

C and C++ SEI CERT based secure coding (ARM)

CL-CPAS | Virtual classroom | 3 days

Variant: ARM

Audience: C and C++ developers, software architects and testers

Preparedness: General C/C++ development

Exercises: Hands-on

This course teaches you how to write code according to the SEI CERT Secure Coding standards. It starts with a brief introduction to the SEI. Then, we dive into how an ARM processor is built and how functions work on this architecture. Understanding these basics is the first step in developing a security-focused mindset. Later, we explore how an attacker can exploit simple buffer overflows and utilize more complex attack techniques. The course covers mitigation techniques used during coding, compilation, and runtime.

When coding in C/C++, buffer overflow isn't the only thing you need to worry about. The course also highlights the most common mistakes and defense techniques. After this training, participants will apply the learned mitigation techniques in their daily work to prevent them from happening.

All of this is complemented with hands-on exercises using exploitable applications and a debugger. Step-by-step instructions, challenge-based exercises, and visual slides are blended seamlessly to create the best learning experience.

Participants usually attend a master's course to refresh their knowledge and dive deeper into secure coding and some advanced topics such as cryptography.

Outline:

- IT security and secure coding
- ARM machine code, memory layout and stack operations
- Buffer overflow
- Summary of Buffer Overflow protections
- SEI CERT examples
- XML security
- Common coding errors and vulnerabilities
- Denial of service
- Principles of security and secure coding
- Knowledge sources

Participants attending this course will:

Be able to write code according to the SEI CERT Coding Standards
Understand basic concepts of security, IT security and secure coding
Realize the severe consequences of unsecure buffer handling
Understand the architectural protection techniques and their weaknesses
Learn about XML security
Learn about typical coding mistakes and how to avoid them
Be informed about recent vulnerabilities in various platforms, frameworks and libraries
Learn about denial of service attacks and protections
Get sources and further readings on secure coding practices

Related courses:

- CL-CMI - C and C++ security master course (x86) (Onsite / Virtual classroom, 5 days)
- CL-CMA - C and C++ security master course (ARM) (Onsite / Virtual classroom, 5 days)
- CL-CTS - Security testing native code (Onsite / Virtual classroom, 3 days)

Detailed table of contents

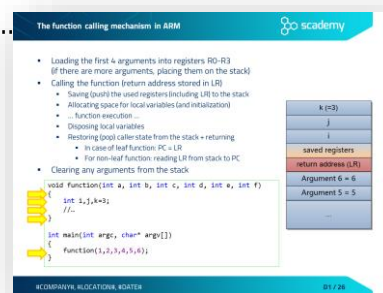
Day 1

IT security and secure coding

- Nature of security
- What is risk?
- IT security vs. secure coding
- From vulnerabilities to botnets and cybercrime
 - Nature of security flaws
 - From an infected computer to targeted attacks
- Classification of security flaws
 - Landwehr's taxonomy
 - The Seven Pernicious Kingdoms
- The SEI CERT Coding Standard
 - SEI CERT Coding Standards
 - The SEI CERT C Coding Standard
 - SEI CERT – Rules and Recommendations
 - SEI CERT in this course

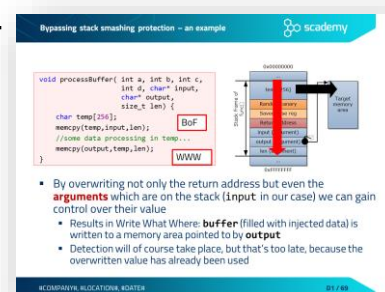
ARM machine code, memory layout and stack operations

- ARM Processors – main registers
- ARM Processors – most important instructions
- ARM Processors – flags and condition fields
- ARM Processors – control instructions
- ARM Processors – stack handling instructions
- Understanding complex ARM instructions
- The function calling mechanism in ARM
- The local variables and the stack frame
- Function calls – prologue and epilogue of a function (ARM)
- Stack frame of nested calls
- Stack frame of recursive functions



Buffer overflow

- Stack overflow
 - Buffer overflow on the stack
 - Overwriting the return address
 - Breakout Session
 - Exercises – introduction
 - Exercise - Tools
 - Exercises - QEMU
 - Exercise BOFIntro
 - Exercise BOFShellcode
 - End of Breakout Session
- Protection against stack overflow
 - Specific protection methods
 - Protection methods at different layers
 - The protection matrix of software security
 - Stack overflow – Prevention (during development)
 - The protection matrix of software security
 - Stack overflow – Detection (during execution)
 - Fortify compiler option (FORTIFY_SOURCE)
 - Bypassing stack smashing protection – an example.....
 - Overwriting arguments – Mitigation
 - The protection matrix of software security
- Address Space Layout Randomization (ASLR)
 - Randomization with ASLR
 - Software ASLR
 - Practical weaknesses and limitations to ASLR
 - Circumventing ASLR: NOP sledding
- Non executable memory areas – the NX bit
 - Access control on memory segments
 - The Never eXecute (NX) bit
 - Breakout Session
 - Exercise BOFShellcode – Using the Fortify compiler option
 - Exercise BOFShellcode – Stack smashing protection
 - Effects of stack smashing protection – prologue
 - Effects of stack smashing protection – epilogue
 - Exercise BOFShellcodeAddress Space Layout Randomization
 - Using ASLR
 - Exercise BOFShellcodeEnforcing NX memory segments



- Exercise BOFShellcodeTurning on all compiler protections
- Exercise BOFShellcode – Turning on all compiler protections
- End of Breakout Session

Summary of Buffer Overflow protections

- Summary of compiler options
- The protection matrix of software security

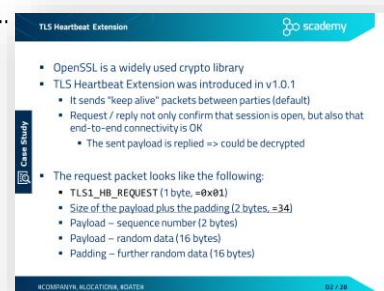
Day 2

Buffer overflow

- Return oriented programming (ROP)
 - Circumventing memory execution protection
 - Return-to-libc attack in ARM
 - ROP gadget - Register fill with constants
 - ROP gadget – Memory write
 - Combining the ROP gadgets
 - Real ROP attack scenarios
 - ROP mitigation
 - Mitigation techniques of ROP attack
- Heap overflow
 - Memory allocation managed by a doubly-linked list
 - Buffer overflow on the heap
 - Steps of freeing and joining memory blocks
 - Freeing allocated memory blocks
 - Case study – Heartbleed
 - TLS Heartbeat Extension.....
 - Heartbleed – information leakage in OpenSSL
 - Heartbleed – fix in v1.0.1g
 - Protection against heap overflow

SEI CERT examples

- Undefined and unspecified behavior
 - Undefined behavior
 - Unspecified behavior



- Smart Pointers
 - RAII
 - Exclusive ownership
 - Let's write our own RAII wrapper
 - Spot the bug!
 - Smart pointers – exclusive ownership
 - Spot the bug!
 - `std::make_unique<T>(...)`
 - Shared ownership
 - `std::shared_ptr`'s control block
 - `std::shared_ptr` internals
 - `std::shared_ptr`
 - What is worse than leaking the memory?
 - Spot the bug!
 - Fixed
 - Spot the bug!
 - Fixed
 - Spot the bug!
 - Fixed
 - Related smart pointers
 - Summary
- Object lifetime management
 - Manually managing object lifetime
 - Spot the bug – Object lifetime management
 - Manual Lifetime management
 - Summary
- Container iteration
 - Vector basics
 - Spot the bug – Container iteration
 - Container Iteration
 - Summary
- Random Number Generators
 - Random Number Generator basics
 - How can you generate random numbers?
 - Generating random numbers
 - What happens if the random number is weak or not strong enough?
 - PlayStation 3
 - Java nonce collision
 - Microsoft Windows XP random number generator

- Pseudorandom Number Generators (PRNG)
 - PRNGs in a nutshell
 - Random number engines
 - Spot the bug – PRNGs 1
 - Spot the bug – PRNGs 2
 - Lottery Application
 - Lottery Application 1
 - Lottery Application 2
 - Summary

XML security

- XML injection
 - Injection principles
 - Exercise – XML injection
 - Protection through sanitization and XML validation
 - XML parsing in C++
- Abusing XML Entity
 - XML Entity introduction
 - Exercise – XML bomb
 - XML bomb
 - XML external entity attack (XXE) – resource inclusion
 - Exercise – XXE attack
 - Preventing entity-related attacks
 - Case study – XXE in Google Toolbar
 - End of Breakout Session

Common coding errors and vulnerabilities

- Improper error and exception handling
 - Typical problems with error and exception handling
 - Empty catch block
 - Overly broad catch
 - Exercise ErrorHandling – spot the bug!
 - Exercise – Error handling
- Code quality problems
 - Dangers arising from poor code quality
 - Poor code quality – spot the bug!
 - Unreleased resources
 - Type mismatch – Spot the bug!
 - Exercise TypeMismatch



- Memory allocation problems
 - Smart pointers
 - Zero length allocation
 - Double free
 - Mixing delete and delete[]
- Use after free
 - Use after free – Instance of a class
 - Spot the bug
 - Use after free – Dangling pointers
 - Case study - WannaCry

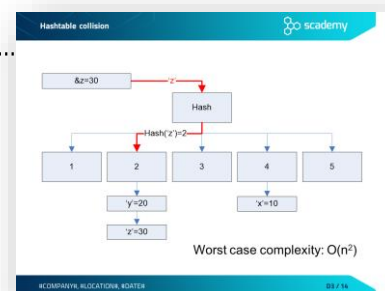
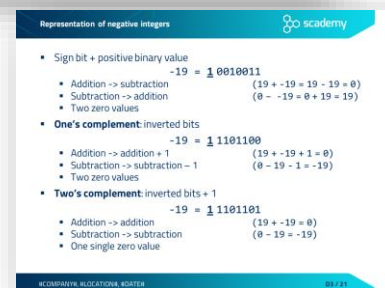
Day 3

Denial of service

- DoS introduction
- Asymmetric DoS
- Regular expression DoS (ReDoS)
 - Exercise ReDoS
 - Case study – ReDos in Stack Exchange
- Hashtable collision attack
 - Using hashtables to store data
 - Hashtable collision.....

Common coding errors and vulnerabilities

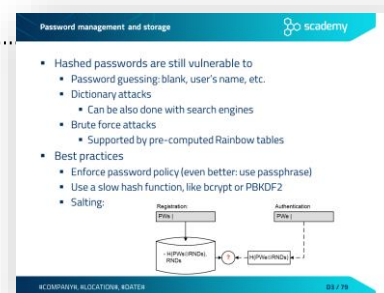
- Input validation
 - Input validation concepts
 - Integer problems
 - Representation of negative integers.....
 - Integer ranges
 - Integer overflow
 - Integer problems in C/C++
 - The integer promotion rule in C/C++
 - Arithmetic overflow – spot the bug!
 - Breakout Session
 - Exercise IntOverflow
 - What is the value of abs(INT_MIN)?
 - Signedness bug – spot the bug!
 - Integer truncation – spot the bug!
 - Integer problem – best practices
 - Case study – Android Stagefright

Representation of negative integers

- Sign bit + positive binary value**
 $-19 = 1\ 0010011$
 • Addition \rightarrow subtraction: $(19 + -19 = 19 - 19 = 0)$
 • Subtraction \rightarrow addition: $(0 - -19 = 0 + 19 = 19)$
 • Two zero values
- One's complement: inverted bits**
 $-19 = 1\ 1101100$
 • Addition \rightarrow addition + 1: $(19 + -19 + 1 = 0)$
 • Subtraction \rightarrow subtraction - 1: $(0 - -19 - 1 = -19)$
 • Two zero values
- Two's complement: inverted bits + 1**
 $-19 = 1\ 1101101$
 • Addition \rightarrow addition: $(19 + -19 = 0)$
 • Subtraction \rightarrow subtraction: $(0 - -19 = 19)$
 • One single zero value

- Printf format string bug
 - Printf format strings
 - Printf format string bug – exploitation
 - Exercise Printf
- Printf format string problem – best practices
- Some other input validation problems
 - Array indexing – spot the bug!
 - Off-by-one and other null termination errors
 - The Unicode bug
- Path traversal vulnerability
 - Path traversal – weak protections
 - Path traversal – best practices
- Log forging
 - Some other typical problems with log files
- Improper use of security features
 - Typical problems related to the use of security features
 - Password management
 - Exercise – Weakness of hashed passwords
 - Password management and storage
 - Special purpose hash algorithms for password storage
 - Argon2 and PBKDF2 implementations in C/C++
 - bcrypt and scrypt implementations in C/C++
 - Case study – the Ashley Madison data breach
 - Typical mistakes in password management
 - Exercise – Hard coded passwords
 - Sensitive information in memory
 - Protecting secrets in memory
 - Sensitive info in memory - minimize the attack surface
 - Your secrets vs. dynamic memory
 - Zeroisation
 - Zeroisation vs. optimization – Spot the bug!
 - Copies of sensitive data on disk
 - Core dumps
 - Disabling core dumps
 - Swapping
 - Memory locking - preventing swapping
 - Problems with page locking
 - Best practices
 - Time and state problems
 - Time and state related problems
 - Serialization errors
 - Exercise TOCTTOU
 - Best practices against TOCTTOU



Principles of security and secure coding

- Matt Bishop's principles of robust programming
- The security principles of Saltzer and Schroeder

Knowledge sources

- Secure coding sources – a starter kit
- SEI CERT sources
- Vulnerability databases
- Recommended books – C/C++