Secure coding in C and C++ masterclass - ARM

CYDCp5d_ARM | 5 days | On-site or online | Hands-on

Your application written in C and C++ is tested functionally, 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.

Testing for security needs a remarkable software security expertise and 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.

A special focus is given to finding all discussed issues during testing, and an overview is provided on security testing methodology, techniques and tools.

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

So that nothing unexpected happens.

Nothing.

Skills and drills

37 LABS

10 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
- Cryptography for developers
- Security testing
- 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++
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in C and C++
- Input validation approaches and principles
- Understanding security testing methodology and approaches
- Getting familiar with security testing techniques and tools

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
- 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

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

- Unsafe unlinking

Case study – Heartbleed

- Pointer manipulation
  - Modification of jump tables
  - GOT and PLT
  - Overwriting function pointers

Best practices and some typical mistakes

- Unsafe functions
- Dealing with unsafe functions

Lab – Fixing buffer overflow

- Using std::string in C++

Day 2

› Memory management vulnerabilities

Unterminated strings

Manipulating C-style strings in C++

Malicious string termination

Lab – String termination confusion

String length calculation mistakes

Off-by-one errors

Allocating nothing

Testing for typical mistakes

› Memory management hardening

Securing the toolchain

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

Lab – Using AddressSanitizer

- 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
- Heap cookies and guard pages

Runtime protections

- Runtime instrumentation
- 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

    - Protection against ROP
    - ARM-specific ROP protection techniques

› Common software security weaknesses

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

  - Unreleased resource
    - 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
  - Testing for null pointers
- Pointer usage in C and C++
  - Use after free
  - *Lab – Use after free*
  - *Lab – Runtime instrumentation*
  - Double free
  - Smart pointers
- Testing for memory and pointer issues

Day 3

› Common software security weaknesses

Time and state

- Race conditions
  - File race condition
    - Time of check to time of usage – TOCTTOU
    - TOCTTOU attacks in practice
    - *Lab - TOCTTOU*
    - Insecure temporary file
- Testing for time and state problems

› Cryptography for developers

Cryptography basics

OpenSSL in brief

Elementary algorithms

- Hashing
  - Hashing basics
  - Common hashing mistakes
  - Hashing in OpenSSL
  - *Lab – Hashing in OpenSSL*
- Random number generation
  - Pseudo random number generators (PRNGs)
  - Cryptographically secure PRNGs
  - Using virtual random streams

- Weak and strong PRNGs
- Using random numbers in OpenSSL

Lab – Using random numbers in OpenSSL

Case study – Equifax credit account freeze

Confidentiality protection

- Symmetric encryption
  - Block ciphers
  - Modes of operation
  - Modes of operation and IV – best practices
  - Symmetric encryption in OpenSSL

Lab – Symmetric encryption in OpenSSL

- Asymmetric encryption
  - The RSA algorithm
    - Using RSA – best practices
    - RSA in OpenSSL

Case study – RSA attacks: Bleichenbacher, ROBOT, and Marvin

- Combining symmetric and asymmetric algorithms
- Key exchange and agreement
  - Key exchange
  - Diffie-Hellman key agreement algorithm
  - Key exchange pitfalls and best practices

Integrity protection

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

Lab – Calculating MAC in OpenSSL

- Digital signature
  - Digital signature with RSA
  - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
  - Digital signature in OpenSSL
  - ECDSA in OpenSSL

Lab – Digital signature with ECDSA in OpenSSL

Public Key Infrastructure (PKI)

- Key management challenges
- Certificates
  - Certificates and PKI
  - Chain of trust
  - X.509 certificates

- PKI actors and procedures
- Certificate revocation
- Testing for certificate and PKI weaknesses

Transport security

- The TLS protocol
  - TLS basics
  - TLS features (changes in v1.3)
  - The handshake in a nutshell (v1.3)
  - An example TLS 1.3 handshake
  - TLS best practices
  - Testing transport security

Day 4

› 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
- 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 AVTECH IP cameras*
    - Testing for command injection
- 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++*
  - Testing for numeric problems
- Files and streams
  - Path traversal
  - *Lab – Path traversal*
  - Path traversal-related examples
  - *Case study – File spoofing in WinRAR*
  - Virtual resources
  - Path traversal best practices
  - *Lab – Path canonicalization*
  - Testing for path traversal

Day 5

› Security testing

Security testing vs functional testing

Manual and automated methods

Black box, white box, and hybrid testing

Security testing methodology

- Security testing – goals and methodologies
- Overview of security testing processes
- Identifying and rating assets
  - Preparation and scoping
  - Identifying assets
  - Identifying the attack surface
  - Assigning security requirements
  - *Lab – Identifying and rating assets*
- Threat modeling
  - SDL threat modeling
  - Mapping STRIDE to DFD
  - DFD example
  - Attack trees
  - Attack tree example
  - *Lab – Crafting an attack tree*
  - Misuse cases
  - Misuse case examples
  - Risk analysis
  - *Lab – Risk analysis*
- Accomplishing the tests
- Reporting, recommendations, and review

Security testing techniques and tools

- Code analysis
  - Static Application Security Testing (SAST)
  - *Lab – Using static analysis tools*
- Dynamic analysis
  - Security testing at runtime
  - Penetration testing
  - Memory inspection and analysis
  - *Lab – Dumping process memory*

- Stress testing
- Dynamic Application Security Testing (DAST)
- Fuzzing
- Fuzzing techniques
- Fuzzing – Observing the process
- American Fuzzy Lop (AFL)

› Common software security weaknesses

Errors

- Error and exception handling principles
- Error handling
  - Returning a misleading status code
  - Error handling in C
  - Error handling in C++
  - Using std::optional safely
  - Information exposure through error reporting
    - *Case study – Information leakage via errors in Apache Superset*
- Exception handling
  - In the catch block. And now what?
  - Empty catch block
  - Exception handling in C++
  - *Lab – Exception handling mess*
- Testing for error and exception handling problems

Denial of service

- Flooding
- Resource exhaustion
- Sustained client engagement
- Denial of service problems in C/C++
- Infinite loop
- 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

› 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
- Security testing resources