

Combined C#, C/C++ and Web application security

CL-CNA | Classroom | 4 days

Audience: C/C++, C# and Web application developers, architects and testers

Preparedness: Advanced C/C++, C# and Web development

Exercises: Hands-on

Serving teams that use managed code (.NET and ASP.NET typically written in C#) together with native code development (typically C/C++), this training gives a comprehensive overview of the security issues in both environments.

Concerning C/C++, common security vulnerabilities are discussed, backed by practical exercises about the attacking methods that exploit these vulnerabilities, with the focus on the mitigation techniques that can be applied to prevent the occurrences of these dangerous bugs, detect them before market launch or prevent their exploitation.

The course also covers both the various general (like web services) and specific security solutions and tools, and the most frequent and severe security flaws of managed code, dealing with both language-specific issues and the problems stemming from the runtime environment. The vulnerabilities relevant to the ASP.NET platform are detailed along with the general web-related vulnerabilities following the OWASP Top Ten list. The course consists of a number of exercises through which attendees can easily understand and execute attacks and protection methods.

Outline:

- IT security and secure coding
- Web application security (OWASP Top Ten 2017)
- Client-side security
- .NET security architecture and services
- Practical cryptography
- x86 machine code, memory layout and stack operations
- Buffer overflow
- Some additional native code-related vulnerabilities
- 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
- 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
- Learn to use various security features of the .NET development environment
- Have a practical understanding of cryptography
- 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 typical coding mistakes and how to avoid them
- Get sources and further readings on secure coding practices

Related courses:

- CL-NWA - C# and Web application security (Classroom, 3 days)
- CL-JNW - Combined Java, C# and Web application security (Classroom, 3 days)
- CL-CJW - Combined C/C++, Java and Web application security (Classroom, 4 days)
- CL-WSC - Web application security (Classroom, 3 days)
- CL-WTS - Web application security testing (Classroom, 3 days)
- CL-CSM - C and C++ security master course (Classroom, 5 days)
- CL-NSM - C# and Web application security master course (Classroom, 5 days)

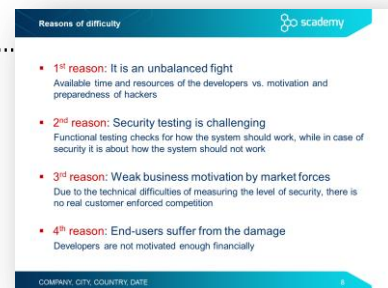
Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of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

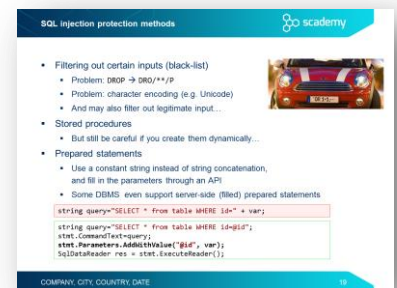
IT security and secure coding

- 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

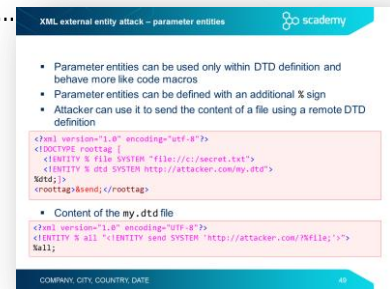


Web application security (OWASP Top Ten 2017)

- A1 - 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 .NET
 - Other injection flaws
 - Command injection
 - Command injection exercise – starting Netcat
 - Case study – ImageMagick
 - Cookie injection / HTTP parameter pollution
 - Exercise – Value shadowing
- A2 - Broken authentication
 - Session handling threats
 - Session fixation
 - Exercise – Session fixation
 - Session handling best practices
 - Setting cookie attributes – best practices
 - Cross site request forgery (CSRF)
 - CSRF prevention



- A3 - Sensitive data exposure
 - Sensitive data exposure
 - Transport layer security
- A4 - 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
- A5 - Broken access control
 - Typical access control weaknesses
 - Insecure direct object reference (IDOR)
 - Exercise – Insecure direct object reference
 - Protection against IDOR
 - Case study – Facebook Notes
- A6 - Security misconfiguration
 - ASP.NET components and environment overview
 - Insecure file uploads
 - Exercise – Uploading executable files
 - Filtering file uploads – validation and configuration
- A7 - Cross-Site Scripting (XSS)
 - Persistent XSS
 - Reflected XSS
 - DOM-based XSS
 - Exercise – Cross Site Scripting
 - XSS prevention
 - Output encoding API in C#
 - XSS protection in ASP.NET – validateRequest
 - Web Protection Library (WPL)



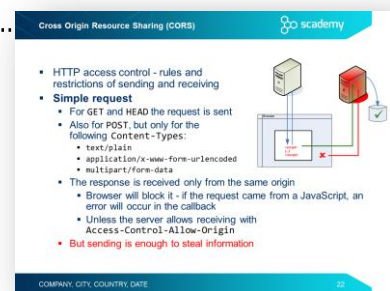
Day 2

Web application security (OWASP Top Ten 2017)

- A8 - Insecure deserialization
 - Deserialization basics
 - Security challenges of deserialization
 - Deserialization in .NET
 - From deserialization to code execution
 - POP payload targeting MulticastDelegate (C#)
 - Real-world .NET examples of deserialization vulnerabilities
 - Issues with deserialization – JSON
 - Best practices against deserialization vulnerabilities
- A9 - Using components with known vulnerabilities
- A10 - Insufficient logging and monitoring
 - Detection and response
 - Logging and log analysis
 - Intrusion detection systems and Web application firewalls

Client-side security

- JavaScript security
- Same Origin Policy
- Cross Origin Resource Sharing (CORS).....
- 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
 - Form tampering
 - Exercise – Form tampering
 - Cross-origin requests
 - HTML proxy with cross-origin request.....
 - Exercise – Client side include

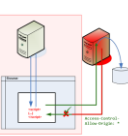
.NET security architecture and services

- .NET architecture
- Code Access Security
 - Full and partial trust
 - Evidence classes
 - Permissions
 - Code access permission classes
 - Deriving permissions from evidence
 - Defining custom permissions
 - .NET runtime permission checking
 - The Stack Walk
 - Effects of Assert().....
 - Class and method-level declarative permission
 - Imperative (programmatic) permission checking
 - Exercise – sandboxing .NET code
 - Using transparency attributes
 - Allow partially trusted callers
 - Exercise – using transparency attributes

HTML proxy with cross-origin request

```

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

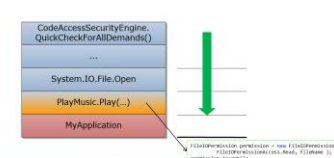


- Can we do a HTTP proxy with this?
- Reverse Web shell: a Javascript (XSS) tunneling HTTP
 - XSS + COR can be used for tunneling HTTP traffic between the user and the attacker
 - With the injected script an attacker can access vulnerable sites via the victim's browser by sending requests over the channel

COMPANY, CITY, COUNTRY, DATE 48

Effects of Assert()

- With Assert() one can perform an action "with the privilege of the calling class", which simply means that...
 - The stack will be checked up to the caller of Assert()



COMPANY, CITY, COUNTRY, DATE 61

Practical cryptography

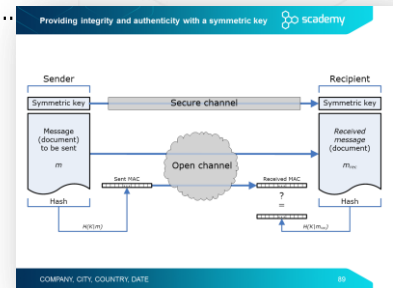
- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation

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 74

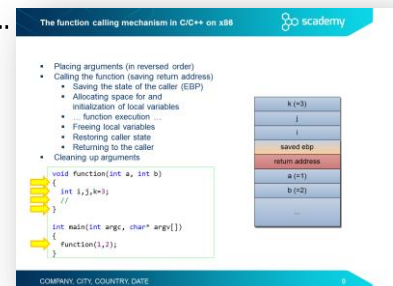
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms
 - SHAttered
 - Message Authentication Code (MAC)
 - Providing integrity and authenticity with a symmetric key.....
 - Random numbers and cryptography
 - Cryptographically-strong PRNGs
 - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
 - Providing confidentiality with public-key encryption
 - Rule of thumb – possession of private key
 - Combining symmetric and asymmetric algorithms
- 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



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



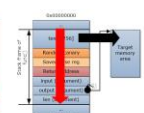
Buffer overflow

- 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
 - Bypassing stack smashing protection – an example.....
 - Overwriting arguments – Mitigation
- Address Space Layout Randomization (ASLR)
 - Randomization with ASLR
 - Using 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 BOFShellcode – Enforcing NX memory segments
- 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

By passing stack smashing protection – an example 

```

void processbuffer( char* input,
                  char* output,
                  size_t len)
{
  char temp[256];
  memcpy(temp,input,len);
  //some data processing in temp...
  memcpy(output,temp,len);
}
  
```

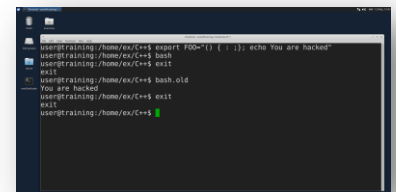


- By overwriting not only the return address but even the arguments which are on the stack (input in our case) we can gain control over their value
 - Results in Write What Where: **buffer** (filled with injected data) is written to a memory area pointed to by **output**
 - Detection will of course take place, but that's too late, because the overwritten value has already been used

COMPANY CITY, COUNTRY, DATE 51

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[]
 - Use after free
 - Use after free – Instance of a class
 - Spot the bug
 - Use after free – Dangling pointers
 - Case study - WannaCry
- 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:~/home/cx/c++$ export FOO=() { echo You are hacked };
user@training:~/home/cx/c++$ bash
user@training:~/home/cx/c++$ exit
exit
user@training:~/home/cx/c++$ bash.old
You are hacked
user@training:~/home/cx/c++$ exit
exit
user@training:~/home/cx/c++$
```

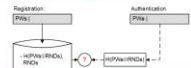
Day 4

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 Math.Abs(int.MinValue)?
 - Signedness bug – spot the bug!
 - Integer truncation – spot the bug!
 - Integer problem – best practices
 - Case study – Android Stagefright
 - Path traversal vulnerability
 - Path traversal – best practices
 - Unvalidated redirects and forwards
 - Log forging
 - Some other typical problems with log files
- 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 .NET
 - 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 .NET
 - bcrypt and scrypt implementations in .NET
 - Case study – the Ashley Madison data breach
 - Typical mistakes in password management
 - Exercise – Hard coded passwords

Password management and storage scademy

- 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:



COMPANY, CITY, COUNTRY, DATE 95

- Accessibility modifiers
 - Accessing private fields with reflection in .NET
 - Exercise Reflection – Accessing private fields with reflection

Common coding errors and vulnerabilities

- Improper error and exception handling
 - Typical problems with error and exception handling
 - Empty catch block
 - Overly broad catch
 - Using multi-catch
 - Catching NullReferenceException
 - Exception handling – spot the bug!
 - Exercise – Error handling
- Code quality problems
 - Dangers arising from poor code quality
 - Serialization – spot the bug!
 - Exercise – Serializable sensitive
 - Class not sealed – object hijacking
 - Exercise – Object hijacking
 - Immutable string – spot the bug!
 - Exercise – Immutable strings
 - Using SecureString.....



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
- .NET secure coding guidelines at MSDN
- .NET secure coding cheat sheets
- Recommended books – C/C++
- Recommended books – .NET and ASP.NET

