Security testing C and C++ applications

CYDCpTst3d | 3 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.

Cyber security skills and drills

Audience

C/C++ developers and testers

Group size

12 participants

Outline

- Cyber security basics
- Memory management vulnerabilities
- Memory management hardening
- Security testing
- Common software security weaknesses
- Using vulnerable components
- Wrap up

Preparedness

General C/C++ development, testing and QA

25 labs

6 case studies

What you'll have learned

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

Table of contents

Day 1

- 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++
- 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
- Testing for typical mistakes

Day 2

› Memory management hardening

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

› Security testing

Security testing vs functional testing

Manual and automated methods

Security testing methodology

- Security testing – goals and methodologies
- Overview of security testing processes
- Identifying and rating assets
  - Preparation
  - 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*
- Security testing approaches
  - Reporting, recommendations, and review

› Common software security weaknesses

Security features

- Authentication
  - Authentication basics
  - Multi-factor authentication
  - Authentication weaknesses - spoofing
  - *Case study – PayPal 2FA bypass*
- 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
- Testing for password management issues

Time and state

- Race conditions
  - File race condition
    - Time of check to time of usage – TOCTTOU
    - *Lab - TOCTTOU*
    - Insecure temporary file

› Using vulnerable components

Assessing the environment

Hardening

Vulnerability management

- Patch management
- Vulnerability management
- Vulnerability databases

Lab – Finding vulnerabilities in third-party components

Day 3

› Common software security weaknesses

Input validation

- Input validation principles
  - Blacklists and whitelists
  - Data validation techniques
  - What to validate – the attack surface
  - Where to validate – defense in depth
  - How to validate – validation vs transformations

- Output sanitization
- Encoding challenges
- Validation with regex

- 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*
    - Testing for command injection
    - Process control – library injection
    - DLL hijacking
    - *Lab – DLL hijacking*
- 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
    - Using big integer libraries
    - Best practices in C
    - UBSan changes to arithmetics
    - *Lab – Handling integer overflow on the toolchain level in C/C++*
    - Best practices in C++
    - *Lab – Integer handling best practices in C++*
  - Testing for numeric problems
- Files and streams

- Path traversal
- Path traversal-related examples
- *Lab – Path traversal*
- Path traversal best practices
- *Lab – Path canonicalization*
- Testing for path traversal

› Security testing

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
- Security testing resources