

Advanced software security - beyond ethical hacking

CL-BEH | Classroom | 5 days

Variants: C#, Java, C++

Audience: Software engineers

Preparedness: General software development

Exercises: Hands-on

Are you looking to start your journey towards becoming a security champion?

Despite all of your efforts, the code you have been writing your entire career is full of weaknesses you never knew existed. You will be provided with all of the attackers' tricks and how to mitigate them, leaving you with no other feeling than the desire to know more.

This is a fully customizable 5 day long course ideal for advanced software engineers, or as a perfect complement to a coding bootcamp.

It is your choice to be ahead of the pack, and be seen as a game changer in the fight against cybercrime.

The secure coders' community awaits you.

Outline:

- IT security and secure coding
- Web application security
- Client-side security
- Practical cryptography
- Cryptographic vulnerabilities
- x86 machine code, memory layout and stack operations
- Buffer overflow and its exploitation
- Some additional native code-related vulnerabilities
- XML security
- Denial of service
- Input validation
- Error and exception handling

Improper use of security features
Code quality problems
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
Learn Web vulnerabilities beyond OWASP Top Ten and know how to avoid them
Learn about XML security
Learn client-side vulnerabilities and secure coding practices
Have a practical understanding of cryptography
Understand some recent attacks against cryptosystems
Realize the severe consequences of unsecure buffer handling in native code
Understand the architectural protection techniques and their weaknesses
Realize the severe consequences of unsecure buffer handling
Learn about denial of service attacks and protections
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-JWA - Java and Web application security (Classroom, 3 days)
- CL-ANS - Secure desktop application development in C# (Classroom, 3 days)
- CL-NWA - C# and Web application security (Classroom, 3 days)
- CL-WSC - Web application security (Classroom, 3 days)
- CL-WTS - Web application security testing (Classroom, 3 days)
- CL-STS - Security testing (Classroom, 3 days)
- CL-CSM - C and C++ security master course (Classroom, 5 days)
- CL-JSM - Java and Web application security master course (Classroom, 5 days)
- CL-NSM - C# and Web application security master course (Classroom, 5 days)



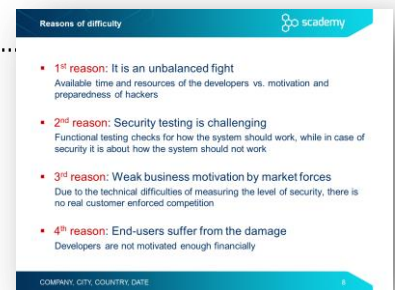
Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of 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

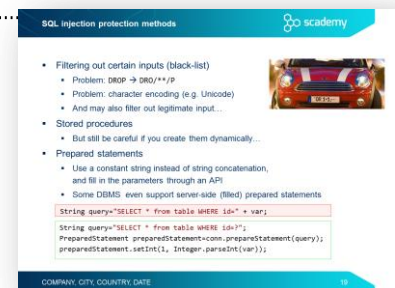
IT security and secure coding

- 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

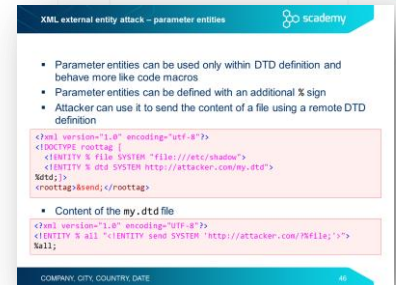


Web application security

- Injection
 - Injection principles
 - SQL injection
 - Exercise – SQL injection
 - Typical SQL Injection attack methods
 - Blind and time-based SQL injection
 - SQL injection protection methods.....
 - Effect of data storage frameworks on SQL injection in Java
 - Other injection flaws
 - Command injection
 - Case study – ImageMagick
- Broken authentication
 - Session handling threats
 - Session handling best practices
 - Session handling in Java
 - Setting cookie attributes – best practices
 - Cross site request forgery (CSRF)
 - CSRF prevention
 - CSRF prevention in Java frameworks
- Sensitive data exposure
 - Transport layer security
 - Enforcing HTTPS



- XML external entity (XXE)
 - XML Entity introduction
 - XML external entity attack (XXE) – resource inclusion
 - XML external entity attack – URL invocation
 - XML external entity attack – parameter entities
 - Exercise – XXE attack
 - Preventing entity-related attacks
 - Case study – XXE in Google Toolbar
- Broken access control
 - Typical access control weaknesses
 - Insecure direct object reference (IDOR)
 - Exercise – Insecure direct object reference
 - Protection against IDOR
 - Case study – Facebook Notes
- Cross-Site Scripting (XSS)
 - Persistent XSS
 - Reflected XSS
 - DOM-based XSS
 - Exercise – Cross Site Scripting
 - XSS prevention
 - XSS prevention tools in Java and JSP
- Insecure deserialization
 - Deserialization basics
 - Security challenges of deserialization
 - Deserialization in Java
 - From deserialization to code execution
 - POP payload targeting the Apache Commons gadget (Java)
 - Real-world Java examples of deserialization vulnerabilities
 - Issues with deserialization – JSON
 - Best practices against deserialization vulnerabilities

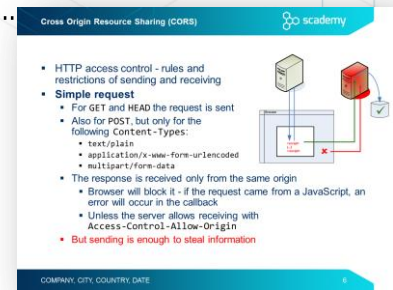


Day 2

Client-side security

- JavaScript security
- Same Origin Policy

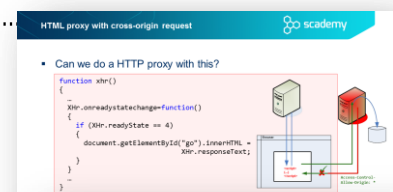
- Cross Origin Resource Sharing (CORS).....
- Exercise – Client-side authentication
- Client-side authentication and password management
- Protecting JavaScript code
- Clickjacking
 - Exercise – Do you Like me?
 - Protection against Clickjacking
 - Anti frame-busting – dismissing protection scripts
 - Protection against busting frame busting
- AJAX security
 - XSS in AJAX
 - Script injection attack in AJAX
 - Exercise – XSS in AJAX
 - XSS protection in Ajax
 - Exercise CSRF in AJAX – JavaScript hijacking
 - CSRF protection in AJAX
- HTML5 security
 - New XSS possibilities in HTML5
 - Client-side persistent data storage
 - HTML5 clickjacking attack – text field injection
 - HTML5 clickjacking – content extraction
 - Form tampering
 - Exercise – Form tampering
 - Cross-origin requests
 - HTML proxy with cross-origin request.....
 - Exercise – Client side include



Cross Origin Resource Sharing (CORS)

- HTTP access control - rules and restrictions of sending and receiving
- **Simple request**
 - For GET and HEAD the request is sent
 - Also for POST, but only for the following Content-Types:
 - text/plain
 - application/x-www-form-urlencoded
 - multipart/form-data
- The response is received only from the same origin
 - Browser will block it - if the request came from a JavaScript, an error will occur in the callback
 - Unless the server allows receiving with Access-Control-Allow-Origin
 - **But sending is enough to steal information**

COMPANY, CITY, COUNTRY, DATE



HTML proxy with cross-origin request

Can we do a HTTP proxy with this?

```
function xhr() {
  {
    xhr.onreadystatechange=function()
    {
      if (xhr.readyState == 4)
      {
        document.getElementById("go").innerHTML +=
          xhr.responseText;
      }
    }
  }
}
```

COMPANY, CITY, COUNTRY, DATE



Rule #1 of implementing cryptography

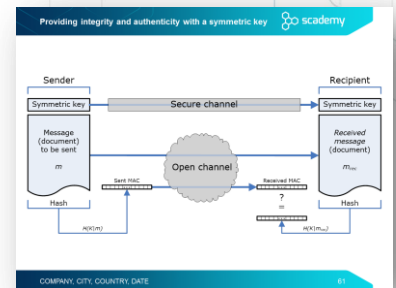
- Rule #1 of implementing cryptography
- **"Don't do it!"**
- Don't invent your own algorithms
 - "It will be more secure because nobody knows how it works" is a common misconception
 - It is **security by obscurity**
- Don't implement existing algorithms either
 - Using available implementations from established libraries is more secure and more efficient anyway

COMPANY, CITY, COUNTRY, DATE

Practical cryptography

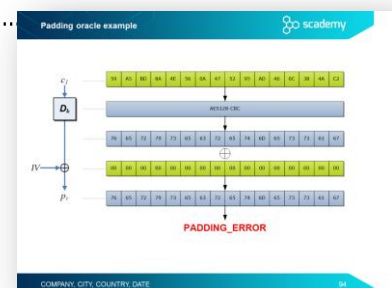
- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms

- 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



Cryptographic vulnerabilities

- Protocol-level vulnerabilities
 - BEAST
- Padding oracle attacks
 - Adaptive chosen-ciphertext attacks
 - Padding oracle attack
 - CBC decryption
 - Padding oracle example.....
 - POODLE

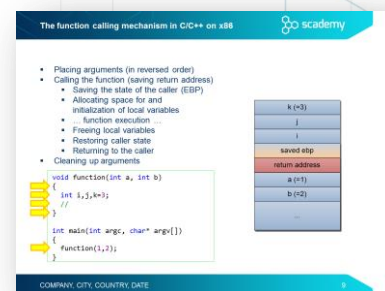


Day 3

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



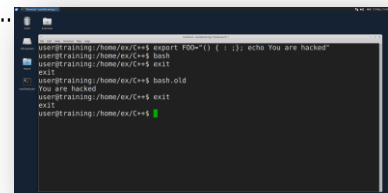
Buffer overflow and its exploitation

- Stack overflow
 - Buffer overflow on the stack
 - Overwriting the return address
 - Exercises – introduction
 - Exercise BOFIntro
 - Exercise BOFShellcode
- 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 instrumentation (FORTIFY_SOURCE)
 - Exercise BOFShellcode – Fortify
- 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

Some additional native code-related vulnerabilities

- Boundary violation
 - Array indexing – spot the bug!
 - Off-by-one and other null termination errors
 - The Unicode bug
- Code quality problems
 - 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[]
- Race condition
 - Serialization errors (TOCTTOU)
 - Attacks with symbolic links
 - Exercise TOCTTOU
- Case study - the Shellshock bash vulnerability
 - Shellshock – basics of using functions in bash
 - Shellshock – vulnerability in bash
 - Exercise - Shellshock.....
 - Shellshock fix and counterattacks
 - Exercise – Command override with environment variables



```
user@training:~/room/ox/c++$ export FOO="() { : }; echo You are hacked"
user@training:~/room/ox/c++$ bash
user@training:~/room/ox/c++$ exit
exit
user@training:~/room/ox/c++$ bash_old
You are hacked
user@training:~/room/ox/c++$ exit
exit
user@training:~/room/ox/c++$
```

Day 4

XML security

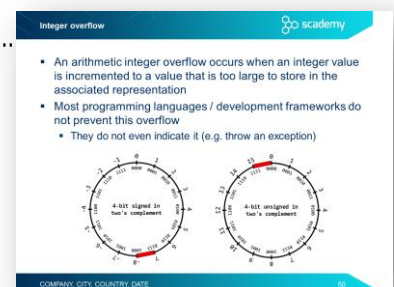
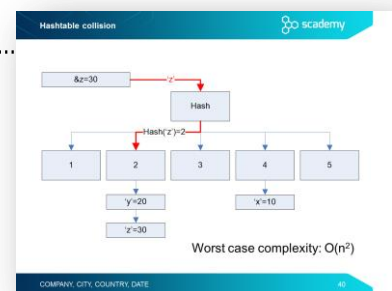
- Introduction
- XML parsing
- XML injection
 - (Ab)using CDATA to store XSS payload in XML
 - Exercise – XML injection
 - Protection through sanitization and XML validation
 - XML bomb
 - Exercise – XML bomb

Denial of service

- DoS introduction
- Asymmetric DoS
- SSL/TLS renegotiation DoS
- Asymmetric DOS with JSON deserialization
- Regular expression DoS (ReDoS)
 - Exercise ReDoS
 - ReDoS mitigation
 - Case study – ReDos in Stack Exchange
- Hashtable collision attack
 - Using hashtables to store inputs
 - Hashtable collision.....
 - Hashtable collision in Java

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 `Math.abs(Integer.MIN_VALUE)`?



- Signedness bug – spot the bug!
- Integer truncation – spot the bug!
- Integer problem – best practices
 - GCC trapv
 - Exercise IntOverflow
 - Avoiding arithmetic overflow – addition
 - Avoiding arithmetic overflow – multiplication
 - Dealing with signed/unsigned integer promotion in C
 - Safe integer handling in C
 - The SafeInt class for C++
 - Detecting arithmetic overflow in Java 8
 - Exercise – Using addExact() in Java
- Case study – Android Stagefright
 - Stagefright – a quick introduction
 - Some Stagefright code examples – spot the bugs!
- Path traversal vulnerability
 - Path traversal – best practices
- Unvalidated redirects and forwards
- Log forging
 - Some other typical problems with log files
- Printf format string bug
 - Printf format strings
 - Printf format string bug – exploitation
 - Exercise Printf
 - Printf format string exploit – overwriting the return address

Error and exception handling

- Typical problems with error and exception handling
- Empty catch block
- Overly broad throws
- Overly broad catch
- Using multi-catch
- Catching NullPointerException
- Exception handling – spot the bug!
- Exercise ScademyPay – Error handling
- Exercise – Error handling

Empty catch block scademy

- Almost all attacks start with the attacker breaking the programmers' assumptions
- We don't handle an exception, because...
 - "This method isn't going to generate any errors..."
 - "Even if an error occurs, it doesn't matter at this point..."

```

try {
    doExchange();
}
catch (RareException e) {
    // this can never happen
}

```

- ... and when the error **does** happen, the program loses the exception and makes it harder to detect the cause of the problem and fix the bug

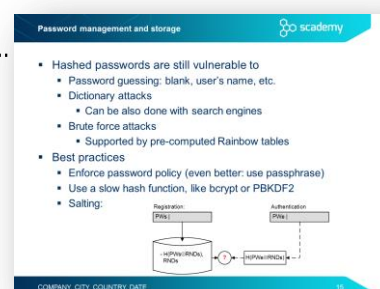
COMPIRY, CITY, COUNTRY, DATE 100

- Case study – "#iamroot" authentication bypass in macOS
 - Authentication process in macOS (High Sierra)
 - Incorrect error handling in opendir.yd
 - The #iamroot vulnerability (CVE-2017-13872)

Day 5

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
 - Weak PRNGs in Java
 - Exercise RandomTest
 - Using random numbers in Java – spot the bug!
- 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++
 - Argon2 and PBKDF2 implementations in Java
 - bcrypt and scrypt implementations in Java
 - 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
- Accessibility modifiers
 - Accessing private fields with reflection in Java
 - Exercise Reflection – Accessing private fields with reflection
- Exercise ScademyPay – Integrity protection weakness



Password management and storage

- Hashed passwords are still vulnerable to
 - Password guessing: blank, user's name, etc.
 - Dictionary attacks
 - Can be also done with search engines
 - Brute force attacks
 - Supported by pre-computed Rainbow tables
- Best practices
 - Enforce password policy (even better: use passphrase)
 - Use a slow hash function, like bcrypt or PBKDF2
 - Salting:

The diagram shows a flow from 'Registration (File)' to a database 'users (bcrypt, bcrypt, bcrypt)' and then to 'Authentication (File)'.

Code quality problems

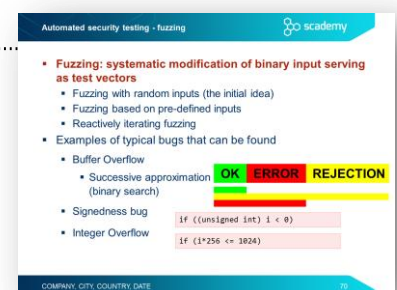
- Dangers arising from poor code quality
- Poor code quality – spot the bug!
- Unreleased resources
- Private arrays – spot the bug!
- Private arrays – typed field returned from a public method
- Exercise Object Hijack
- Public method without final – object hijacking
- Serialization – spot the bug!
- Exercise Serializable Sensitive
- Immutable String – spot the bug!
- Exercise Immutable Strings
- Immutability and security

Security testing techniques and tools

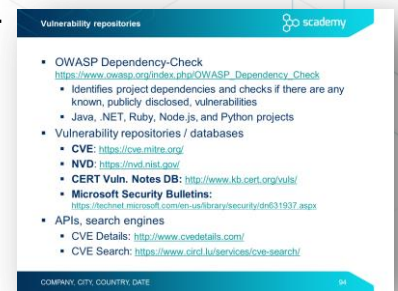
- General testing approaches
- Testing the implementation
 - Dynamic security testing
 - Manual vs. automated security testing
 - Binary and memory analysis
 - Exercise – Binary analysis with strings
 - Instrumentation libraries and frameworks
 - Exercise – Using Valgrind
 - Fuzzing
 - Automated security testing - fuzzing.....
 - Challenges of fuzzing
 - Exercise – Fuzzing with AFL (American Fuzzy Lop)
 - Web vulnerability scanners
 - Exercise – Using a vulnerability scanner
 - SQL injection tools
 - Exercise – Using SQL injection tools

Deployment environment

- Configuration management
- Hardening
- Patch management
- Assessing the environment



- Vulnerability management
 - Vulnerability repositories
 - Vulnerability attributes
 - Common Vulnerability Scoring System – CVSS
 - Vulnerability scanners



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
- Java secure coding sources
- Recommended books – C/C++
- Recommended books – Java