Workshop

Secure Coding

in .NET and .NET Core

Rainer Stropek

software architects gmbh

Web http://www.timecockpit.com
Mail rainer@timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

Your Host

Rainer Stropek

Developer, Entrepreneur
Azure MVP, MS Regional Director
IT-Visions

Contact

software architects gmbh
rainer@timecockpit.com
https://rainerstropek.me
Twitter: @rstropek

Organization

Material: https://github.com/rstropek/SecureCodingDotNet

Time Schedule

```
09:00-10:30
10:30-11:00 – Morning Break
11:00-12:30
12:30-13:15 – Lunch Break
13:15-14:45
14:45-15:15 – Afternoon Break
15:15-16:45
```

General Guidelines

How to Write Secure .NET Code

Don't be lazy. Think!

Avoid making stupid mistakes

Do take responsibility

Avoid delegating security measures to others (*DevSecOps*, infrastructure is code)

Don't miss the big picture

Avoid restrictive policies that lead to an overall lower level of security
E.g. security policies that force developers to run VS as administrators
E.g. proxies that lead to plain-text passwords in text files

Do handle PII with special care

Consider encryption whenever you store PII

How to Write Secure .NET Code

Don't build executable code using string concatination
- HTML, JavaScript, SQL, etc.

Don't put secrets in unencrypted text files
- Always use password keepers (for humans and apps)

Do setup CI/CD
- Make sure CI/CD servers can be trusted
- Containers: Automate rebuilding when base images change

Do write automated tests
- Enables quick deployment of new versions and security patches

How to Write Secure .NET Code

Prefer PaaS and/or Serverless

- *Avoid* IaaS
- If IaaS cannot be avoided: Keep your system up to date (immutable infrastructure)

Don't invent your own encryption algorithms

- or write your own implementation of an existing one
- …unless it is your specific expertise

Don't invent your own security protocols

- or write your own implementation of an existing one
- …unless it is your specific expertise

How to Write Secure .NET Code

Do add logging and telemetry collection

- *Consider* using machine learning to detect suspicious user activities
- Be careful when adding PII to logs/telemetry (GDPR)
- *Avoid* writing secrets to logs/telemetry

Do validate user input

- Client *and* server

Cryptography

.NET Cryptography Model

Derived class hierarchy

- Basic algorithm types (e.g. SymmetricAlgorithm, AsymmetricAlgorithm, HashAlgorithm)
- Algorithm classes (e.g. Aes, RC2, ECDiffieHellman)
- Algorithm implementations (e.g. AesManaged, RC2CryptoServiceProvider)

Different implementations

- E.g. Aes → AesCng, AesCryptoServiceProvider, AesManaged
- *CryptoServiceProvider* → wrapper around *Windows Crypto API* (*CAPI*); legacy
- *Managed* → written entirely in managed Code; x-plat, slower, not certified
- *Cng* → *Cryptography Next Generation* (*CNG*), Windows >= Vista, actively developed

See also Recommended algorithms by application

Stream Design

For symmetric and hash algorithms

CryptoStream

- Derived from *System.IO.Stream*

Can be chained

- E.g. hash, followed by encryption

Random Numbers

Pseudo-random numbers

- *System.Random*
- Predictable

Secure random numbers

- *RNGCryptoServiceProvider*

See also https://lowleveldesign.org/2018/08/15/randomness-in-net/

bcrypt.net

https://github.com/BcryptNet/bcrypt.net

- NuGet: *BCrypt.Net-Next*

Port of *jBCrypt* to C#

- Used to safely store passwords

Hashing of passwords with configurable hash complexity

- Compute power necessary for hashing can be set („work factor“)
- Backward/forward compatible because complexity part of resultant hash

Examples

01-Symmetric-Encryption

Encrypt/decrypt data using AES

02-Asymmetric-Encryption

Transfer secret message using RSA and AES

03-Hashing

Compare two files using their SHA512 hash values

04-Signing

Create a signed hash value and verify it

Examples

05-RandomNumbers

Generating strong random numbers with *RNGCryptoServiceProvider*

06-BCrypt

Hash passwords with *BCrypt*

Protecting Secrets

How to protect connection information

General Guidelines

Prefer not storing secrets at all

- Windows Authentication
- Azure: Use AAD authentication with SQL (docs)
- Azure Managed Identities (docs)

Use Connection String Builders instead of string concat

- Sample: *07-ConnectionStringBuilders*
- Don't set *Persist Security Info=True* (default is False) to prevent security-sensitive information to be obtained from a connection

private/protected/internal is no protection

- See also *BindingFlags.NonPublic* (docs)

See also https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/protecting-connection-information

General Guidelines

Recap .NET Core config system

08-RecapNetCoreConfig

Encrypt configuration files

ASP.NET: Use *aspnet_regiis* tool (docs)

ASP.NET Core *Secret Manager* tool (docs)

Stores secrets separated from application code

ASP.NET Core: *09.1-SecretManager*

dotnet user-secrets set "ping" "pong"

→ Show secrets in *%APPDATA%\Microsoft\UserSecrets\...*

See also https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/protecting-connection-information

Environment Variables

Add non-critical defaults to config files (e.g. *appsettings.json*)
- E.g. Connection string pointing to *LocalDB* with integrated security

Override settings in production using environment variables
- Works on Azure
- Works in containers (e.g. *docker run -e*)

Problem: Environment variables are not encrypted
- If machine is compromised, environment variables can be accessed

Solution: Store secrets in vaults like *KeyVault*
- Settings = environment variables only point to vault
- Use integrated security of platform (e.g. Windows, Azure Managed Identity)

Azure Managed Identity

- Automatically managed identities for services in Azure
  - Backed by Azure AD
  - Goal: No credentials on developer workstations or in source control
- Two types: System- or user-assigned
  - More or less control over service principal creation
- Supported by more and more Azure PaaS/Serverless services
  - Docs
  - Here: Focus on KeyVault

See also https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview

KeyVault

Keep secrets, encryption keys, and certs in Azure

- Can be protected by certified HSMs

Monitor access and use

Integrated in various Azure services

- E.g. SSL certs for App Service (docs)
- E.g. *Always Encrypted* in SQL (docs)

Access to KeyVault can be protected by *Managed Identity*

- Sample: *09.2-KeyVaultManagedIdentity*

See also https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview

KeyVault

KeyVault configuration provider

- Merge secrets from KeyVault to configuration settings

Sample: *09.3-KeyVaultConfigurationProvider*

- Adjust KeyVault name in *appsettings.json* if necessary

See also https://docs.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?view=aspnetcore-2.2

Data Protection API

of ASP.NET Core

Overview

How to round-trip trusted state via an untrusted client?

- E.g. Tokens

ASP.NET Core: *Data Protection API*

- Symmetric encryption
- In ASP.NET 4.x: *machineKey* Element in *web.config* (docs)
- Can also be used as a replacement for *machineKey* (docs)

Functionality

- Protect/unprotect strings (e.g. secrets)
- Hash passwords (consider using *BCrypt* instead)
- Algorithms and key management can be configured to specific needs

See also https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/introduction?view=aspnetcore-2.2

Examples

10-NetCoreDataProtection

Use ASP.NET Core Data Protection API to protect/unprotect strings

11-DataProtectionPasswordHash

Calculate password hashes with ASP.NET Core Data Protection API

Configuration

Meaningful defaults for single-server deployments

Key persistance

- Azure Blob Storage
- Manual settings for file system (e.g. UNC path, *Azure Files*)

Key protection

- Protect using KeyVault keys
- Certificate

For details see https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/configuration/overview?view=aspnetcore-2.2

Exceptions and Logging

Exceptions

New in C# 6: *Exception filtering*

Note: Filter runs before *finally*

Example: *12-ExceptionFilters*

Demonstrates order of execution in exceptions with filters

Recap: *IDisposable, using* statement

Necessary?

Docs

See also https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch

Logging – What?

Client requests and server responses

Helpful in reconstructing sequences of events

Account information

E.g. successful and failed authentication attempts, account changes, use of privileges

Usage information

E.g. number of transactions occurring in a certain period, size of transactions

Significant operational actions

E.g. application startup and shutdown, application failures (exceptions), major application configuration changes

See also https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf

Logging Options

ASP.NET Core built-in logging

[Docs]

Log libraries

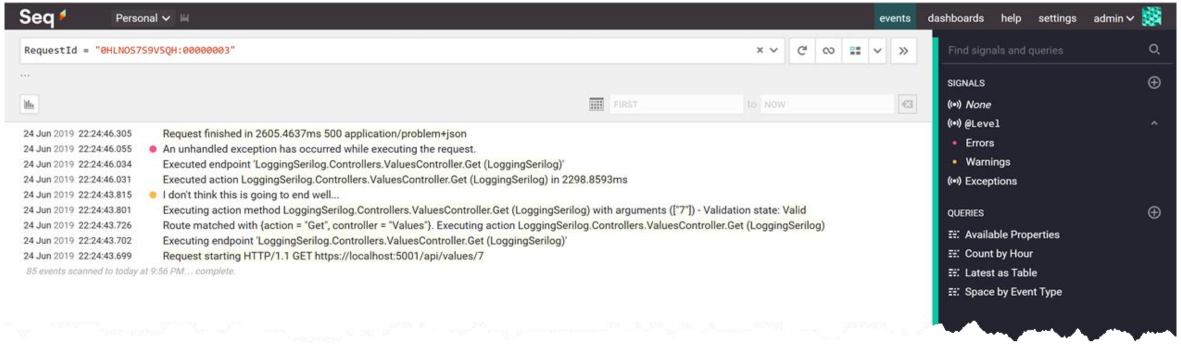
Here: [Serilog]

Many sinks available ([list])

In Azure

[Azure Monitor]

[Azure Application Insights]



```
RequestId = "0HLNOS7S9V5QH:00000003"
```

```
24 Jun 2019 22:24:46.305    Request finished in 2605.4637ms 500 application/problem+json
24 Jun 2019 22:24:46.055  ● An unhandled exception has occurred while executing the request.
24 Jun 2019 22:24:46.034    Executed endpoint 'LoggingSerilog.Controllers.ValuesController.Get (LoggingSerilog)'
24 Jun 2019 22:24:46.031    Executed action LoggingSerilog.Controllers.ValuesController.Get (LoggingSerilog) in 2298.8593ms
24 Jun 2019 22:24:43.815  ● I don't think this is going to end well...
24 Jun 2019 22:24:43.801    Executing action method LoggingSerilog.Controllers.ValuesController.Get (LoggingSerilog) with arguments (["7"]) - Validation state: Valid
24 Jun 2019 22:24:43.726    Route matched with {action = "Get", controller = "Values"}. Executing action LoggingSerilog.Controllers.ValuesController.Get (LoggingSerilog)
24 Jun 2019 22:24:43.702    Executing endpoint 'LoggingSerilog.Controllers.ValuesController.Get (LoggingSerilog)'
24 Jun 2019 22:24:43.699    Request starting HTTP/1.1 GET https://localhost:5001/api/values/7
85 events scanned to today at 9:56 PM... complete.
```

Example

13-LoggingSerilog

- Add logging with *Serilog* to ASP.NET Core app
- Includes log viewing with *Seq*

Database Security

Beware of SQL Injections

Guidelines

Do use parameterized SQL commands

- Sample: *09.2-KeyVaultManagedIdentity*

Avoid concatenating SQL strings with user inputs

Do check user input

- E.g. regular expressions

Do apply the *principle of least privilege* (*PoLP*)

Consider using an OR-Mapper like *Entity Framework*

- Sample: *14-EntityFramework*

Database Encryption

Always Encrypted (docs)

Client encrypts sensitive data inside client applications
Never reveals the encryption keys to the DB

Transparent Data Encryption (docs)

Real-time I/O encryption and decryption of the data and log files
Make stolen physical storage media useless

Encrypted DB Connections (docs)

Use certificate to encrypt DB connections
Automatically done in *Azure SQL Database*

Data Protection

Dynamic Data Masking (docs)

- Limits sensitive data exposure by masking it to non-privileged users

Roslyn Code Analyzer

Overview

Analyzes code for style, quality and maintainability, design, and other issues

Built-in analyzers in VS

- Configure in options and *.editorconfig*

Install additional analyzers with NuGet

- E.g. *FxCopAnalyzers* (docs)

Sample: *15-StaticCodeAnalysis*

- Detection of SQL Injection with code analysis
- Suppression management

See also https://docs.microsoft.com/en-us/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019

Further Readings

Further Readings

.NET

.NET Secure Coding Guidelines
.NET Security Announcements

Web Development Security

ASP.NET Security Announcements
OWASP Security Cheat Sheet for .NET

Database Development

SQL Server Security Guidelines

Workshop

Q&A

Thank your for coming!

Rainer Stropek

software architects gmbh

Mail	rainer@timecockpit.com
Web	http://www.timecockpit.com
Twitter	@rstropek

time cockpit

Saves the day.