Secure coding in C and C++ (x64)

CELCpx64 | 1 year subscription | e-Learning | Online VM

The x64 variant of the comprehensive C and C++ e-learning course provides a structured approach to understanding and addressing various aspects of secure coding in C and C++. After a primer on x64 assembly and memory operations, the curriculum addresses critical security issues related to memory management. The effects of various toolchain-level protection techniques you can apply to defend against such vulnerabilities (such as SSP, ASLR and NX) are also explained.

The secure coding modules are aligned to common software security weaknesses in all major categories: input validation, improper use of security features, code quality, error handling, time and state, and denial of service. The course also provides practical skills related to cryptography that every developer should understand (such as hashing, encryption, digital signatures, PKI), showing how to use these in OpenSSL. Finally, we give an overview of security testing tools and how to use them to find vulnerabilities in your code.

Through hands-on labs and real-world case studies, you will explore best practices to get the appropriate skills and master the secure coding mindset.

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

So that nothing unexpected happens in your code.

Nothing.

Note: This course content is available as an e-learning subscription. We reserve a period of 3 months to digest the foundational material, after which we activate shorter learning units on a monthly basis. This gives secure coding efforts an initial boost, and builds up sustained readiness over time. These learning units are indicated in red in the table of contents below.

Cyber security skills and drills

Foundational material

43 LABS

11 CASE STUDIES

Monthly learning units

2-3 LABS

CASE STUDY

Audience

C/C++ developers

Outline

- Cyber security basics
- Assembly basics and calling conventions
- Memory management vulnerabilities
- Memory management hardening
- Improper use of security features
- Code quality
- Input validation
- Wrap up

Preparedness

General C/C++ development

Standards and references

SEI CERT, CWE and Fortify Taxonomy

What you'll have learned

- 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

Table of contents

Stack overflow - shellcode

- Exploitation – Arbitrary code execution
- Injecting shellcode

Lab – Code injection, exploitation with shellcode

Buffer overflow on the heap

- Unsafe unlinking

Case study – Heartbleed

Advanced heap exploit (Unit 4)

- An example heap exploitation technique: "House of Einherjar"

Lab – Heap overflow exploitation

Array indexing (Unit 4)

- Range error – improper validation of array index

Lab – Array indexing

Containers (Unit 4)

- Security issues with containers and unused elements
- Associative containers
- Iterators

Pointer manipulation

- Modification of jump tables
- Overwriting function pointers

Unsafe functions 1

- Unsafe functions
- Dealing with unsafe functions

Lab – Fixing buffer overflow

Unsafe functions 2

- What's the problem with asctime()?

Lab – The problem with asctime()

Unsafe functions in C++

- Using std::string in C++

String termination

- Unterminated strings
- readlink() and string termination
- Manipulating C-style strings in C++
- Malicious string termination

Lab – String termination confusion

Some other typical mistakes

- String length calculation mistakes
- Off-by-one errors
- Allocating nothing

Memory management hardening

Compiler options

- Securing the toolchain in C and C++
- Compiler warnings and security

Fortify

- Using FORTIFY_SOURCE

Lab – Effects of FORTIFY

AddressSanitizer

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

Lab – Using AddressSanitizer

Further toolchain-related protections

- RELRO protection against GOT hijacking
- Heap overflow protection

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

- Bypassing stack smashing protection

Runtime instrumentation

Address Space Layout Randomization

- Address Space Layout Randomization (ASLR)
- ASLR on various platforms

Lab – Effects of ASLR

- Circumventing ASLR – NOP sleds

- Circumventing ASLR – memory leakage
- 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

- Whatever Oriented Programming
- Protection against ROP

Improper use of security features

Introduction

- Security features

Authentication

- Authentication basics
- Authentication weaknesses
- User interface best practices

Password management

- Storing account passwords
- Password in transit

Lab – Is just hashing passwords enough?

Password storage

- Dictionary attacks and brute forcing
- Salting
- Adaptive hash functions for password storage

Password policy

- NIST authenticator requirements for memorized secrets

Password storage - a case study

Case study – The Ashley Madison data breach

The dictionary attack

The ultimate crack

Exploitation and the lessons learned

Additional password management challenges

- Password database migration
- (Mis)handling NULL passwords

Outbound password management (Unit 1)

- Hard coded passwords
- Best practices

Lab – Hardcoded password

Case study – Hard-coded Telnet password in TOTOLINK T8

Protecting secrets in memory (Unit 1)

- Challenges in protecting memory
- Heap inspection
- Compiler optimization challenges

Lab – Zeroization challenges

- Sensitive info in non-locked memory

› Code quality

Introduction

- Code quality and security

Data

- Type mismatch

Lab – Type mismatch

- Declaration and allocation issues in C
- Declaration and allocation issues in C++
- Unchecked Return Value

Case study – #iamroot hash migration bug

- Omitted return value
- Returning unmodifiable pointer

Data - initialization

- Uninitialized variable
- Constructors and destructors
- Initialization of static objects

Lab – Initialization cycles

Data - release

- Unreleased resource

- Unreleased resource – Synchronization
- Unreleased resource – Files
- Array disposal in C++

Lab – Mixing delete and delete[]

Control flow (Unit 9)

- Incorrect block delimitation
- Dead code
- Leftover debug code
- Backdoors, dev functions and other undocumented functions
- Using if-then-else and switch defensively
- Returning from a [[noreturn]] function
- Signal handlers
- Signal handling best practices

Language elements 1 (Unit 9)

- Language elements
- Function arguments
- Undefined and unspecified behavior
- Using dangerous language elements
- Using obsolete language elements

Language elements 2 (Unit 9)

- Portability flaw
- Preprocessor – Using macros
- Multiple binds to the same port

Object Oriented Programming (Unit 9)

- 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

Memory and pointers

- Memory and pointer issues
- Pointer handling pitfalls
- Alignment

Null pointers

- NULL dereference
- NULL dereference in pointer-to-member operators

Freeing

- Use after free

Lab – Use after free

Lab – Runtime instrumentation

- Double free
- Freeing a stack pointer
- Memory leak
- Resource Acquisition Is Initialization (RAII)

Smart pointers

- Smart pointer best practices
- Smart pointers and security
- Incorrect pointer arithmetic

› Input validation

Input validation principles 1

- Input validation principles
- Denylists and allowlists
- Data validation techniques

Input validation principles 2

- What to validate – the attack surface
- Where to validate – defense in depth
- When to validate – validation vs transformations

Input validation principles 3

- Output sanitization
- Encoding challenges
- Unicode challenges

Injection

- Injection principles
- Injection attacks
- Code injection

Command injection

- OS command injection

Lab – Command injection

Command injection best practices

- OS command injection best practices
- Avoiding command injection with the right APIs

Lab – Command injection best practices

Shellshock

Case study – Shellshock

Lab - Shellshock

Integer handling problems

- Representing signed numbers
- Integer visualization
- Integer promotion
- Integer overflow

Lab – Integer overflow

Integer handling problems 2

- Signed / unsigned confusion

Case study – The Stockholm Stock Exchange

Lab – Signed / unsigned confusion

- Integer truncation

Lab – Integer truncation

Case study – WannaCry

Integer best practices

- Upcasting
- Precondition testing
- Postcondition testing
- Using big integer libraries

Integer best practices in C/C++

- Best practices in C
- Best practices in C++

Lab – Integer handling best practices in C++

Other numeric problems

- Division by zero
- Working with floating-point numbers

Path traversal and file validation (Unit 8)

- Path traversal

Lab – Path traversal

- Path traversal-related examples
- Link and shortcut following
- Virtual resources
- Path traversal best practices

Lab – Path canonicalization

Format strings (Unit 8)

- Format string issues
- The problem with printf()

Lab – Exploiting format string

Time and state (Unit 6)

Introduction

- Time and state
- Thread management best practices in C/C++

Race conditions

- Race condition in object data members

Case study – State confusion in VxWorks IPNet stack

- Time of check to time of usage – TOCTTOU
- TOCTTOU attacks in practice

Lab - TOCTTOU

- Insecure temporary file

Race conditions in C/C++

- Potential race conditions in C and C++
- Race condition in signal handling
- Forking
- Bit-field access
- Using ThreadSanitizer (TSan)

Locking and deadlocks

- Mutual exclusion and locking
- Deadlocks
- Mutual exclusion and locking in C
- Mutual exclusion and locking in C++

Synchronization and thread safety

- Synchronization and thread safety in C/C++

› Error handling (Unit 5)

Principles

- Error and exception handling principles
- Information exposure through error reporting

Error handling

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

Exception handling

- In the catch block. And now what?
- Empty catch block
- Exception handling in C++

Lab – Exception handling mess

› Denial of service (Unit 2)

- Flooding
- Resource exhaustion

Sustained client engagement

- Infinite loop
- Denial of service problems in C/C++

Case study – DoS against Tesla GUI via malicious web page

- Economic Denial of Sustainability (EDoS)

Amplification

- Some amplification examples

Algorithmic complexity issues

- Regular expression denial of service (ReDoS)

Lab – ReDoS

- Dealing with ReDoS
- Hash table collision
- How do hash tables work?
- Hash collision against hash tables

› Cryptography (Unit 7)

Cryptography for developers

- Cryptography basics
- OpenSSL in brief

PRNG

- Random number generation
- Pseudo random number generators (PRNGs)
- Cryptographically secure PRNGs
- Using virtual random streams

Case study – Equifax credit account freeze

PRNG in C/C++

- Weak and strong PRNGs
- Using random numbers in OpenSSL

Lab – Using random numbers in OpenSSL

Hashing

- Hashing basics

Case study - Shattered

- Common hashing mistakes
- Hashing in OpenSSL

Lab – Hashing in OpenSSL

- Hash algorithms for password storage
- Password storage algorithms and considerations
- Best practices when using password hashing algorithms

Encryption

- Confidentiality protection
- Symmetric encryption
- Block ciphers
- Modes of operation
- Modes of operation and IV – best practices

Encryption in C/C++

- Symmetric encryption in OpenSSL

Lab – Symmetric encryption in OpenSSL

Asymmetric encryption

- The RSA algorithm
- Using RSA – best practices

- RSA in OpenSSL
- Combining symmetric and asymmetric algorithms

Integrity protection and MAC

- Integrity protection
- Message Authentication Code (MAC)
- Calculating MAC in OpenSSL

Lab – Calculating MAC in OpenSSL

Digital signatures

- Digital signature
- Digital signature with RSA
- ECC basics
- Digital signature with ECC

Lab – Digital signature with ECDSA in OpenSSL

› Using security testing tools (Unit 3)

Static application security testing (SAST)

- Security aspects of code review
- The OWASP Code Review methodology
- Static Application Security Testing (SAST)

Lab – Using static analysis tools

Dynamic application security testing (DAST)

- Security testing at runtime
- Penetration testing
- Memory inspection and analysis

Lab – Dumping process memory

- Stress testing

Dynamic analysis techniques and tools

- Dynamic Application Security Testing (DAST)
- IAST and DevSecOps
- Fuzzing
- Fuzzing techniques
- Fuzzing – Observing the process
- American Fuzzy Lop (AFL)

Lab – Fuzzing

› Wrap up

Software security principles

- Principles of robust programming by Matt Bishop

Sources and further readings

- Software security sources and further reading
- C and C++ resources