

Secure coding for medical device manufacturers

CL-VMEDD | Classroom | 3 days

Variants: x86, x64, ARM

Audience: C and C++ developers developing medical devices

Preparedness: General C/C++ development

Exercises: Hands-on

The past few years have seen a massive increase in attacks, data breaches and medical identity theft targeting the healthcare industry; there have also been various ransomware attacks paralyzing healthcare computer networks as well as the various medical devices connected to them. The rise of mobile devices used in the industry needs to be addressed as well: there is a huge growth of medical software applications for mobiles and tablets that connect the patient with the organization – carrying and storing personally identifiable information (PII).

Healthcare is one of the business domains where security is absolutely crucial. Vulnerability is not an option when working with life-saving devices. There is also significant compliance pressure – if you want to stay a trusted and reliable vendor, your systems and applications need to comply with Health Information Portability and Accountability Act (HIPAA) requirements. To deal with these challenges, you need motivated secure coders with the right skills and the right attitude to fight security problems: a skilled team of software engineers as well as network administrators.

This training program exclusively targets engineers developing applications or maintaining networks for the healthcare sector. Our dedicated trainers share their experience and expertise through hands-on labs, and give real-life case studies from the healthcare industry – engaging participants in live hacking fun to reveal all consequences of insecure coding.

Outline:

- IT security and secure coding
- Special threats in the healthcare sector
- Regulations and standards
- x86 machine code, memory layout and stack operations
- Buffer overflow

Practical cryptography
Common coding errors and vulnerabilities
Principles of security and secure coding
Knowledge sources

Participants attending this course will:

Understand basic concepts of security, IT security and secure coding
Understand special threats in the healthcare sector
Understand regulations and standards
Realize the severe consequences of unsecure buffer handling
Understand the architectural protection techniques and their weaknesses
Have a practical understanding of cryptography
Learn about typical coding mistakes and how to avoid them
Be informed about recent vulnerabilities in various platforms, frameworks and libraries
Get sources and further readings on secure coding practices

Related courses:

- CL-CPS - C and C++ secure coding (Classroom, 3 days)
- CL-CCP - Comprehensive C and C++ secure coding (Classroom, 4 days)
- CL-CSM - C and C++ security master course (Classroom, 5 days)
- CL-CJW - Combined C/C++, Java and Web application security (Classroom, 4 days)
- CL-CNA - Combined C#, C/C++ and Web application security (Classroom, 4 days)
- CL-AAN - Android Java and native code security (Classroom, 4 days)
- CL-CTS - Security testing native code (Classroom, 3 days)

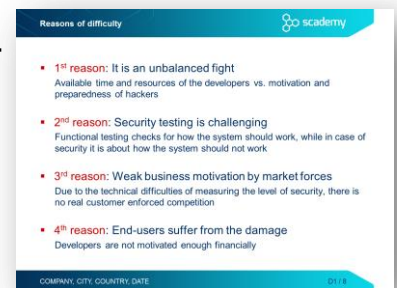
Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of certain security problems. All exercises are prepared in a plug-and-play manner by using a pre-set desktop virtual machine, which provides a uniform development environment.

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
 - Reasons of difficulty.....
 - From an infected computer to targeted attacks
- Classification of security flaws
 - Landwehr’s taxonomy
 - The Seven Pernicious Kingdoms
 - OWASP Top Ten 2017
 - CWE/SANS top 25 most dangerous software errors
 - SEI CERT secure coding standards



Special threats in the healthcare sector

- Threats in healthcare – trends and numbers
- Attacker model
- Most significant targets
- Industry and regulatory response to threats
- How is cybersecurity different for medical devices?
- Attacker tools and vectors

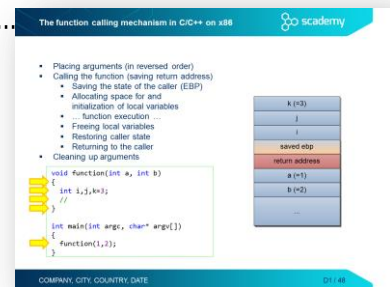
Regulations and standards

- HIPAA
 - What is HIPAA?
 - Amendments
 - Who needs to be regulated by HIPAA?
 - General safety requirements
 - Implementation requirements

- ANSI/UL 2900-2-1: Standard for Healthcare Systems
 - High-level security concepts and goals
 - General security controls
 - Access control, authentication and authorization
 - Remote communication and cryptography
 - Product management
 - Risk management
 - Security testing medical devices

x86 machine code, memory layout and stack operations

- Intel 80x86 Processors – main registers
- Intel 80x86 Processors – most important instructions
- Intel 80x86 Processors – flags
- Intel 80x86 Processors – control instructions
- Intel 80x86 Processors – stack handling and flow control
- The memory address layout
- The function calling mechanism in C/C++ on x86.....
- Calling conventions
- The local variables and the stack frame
- Function calls – prologue and epilogue of a function
- 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 BOFIntro
 - Exercise BOFShellcode

Day 2

Buffer overflow

- Protection against stack overflow
 - Specific protection methods
 - Protection methods at different layers
 - The PreDeCo matrix of software security
 - Stack overflow – Prevention (during development)
 - 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
- Address Space Layout Randomization (ASLR)
 - Randomization with 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
- Return-to-libc attack – Circumventing the NX bit protection
 - Circumventing memory execution protection
 - Return-to-libc attack
- Return oriented programming (ROP)
 - Exploiting with ROP
 - ROP gadgets
- 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

scademy

Case Study

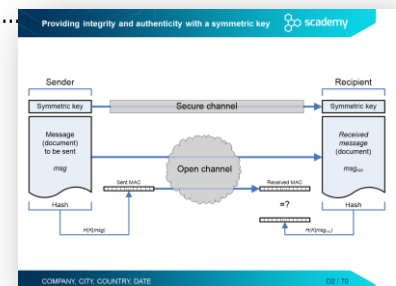
- OpenSSL is a widely used crypto library
- TLS Heartbeat Extension was introduced in v1.0.1
 - It sends "keep alive" packets between parties (default)
 - Request / reply not only confirm that session is open, but also that end-to-end connectivity is OK
 - The sent payload is replied => could be decrypted
- The request packet looks like the following:
 - TLS1_HB_REQUEST (1 byte, 0x01)
 - Size of the payload plus the padding (2 bytes, 0x34)
 - Payload – sequence number (2 bytes)
 - Payload – random data (16 bytes)
 - Padding – further random data (16 bytes)

COMPANY, CITY, COUNTRY, DATE 02 / 41

- Protection against heap overflow

Practical cryptography

- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms
 - SHattered
 - Message Authentication Code (MAC)
 - Providing integrity and authenticity with a symmetric key.....
 - Random numbers and cryptography
 - Cryptographically-strong PRNGs
 - 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
- Public Key Infrastructure (PKI)
 - Man-in-the-Middle (MitM) attack
 - Digital certificates against MitM attack
 - Certificate Authorities in Public Key Infrastructure
 - X.509 digital certificate



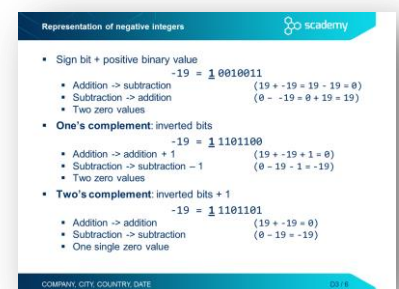
Common coding errors and vulnerabilities

- 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

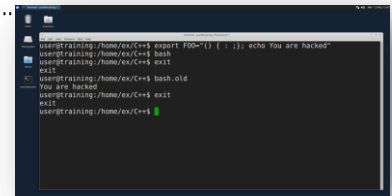
Day 3

Common coding errors and vulnerabilities

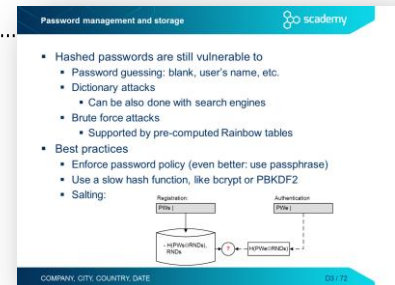
- Input validation
 - Input validation concepts
 - Integer problems
 - Representation of negative integers
 - Integer ranges
 - Integer overflow
 - Integer problems in C/C++
 - The integer promotion rule in C/C++
 - Arithmetic overflow – spot the bug!
 - Exercise IntOverflow
 - What is the value of abs(INT_MIN)?
 - 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 exploit – overwriting the return address



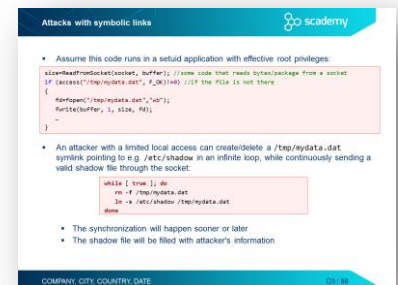
- Printf format string problem – best practices
- Some other input validation problems
 - Array indexing – spot the bug!
 - Off-by-one and other null termination errors
 - The Unicode bug
- Log forging
 - Some other typical problems with log files
- Case study - Shellshock
 - Shellshock – basics of using functions in bash
 - Shellshock – vulnerability in bash
 - Exercise - Shellshock.....
 - Shellshock fix and counterattacks
 - Exercise – Command override with environment variables



- Improper use of security features
 - Typical problems related to the use of security features
 - Insecure randomness
 - Weak PRNGs in C and C++
 - Stronger PRNGs in C
 - 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
 - The loginkey token
 - Revealing the passwords with brute forcing
 - Typical mistakes in password management
 - Exercise – Hard coded passwords
 - Sensitive info in memory - minimize the attack surface



- Time and state problems
 - Time and state related problems
 - Serialization errors (TOCTTOU)
 - Attacks with symbolic links
 - 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
- Vulnerability databases
- Healthcare cybersecurity resources
- Recommended books – C/C++