Web application security masterclass in C#

CYDCsWeb5d | 5 days | On-site or online | Hands-on

Your Web application written in 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 -1 or -2^31? 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 hands-on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

The curriculum goes through the common Web application security issues following the OWASP Top Ten but goes far beyond it both in coverage and the details.

All this is put in the context of C#, and extended by core programming issues, discussing security pitfalls of the C# language and ASP.NET.

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

So that nothing unexpected happens.

Nothing.

Skills and drills

39 LABS

21 CASE STUDIES

Audience

C# developers working on Web applications

Group size

12 participants

Preparedness

General C# and Web development

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Security testing
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in C#
- Understanding Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of C#
- Going beyond the low hanging fruits
- Input validation approaches and principles
- Managing vulnerabilities in third party components
- Getting familiar with security testing techniques and tools

Table of contents

Day 1

- *Lab – CSRF protection with tokens*

A02 - Cryptographic Failures

- Information exposure
  - Exposure through extracted data and aggregation
  - *Case study – Strava data exposure*
  - System information leakage
    - Leaking system information
  - Information exposure best practices
- Cryptography for developers
  - Cryptography basics
  - Crypto APIs in C#
  - Elementary algorithms
    - Random number generation
    - Pseudo random number generators (PRNGs)
    - Cryptographically secure PRNGs
    - Seeding
    - Weak and strong PRNGs
    - Using random numbers in C#
    - *Lab – Using random numbers in C#*
    - True random number generators (TRNG)
    - *Case study – Equifax credit account freeze*

Day 2

› The OWASP Top Ten 2021

A02 - Cryptographic Failures (continued)

- Cryptography for developers (continued)
  - Elementary algorithms
    - Hashing
    - Hashing basics
    - Hashing in C#
    - *Lab – Hashing in C#*
  - Confidentiality protection (continued)
    - Confidentiality protection
    - Symmetric encryption
    - Block ciphers
    - Modes of operation
    - Modes of operation and IV – best practices
    - Symmetric encryption in C#
    - Symmetric encryption in C# with streams

- *Lab – Symmetric encryption in C#*
- *Case study – Padding oracle used in RCE against Citrix ShareFile*
- Asymmetric encryption
- The RSA algorithm
- RSA in C#
- Combining symmetric and asymmetric algorithms
- Key exchange and agreement
- Key exchange
- Diffie-Hellman key agreement algorithm
- Key exchange pitfalls and best practices

A03 - Injection

- Input validation
  - Input validation principles
  - Denylists and allowlists
- Injection
  - Injection principles
  - Injection attacks
- SQL injection
  - SQL injection basics
  - *Lab – SQL injection*
  - Attack techniques
  - Content-based blind SQL injection
  - Time-based blind SQL injection
  - SQL injection best practices
    - Input validation
    - Parameterized queries
    - *Lab – Using prepared statements*
    - Database defense in depth
    - *Case study – SQL injection leading to RCE in Ivanti Endpoint Manager*
- Code injection
  - OS command injection
    - *Lab – Command injection*
    - OS command injection best practices
    - Avoiding command injection with the right APIs
    - *Lab – Command injection best practices*
    - *Case study – Command injection in Ruckus*
- HTML injection - Cross-site scripting (XSS)
  - Cross-site scripting basics
  - Cross-site scripting types
    - Persistent cross-site scripting
    - Reflected cross-site scripting
    - Client-side (DOM-based) cross-site scripting

- *Lab – Stored XSS*
- *Lab – Reflected XSS*
- *Case study – XSS to RCE in Azure Service Fabric*
- XSS protection best practices
  - Protection principles - escaping
  - XSS protection APIs
  - Further XSS protection techniques
  - *Lab – XSS fix / stored*
  - *Lab – XSS fix / reflected*
  - Client-side protection principles
  - Additional protection layers – defense in depth

Day 3

› The OWASP Top Ten 2021

A03 - Injection (continued)

- Input validation
  - What to validate – the attack surface
  - Where to validate – defense in depth
  - When to validate – validation vs transformations
  - Output sanitization
  - Encoding challenges
  - Unicode challenges
  - Validation with regex
  - Integer handling problems
    - Representing signed numbers
    - Integer visualization
    - Integer overflow
    - *Lab – Integer overflow*
    - Signed / unsigned confusion
    - *Case study – The Stockholm Stock Exchange*
    - *Lab – Signed / unsigned confusion*
    - Integer truncation
    - Best practices
    - Upcasting
    - Precondition testing
    - Postcondition testing
    - Using big integer libraries
    - Integer handling in C#
    - *Lab – Checked arithmetics*
- Unsafe reflection

- Reflection without validation

Lab – Unsafe reflection

- Unsafe native code
  - Native code dependence

Lab – Unsafe native code

  - Best practices for dealing with native code

› The OWASP Top Ten 2021

A04 - Insecure Design

- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder
  - Economy of mechanism
  - Fail-safe defaults
  - Complete mediation
  - Open design
  - Separation of privilege
  - Least privilege
  - Least common mechanism
  - Psychological acceptability
- Client-side security
  - Same Origin Policy
    - Simple request
    - Preflight request
    - Cross-Origin Resource Sharing (CORS)
    - Relaxing the Same Origin Policy
  - Frame sandboxing
    - Cross-Frame Scripting (XFS) attacks

Lab - Clickjacking

    - Clickjacking beyond hijacking a click
    - Clickjacking protection best practices

Lab – Using CSP to prevent clickjacking

  - Some further best practices
    - HTML5 security best practices
    - CSS security best practices
    - Ajax security best practices
  - JSON security
    - JSON validation
    - JSON injection
    - Best practices

Case study – ReactJS vulnerability in HackerOne

  - XML security

- XML validation
- XML injection
- XPath injection
- Blind XPath injection
- XSLT injection
- XML signature
  - XML signature wrapping
  - *Case study – signature wrapping in single sign-on solutions*

Day 4

› The OWASP Top Ten 2021

A05 - Security Misconfiguration

- Configuration principles
- Server misconfiguration
- ASP.NET and IIS configuration best practices
  - ASP.NET configuration
  - IIS configuration
- Cookie security
  - Cookie attributes
- XML entities
  - DTD and the entities
  - Entity expansion
  - External Entity Attack (XXE)
    - File inclusion with external entities
    - Server-Side Request Forgery with external entities
    - *Lab – External entity attack*
    - Preventing XXE
    - *Lab – Prohibiting DTD*
    - *Case study – XXE vulnerability in SharePoint*

A06 - Vulnerable and Outdated Components

- Using vulnerable components
- Assessing the environment
- Hardening
- Untrusted functionality import
- *Case study – The Polyfill.io supply chain attack*
- Vulnerability management
  - Patch management
  - Vulnerability management

- Vulnerability databases
- *Lab – Finding vulnerabilities in third-party components*
- DevOps, the CI / CD build process and Software Composition Analysis
- Dependency checking in C#
- *Lab – Detecting vulnerable components*

A07 - Identification and Authentication Failures

- Authentication
  - Authentication basics
  - Multi-factor authentication (MFA)
  - *Case study – The InfinityGauntlet attack*
- Session management
  - Session management essentials
  - Why do we protect session IDs – Session hijacking
  - Session fixation
- Password management
  - Inbound password management
    - Storing account passwords
    - Password in transit
    - *Lab – Is just hashing passwords enough?*
    - Dictionary attacks and brute forcing
    - Salting
    - Adaptive hash functions for password storage
    - *Lab – Using adaptive hash functions in C#*
    - *Case study – Veeam missing authentication and cleartext password storage*
    - Password policy
    - NIST authenticator requirements for memorized secrets

A08 - Software and Data Integrity Failures

- Integrity protection
  - Message Authentication Code (MAC)
    - Calculating HMAC in C#
    - *Lab – Calculating MAC in C#*
  - Digital signature
    - Digital signature with RSA
    - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
    - Digital signature in C#
    - *Lab – Digital signature with ECDSA in C#*
- Public Key Infrastructure (PKI)
  - Key management challenges
  - Certificates

- Certificates and PKI
- X.509 certificates
- Chain of trust

Day 5

› The OWASP Top Ten 2021

A08 - Software and Data Integrity Failures (continued)

- Subresource integrity
  - Importing JavaScript
  - *Lab – Importing JavaScript*
  - *Case study – The British Airways data breach*
- Insecure deserialization
  - Serialization and deserialization challenges
  - Integrity – deserializing untrusted streams
  - Integrity – deserialization best practices
  - Look ahead deserialization
  - Property Oriented Programming (POP)
    - Creating a POP payload
    - *Lab – Creating a POP payload*
    - *Lab – Using the POP payload*
    - *Case study – Deserialization RCE in Veeam*
    - Summary – POP best practices

A09 - Security Logging and Monitoring Failures

- Logging and monitoring principles
- Insufficient logging
- *Case study – Plaintext passwords at Facebook*
- Logging best practices
- Monitoring best practices

A10 - Server-side Request Forgery (SSRF)

- Server-side Request Forgery (SSRF)
- *Case study – SSRF and the Capital One breach*

Web application security beyond the Top Ten

- Code quality
  - Code quality and security
  - Data handling
  - Function return values

- Unchecked Return Value
- Initialization and cleanup
  - Uninitialized variable
  - Class initialization cycles
  - *Lab – Initialization cycles*
- Unreleased resource
  - Unreleased resource – Database
  - Unreleased resource – Synchronization
  - Unreleased resource – Various
- Language elements
  - Using dangerous language elements
  - Using obsolete language elements
  - Portability flaw
- Memory and pointers
  - Garbage collection
  - Null pointers
  - null dereference
  - null dereference – best practices
  - Using Nullable type parameters
  - Memory leak
  - Unmanaged memory leaks
- Object oriented programming pitfalls
  - Accessibility modifiers
  - Are accessibility modifiers a security feature?
  - Accessibility modifiers – best practices
  - Inheritance and overriding
  - Implementing Equals()
  - Mutability
  - *Lab – Mutable object*
  - Readonly collections

- Denial of service
  - Flooding
  - Resource exhaustion
  - Sustained client engagement
  - Denial of service problems in C#
  - Infinite loop
  - Economic Denial of Sustainability (EDoS)
  - Amplification
    - Some amplification examples
  - Algorithmic complexity issues
    - Regular expression denial of service (ReDoS)
    - *Lab – ReDoS*
    - Dealing with ReDoS

- Hash table collision
- How do hash tables work?
- Hash collision against hash tables

› Security testing

Security testing techniques and tools

- Code analysis
  - Static Application Security Testing (SAST)
  - *Lab – Using static analysis tools*
- Dynamic analysis
  - Security testing at runtime
  - Penetration testing
  - Stress testing
  - Dynamic analysis tools
    - Dynamic Application Security Testing (DAST)
    - Web vulnerability scanners
    - *Lab – Using web vulnerability scanners*
    - Fuzzing

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- .NET and C# resources