Secure coding in C and C++ (ARM)

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

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 -2^31? 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.

Cyber security skills and drills

Base training

30 LABS

6 CASE STUDIES

Monthly feed for a year

2-3 LABS

CASE STUDY

Audience

C/C++ developers

Preparedness

General C/C++ development

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

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

Pointer manipulation

- Modification of jump tables
- GOT and PLT
- 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
- ARM-specific ROP protection techniques

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?

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

Lab – Password audit with John the Ripper

Password policy

- NIST authenticator requirements for memorized secrets
- Password hardening
- Using passphrases

Password handling challenges

- Password change
- Password recovery issues
- Password recovery best practices

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

› Code quality

Introduction

- Code quality and security

Data

- Type mismatch

Lab – Type mismatch

- Declaration and allocation issues in C

Data - C++

- Declaration and allocation issues in C++

Data - return values

- 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[]

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

Smart pointers

- Smart pointers and RAII
- Smart pointer challenges
- Incorrect pointer arithmetics

› 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
- Validation with regex

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

Process control

- Library injection

Lab – Library hijacking

Integer handling problems

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

Lab – Integer overflow

Signed-unsigned confusion

- Signed / unsigned confusion

Lab – Signed / unsigned confusion

Case study – The Stockholm Stock Exchange

Integer truncation

Lab – Integer truncation

Case study – WannaCry

Integer best practices 1

- Upcasting
- Precondition testing
- Postcondition testing

Integer best practices 2

- Using big integer libraries
- Best practices in C

Lab – Handling integer overflow on the toolchain level in C and C++

Integer 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

Lab – Path traversal

- Path traversal-related examples

Other file validation issues

- Link and shortcut following
- Virtual resources

File validation best practices

- Path traversal best practices

Lab – Path canonicalization

Format strings

- Format string issues
- The problem with printf()

Lab – Exploiting format string

› Wrap up

Software security principles

- Principles of robust programming by Matt Bishop
- Secure design principles of Saltzer and Schroeder
- Some more principles

Sources and further readings

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