- Inheritance and object slicing
- Implementing the copy operator
- The copy operator and mutability
- Mutability

- Mutable predicate function objects
- Lab – Mutable predicate function object
- Memory and pointers

- Memory and pointer issues
- Pointer handling pitfalls
- Alignment
- Null pointers

- NULL dereference
- NULL dereference in pointer-to-member operators
- 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
- Incorrect pointer arithmetics
- File I/O

- Working with file descriptors, structures and objects

- File reading and writing

- File access functions and methods

Using vulnerable components

- Assessing the environment

- Hardening

- Vulnerability management

- Patch management

- Bug bounty programs

- Vulnerability databases

- Vulnerability rating – CVSS

- Lab – Finding vulnerabilities in third-party components

- 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

- Security aspects of code review

- Static Application Security Testing (SAST)

- Lab – Using static analysis tools

- Dynamic analysis

- Security testing at runtime

- Penetration testing

- Stress testing

- Dynamic analysis tools

- Dynamic Application Security Testing (DAST)

- Fuzzing

- Fuzzing techniques

- Fuzzing – Observing the process

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

- Links to automotive coding standards

Skills You Will Learn:

- Getting familiar with essential cyber security concepts
- Learning about security specialties of the automotive sector
- Identify vulnerabilities and their consequences
- Learn the security best practices in C and C++
- Input validation approaches and principles
- Managing vulnerabilities in third party components
- Understanding security testing methodology and approaches
- Getting familiar with common security testing techniques and tools

Frequently Asked Questions

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

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 4-day Secure Coding in C and C++ for Automotive course provides up to 26.0 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++ for Automotive training?

The training takes place over 4 day(s), with each day lasting approximately 32.00 hours including breaks for lunch and refreshments.

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

Yes, we provide corporate training, dedicated training, and closed classes for Secure Coding in C and C++ for Automotive. 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++ for Automotive, Secure Coding, SECC-CCPPA

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

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++ for Automotive 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 14/11/2026.