scademy
secure coding academy

C and C++ secure coding (ARM) and security testing

CL-CPAT | Virtual classroom | 4 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 test and write secure code in C and C++. We quickly 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. In the second part of the course, you will learn about cryptography and security testing to make your secure coding knowledge complete.

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, open-source testing tools and visual slides are blended seamlessly to create the best learning experience.

Outline:

- IT security and secure coding
- ARM machine code, memory layout and stack operations
- Buffer overflow
- XML security
- Common coding errors and vulnerabilities
- Denial of service
- Practical cryptography
- Security in the software development lifecycle
- Security testing
- Security testing techniques and tools
- Deployment environment

Principles of security and secure coding
Knowledge sources

Participants attending this course will:

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
Have a practical understanding of cryptography
Understand security considerations in the SDLC
Understand security testing approaches and methodologies
Get practical knowledge in using security testing techniques and tools
Learn how to set up and operate the deployment environment securely
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-STS - Security testing (Onsite / Virtual classroom, 3 days)

scademy
secure coding academy

Detailed table of contents

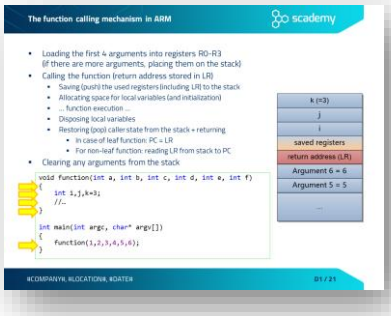
Day 1

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

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

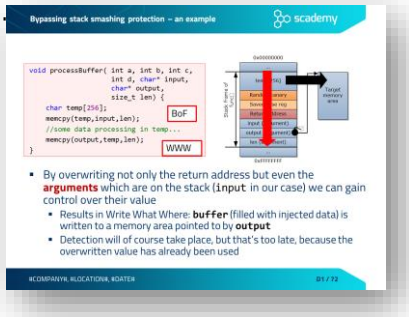


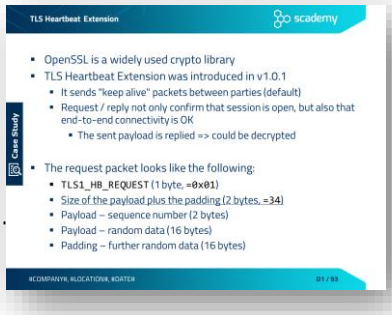
Buffer overflow

- Stack overflow
  - Buffer overflow on the stack
  - Overwriting the return address
  - Exercises – introduction
  - Exercise - Tools

- Exercises - QEMU
- Exercise BOFIntro
- Exercise BOFShellcode

- 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)
  - Exercise BOFShellcode – Using the Fortify compiler option
- Stack smashing protection
  - Stack smashing protection variants
  - Stack smashing protection in GCC
  - Exercise BOFShellcode – Stack smashing protection
  - Effects of stack smashing protection – prologue
  - Effects of stack smashing protection – epilogue
  - Bypassing stack smashing protection – an example.....
  - Overwriting arguments – Mitigation
- 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
  - Exercise BOFShellcodeEnforcing NX memory segments
- 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

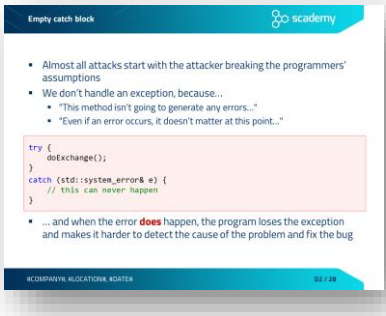
Day 2

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

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

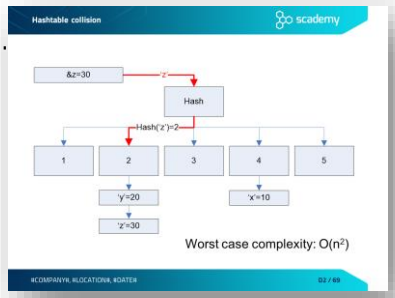


- Code quality problems
  - Dangers arising from poor code quality
  - Poor code quality – spot the bug!
  - Unreleased resources
  - Type mismatch – Spot the bug!
  - Exercise TypeMismatch
  - Memory allocation problems
    - Smart pointers
    - Zero length allocation
    - Double free
    - Mixing delete and delete[]

- Use after free
  - Use after free – Instance of a class
  - Spot the bug
  - Use after free – Dangling pointers
  - Case study - WannaCry

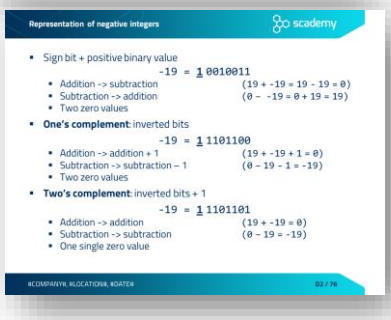
Denial of service

- 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



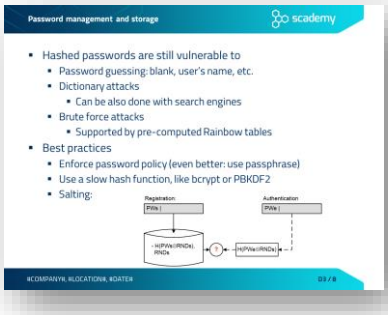
    - Integer ranges
    - Integer overflow
    - Integer problems in C/C++
    - The integer promotion rule in C/C++
    - Arithmetic overflow – spot the bug!
    - Exercise IntOverflow
    - What is the value of abs(INT_MIN)?
    - Signedness bug – spot the bug!
    - Integer truncation – spot the bug!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

Day 3

Common coding errors and vulnerabilities

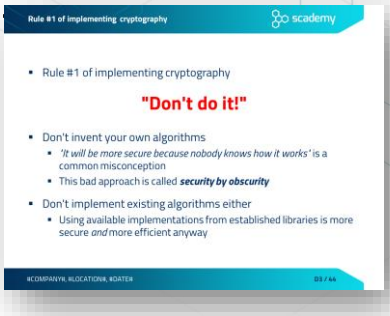
- Improper use of security features
  - Typical problems related to the use of security features
  - Password management
    - Exercise – Weakness of hashed passwords
    - Password management and storage .....
    - Special purpose hash algorithms for password storage
    - Argon2 and PBKDF2 implementations 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



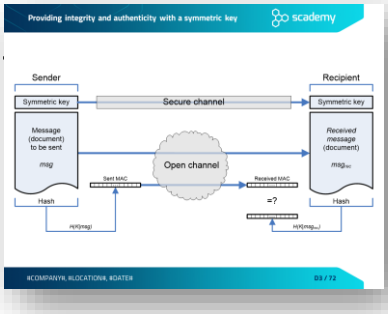
  - Sensitive information in memory
    - Protecting secrets in memory
    - Sensitive info in memory - minimize the attack surface
    - Your secrets vs. dynamic memory
    - Zeroisation
    - Zeroisation vs. optimization – Spot the bug!
    - Copies of sensitive data on disk
    - Core dumps
    - Disabling 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

Practical cryptography

- Rule #1 of implementing cryptography.....

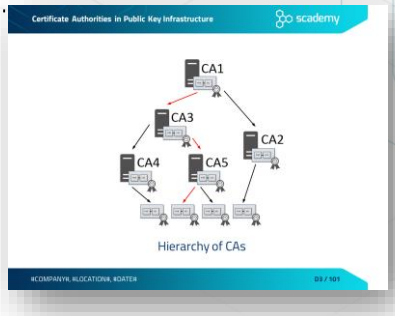


- Cryptosystems
  - Elements of a cryptosystem
  - FIPS 140-2
- Symmetric-key cryptography
  - Providing confidentiality with symmetric cryptography
  - Symmetric encryption algorithms
  - Modes of operation
  - Comparing the modes of operation
  - Symmetric encryption with OpenSSL: encryption
  - Symmetric encryption with OpenSSL: decryption
- Other cryptographic algorithms
  - Hash or message digest
  - Hash algorithms
  - SHAttered
  - Hashing with OpenSSL
  - Message Authentication Code (MAC)
  - Providing integrity and authenticity with a symmetric key.....



  - Random number generation
    - Random numbers and cryptography
    - Cryptographically-strong PRNGs
    - Weak PRNGs in C and C++
    - Stronger PRNGs in C
    - Generating random numbers with OpenSSL
    - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
  - Providing confidentiality with public-key encryption
  - Rule of thumb – possession of private key
  - The RSA algorithm
    - Introduction to RSA algorithm
    - Encrypting with RSA
    - Combining symmetric and asymmetric algorithms
    - Digital signing with RSA
    - Asymmetric encryption with OpenSSL
    - Digital signatures with OpenSSL
- Public Key Infrastructure (PKI)
  - Root of Trust Concept
    - Man-in-the-Middle (MitM) attack
    - Digital certificates against MitM attack

- Certificate Authorities in Public Key Infrastructure
- X.509 digital certificate



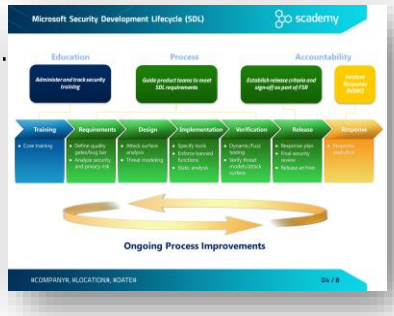
Buffer overflow

- 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

Day 4

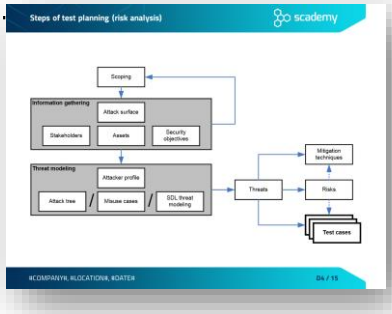
Security in the software development lifecycle

- Building Security In Maturity Model (BSIMM)
- Software Assurance Maturity Model (SAMM)
- Microsoft Security Development Lifecycle (SDL)

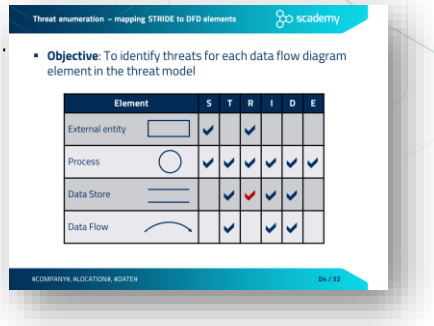


Security testing

- Functional testing vs. security testing
- Security vulnerabilities
- Prioritization – risk analysis
- Security assessments in various SDLC phases
- Security testing methodology
  - Steps of test planning (risk analysis)
  - Scoping and information gathering
    - Stakeholders
    - Assets
    - Security objectives for testing
  - Threat modeling
    - Attacker profiles
    - Threat modeling
    - Threat modeling based on attack trees
    - Threat modeling based on misuse/abuse cases
    - Misuse/abuse cases – a simple example
    - SDL threat modeling



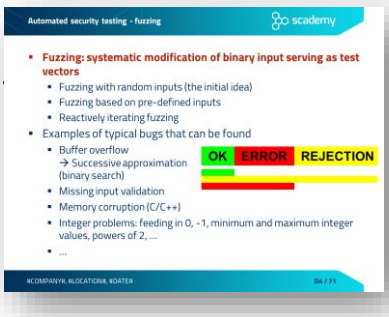
- The STRIDE threat categories
- Diagramming – elements of a DFD
- Data flow diagram – example
- Threat enumeration – mapping STRIDE to DFD elements
- Risk analysis – classification of threats
- The DREAD risk assessment model



- Testing steps
  - Deriving test cases
  - Accomplishing the tests
  - Processing test results
  - Mitigation concepts
  - Standard mitigation techniques of MS SDL
  - Review phase

Security testing techniques and tools

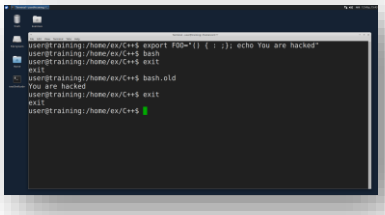
- General testing approaches
- Design review
  - Assessment of security requirements
  - Identifying security-critical aspects – hotspots
- Source code review
  - Code review for software security
  - Taint analysis
  - Heuristic-based
  - Static code analysis
    - Exercise – Static code analysis using FlawFinder
- Testing the implementation
  - Manual vs. automated security testing
  - Penetration testing
  - Stress tests
  - Binary and memory analysis
    - Exercise – Binary analysis with strings
  - Instrumentation libraries and frameworks
    - Exercise – Using Valgrind
  - Fuzzing
    - Automated security testing - fuzzing
    - Challenges of fuzzing
    - Exercise – Fuzzing with AFL (American Fuzzy Lop)



scademy
secure coding academy

Deployment environment

- Assessing the environment
  - Searching for online devices with SHODAN
  - Exercise – using SHODAN
  - Finding weaknesses with search engines
  - Exercise – Finding weaknesses with search engines
  - Password audit
  - Exercise – using John the Ripper
  - Testing random number generators
  - Exercise – Testing random number generators
- Configuration management
- Hardening
  - Network-level hardening
  - Hardening the deployment – server administration
  - Hardening the deployment – access control
- Patch and vulnerability management
  - Patch management
  - Vulnerability repositories
  - Vulnerability attributes
  - Software identification through CPE and SWID
  - Common Vulnerability Scoring System – CVSS
  - Vulnerability management software
  - Case study - Shellshock
    - Shellshock – basics of using functions in bash
    - Shellshock – vulnerability in bash
    - Exercise - Shellshock
    - Shellshock fix and counterattacks
    - Exercise – Command override with environment variables



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
- Vulnerability databases
- Recommended books – C/C++