TRAINING BROCHURE

Secure coding in C and C++ training

Provisional reservation

Book now

high tech institute

Secure coding in C and C++

Price: € 2,400 excl. VAT *

Duration: 3 consecutive days

Contact: training@hightechinstitute.nl, +31 85 401 3600

Score: 9.2 ★★★★☆

Pitch: https://youtu.be/MlqfWX6krJc

Intro

Your application written in C or C++ works as intended, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -2^32? Because that's what the bad guys will do – and the list is far from complete.

Handling security needs a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hand on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences and best practices are our blood, sweat and tears.

All this is put in the context of C and C++, and extended by core programming issues, discussing security pitfalls of code written in these languages.

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

So that nothing unexpected happens.

Nothing.

PRACTICAL INFO

- *The 'Secure coding in C and C++' training can be organized as in-company training as well.*
- *If on-site training is not feasible, we can discuss providing a live, interactive online (virtual) or hybrid training. The standard program with 3-day content can also be delivered in 5 half days (from Monday to Friday).*
- *Curious about how to quantify the return on investment (ROI) of secure coding trainings? Check out this article.*

Objective

- Explain approaches in handling security challenges in code;
- Identify security vulnerabilities and their consequences;
- Learn the best practices in how to avoid these mistakes.

Target audience

This course is intended for C and C++ developers.

Program

Day 1

Security basics

Certification

After attending this training, participants will receive a High Tech Institute certificate.

Trainers

Ernő Jeges MSc
Balazs Kiss

** Prices are subject to change. Price correction will be applied at the end of the year.*

Keep me posted

What is security?

Threat and risk

Types of threats against computer systems

Consequences of insecure software

Constraints and the market

Bugs, vulnerabilities and exploits

Categorization of bugs

- Seven pernicious kingdoms
- Common Weakness Enumeration (CWE)
- CWE/SANS Top 25 Most Dangerous Software Errors
- SEI Cert Secure Coding Guidelines
- Vulnerabilities in the environment and the dependencies

Buffer overflow

x86 assembly and calling conventions

- X86 assembly essentials
- Registers and addressing
- Instructions
- Calling conventions on x86
- Calling convention – what it is all about
- The stack frame
- Prologue and epilogue
- Stacked function calls
- Recursion

Buffer overflow on the stack

- Buffer overflow – basics
- Buffer overread and overwrite
- Stack smashing
- Exploitation – Hijacking the control flow
- Lab – Buffer overflow 101, code reuse
- Exploitation – Injecting a shellscript
- Lab – Code injection, BoF exploitation with a shellcode
- Buffer overflow on the heap
- Buffer overflow on the heap – an example exploitation
- Lab – Heap overflow
- Heap overflow best practices
- Case study – Heartbleed
- Lab - Heartbleed
- Pointer subterfuge
- Pointer manipulation
- Write-what-where
- Modification of jump tables
- Hijacking GOT and RELRO protection
- Overwriting function pointers
- Lab – Overwriting virtual function table
- Some typical mistakes leading to BoF
- Off-by-one
- Allocating nothing
- String length calculation mistakes
- Lab – Analyze UTF-8 encoding
- String termination confusion
- Lab – String termination confusion
- Other typical BoF weaknesses

BoF protection best practices

- Safe and unsafe function
- base_string and std::string
- Some less-known dangerous function

- Lab – Fixing buffer overflow
- Compiler options and instrumentation
- Using FORTIFY_SOURCE
- Lab – Effects of FORTIFY
- Compile-time instrumentation
- Stack smashing protection
- Detecting BoF with the canary
- Argument cloning
- Stack smashing protection on various platforms
- The changed prologue and epilogue
- Lab – Effects of stack smashing protection
- Runtime protection
- Runtime instrumentation
- Address Space Layout Randomization (ASLR)
- ASLR on various platforms
- Lab – Effects of ASLR
- Circumventing ASLR – NOP sledging
- 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
- Arc Injection – Return-to-libc
- Lab – Exploit return-to-libc
- Cascading return-to-libc
- Return Oriented Programming (ROP)
- Lab – ROP demonstration
- Whatever Oriented Programming
- Protection against ROP

Day 2

Common software security weaknesses

Input validation

- Input validation principles
- Blacklists and whitelists
- Validation with regex
- What to validate – the attack surface
- When to validate – validation vs transformations
- Where to validate – defense in depth
- Injection
- Injection principles
- Injection attacks
- Code injection
- Command injection
- Lab – Command injection
- Command injection best practices
- Case study – Shellshock
- Lab - Shellshock
- Process control – library injection
- DLL hijacking
- Lab – DLL hijacking
- Injection best practices
- Input validation
- Output sanitization
- Encoding and escaping the output
- Encoding challenges
- Integer handling
- Representing signed numbers
- Integer visualization
- Integer problems
- Integer overflow
- Lab – Integer overflow
- Case study – Android Stagefright

- Signed / unsigned confusion
- Lab – Signed / unsigned confusion
- Integer truncation
- Case study – Wannacry
- Best practices
- Upcasting
- Precondition testing
- Postcondition testing
- Using big integer libraries
- Lab – Integer handling best practices
- The AIR integer model
- Other numeric problems
- Division by zero
- Working with floating-point numbers
- Format string issues
- The problem with printf()
- Format specifiers of printf()
- Exploiting the printf format string weakness
- Lab – Exploiting format string
- Some other input validation problems
- Improper address validation in IOCTL

Security features

- Authentication
- Authentication basics
- Authentication weaknesses
- Case study – PayPal two factor authentication bypass
- User interface best practices
- Password management
- Inbound password management
- Storing account passwords
- Plaintext passwords at Facebook
- Lab – Why just hashing passwords is not enough?
- Dictionary attacks and brute forcing
- Salting
- Adaptive hash functions for password storage
- Password in transit
- Password policy
- Weak and strong passwords
- Using passphrases
- Lab – Applying a password policy
- The Ashley Madison data breach
- The dictionary attack
- The ultimate attack
- Exploitation of the results and the lessons learnt
- Outbound password management
- Hard coded passwords
- Lab – Hardcoded password
- Password in configuration file
- Protecting sensitive information in memory
- Challenges in protecting memory
- Heap inspection
- Compiler removal of memory clearing code
- Sensitive information in non-locked memory

- Authorization
- Access control basics
- Missing or improper authorization
- File system access control
- Improper file system access control
- Ownership
- chroot jail
- Using umask()
- Linux filesystem
- LDAP

- Access control in databases
- Lab – Database access control
- Privileges and permissions
- Permission manipulation
- Incorrect use of privileged APIs
- Exposed IOCTL with Insufficient access control
- Permission best practices
- Principle of least privilege
- Principle of separation of privileges
- Permission granting
- Privilege dropping
- Handling of insufficient privileges

- Information exposure
- Exposure through extracted data and aggregation
- System information leakage
- Leaking system information
- Relying on accessibility modifiers
- Lab – Inappropriate protection by accessibility modifier
- Information exposure best practices

- UI security
- UI security principles
- Sensitive information in the user interface
- Misinterpretation of UI features or actions
- Insufficient UI feedback
- Relying on hidden or disabled UI element
- Lab – Hidden or disabled UI element
- Insufficient anti-automation

Day 3

Common software security weaknesses

Time and state

- Thread management best practices
- Thread management best practices in C/C++
- Race conditions
- Race condition in object data members
- Lab – Race condition
- File race condition
- Time-of-check-to-time-of-usage (TOCTTOU)
- Lab - TOCTTOU
- Insecure temporary file
- Potential race condition in C/C++
- Race condition in signal handling
- Forking
- Bit-field access
- Mutual exclusion and locking
- Deadlocks
- Lab – Locking
- Synchronization and thread safety
- Synchronization and thread safety in C/C++

Errors

- Error and exception handling principles
- Error handling
- Returning a misleading status code
- Error handling in C
- Error handling in C++
- Information exposure through error reporting
- Exception handling
- In the catch block. And now what?
- Empty catch block
- Best practices for catch blocks
- Overly broad throws
- Catching NULL pointer exceptions

- Exception handling in C++
- Lab – Exception handling mess

Code quality

- Data
- Type mismatch
- Lab – Type mismatch
- Function return values
- Unchecked Return Value
- Case study – MacOS X password hash change
- Omitted return value
- Returning unmodifiable pointer
- Initialization and cleanup
- Uninitialized variable
- Constructors and destructors
- Class initialization cycles
- Declaration and allocation issues in C
- Allocation and deallocation in C++
- Unreleased resource
- Array disposal
- Lab – Mixing delete and delete[]
- Object oriented programming pitfalls
- Accessibility modifiers
- Inheritance and overriding
- Implementing the copy operator
- Mutability
- Cloning
- Cloning sensitive classes – object hijacking
- Object hijacking – best practices
- Serialization

Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop;
- Secure design principles of Saltzer and Schröder;
- Some more principles.

And now what?

- Further sources and readings;
- .NET and C# resources;
- Further labs and challenges to do.

Methods

A blended learning journey: live instructor-led training with lab exercises in a top-notch e-learning system. You will keep access to the e-learning system 3-months post-training to revisit the lab exercises and material.

Platform: Linux, Windows.
Labs: Hands-on.

Frequency

Once per year

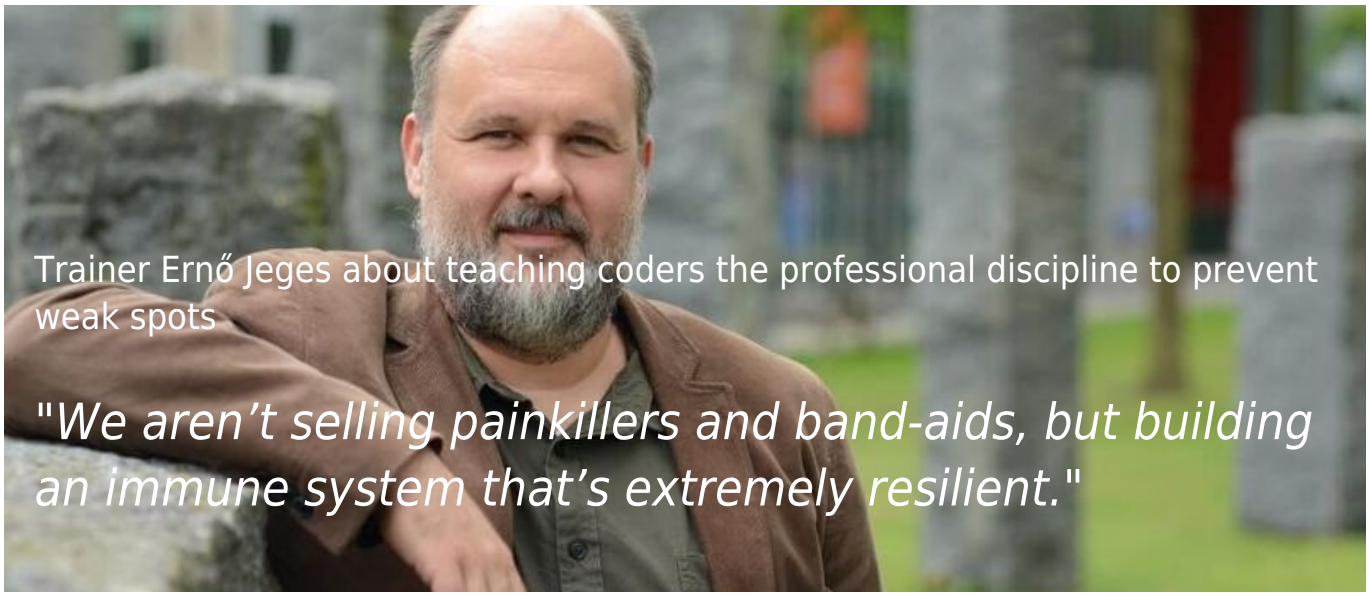
More information

The 5-step Teaching Method

In this video the didactic method is explained ensuring that particpants will leave the training equipped with the best practices to apply the very next day.

Watch video

Read the interview:



Remarks from participants:

- "Big insights how security can be breached and how to fix it." > Ference Schopbarteld – Thales Group
- "Balance between old & new techniques. Good hands-on training." > Hani S. – Sioux Technologies
- "Interesting, challenging and technical tough. Best part is the hands-on." > Gabriele Ricciardi – Thales Group