Secure coding in C and C++ - ARM64

CYDCp3d_ARM64 | 3 days | On-site or online | Hands-on

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.

This is the ARM64 version of the Secure coding in C and C++ course. All relevant hands-on labs will be done in a virtual ARM64 environment.

Skills and drills

26 LABS

7 CASE STUDIES

Audience

C/C++ developers

Group size

12 participants

Preparedness

General C/C++ development

Outline

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

Standards and references

SEI CERT, CWE and Fortify Taxonomy

What you'll have learned

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

Table of contents

Day 1

› Cyber security basics

What is security?

Threat and risk

Cyber security threat types – the CIA triad

Cyber security threat types – the STRIDE model

Consequences of insecure software

› Memory management vulnerabilities

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

Buffer overflow

- Memory management and security
- 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

  Case study – Stack BOF in DrayTek Vigor routers
- Buffer overflow on the heap
  - Unsafe unlinking

Case study – Heartbleed

Best practices and some typical mistakes

- Unsafe functions
- Dealing with unsafe functions

Lab – Fixing buffer overflow

- Using std::string in C++
- Unterminated strings
- Manipulating C-style strings in C++
- Malicious string termination

Lab – String termination confusion

- String length calculation mistakes

Day 2

› Memory management hardening

Securing the toolchain

- Securing the toolchain in C and C++
- AddressSanitizer (ASan)
  - Using AddressSanitizer (ASan)

  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

Runtime protections

- Runtime instrumentation
- 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
  - ARM-specific ROP protection techniques

› Common software security weaknesses

Security features

- Authentication
- 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
  - 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
    - *Case study – Microsoft secret key theft via dump files*

Code quality

- Code quality and security
- 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
  - 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
  - Smart pointers

Day 3

› Common software security weaknesses

Input validation

- Input validation principles
- Denylists and allowlists
- What to validate – the attack surface
- Where to validate – defense in depth
- When to validate – validation vs transformations
- Output sanitization
- Encoding challenges
- Unicode challenges
- Validation with regex
- Regular expression denial of service (ReDoS)

Lab – ReDoS

- Dealing with ReDoS
- Injection
  - 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 AVTECH IP cameras*
  - Process control
    - Library injection

Lab – Library hijacking

- Library injection best practices

- 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
    - Best practices in C++
    - *Lab – Integer handling best practices in C++*
- Files and streams
  - Path traversal
  - *Lab – Path traversal*
  - Path traversal best practices
  - *Lab – Path canonicalization*

› Wrap up

Secure coding principles

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

And now what?

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