Speaker: Barry Dorans

Position: .NET Security PM

.NET Security “Czar”

Barry is the security PM for .NET and .NET Core, which means he gets to look at the framework, tools and template code, searching for potential insecurities and sometimes even fixing them.

@blowdart

Does API reviews

Reviews pen testing

.NET Developers Association – Eastside

Sponsors: Microsoft, TEK systems – help with contracting roles, more on senior side.

DevOps conference – Seattle

chris@kinsman.net

Afterwards after meetup

Microsoft Commons

15253 NE 40th St

Manual Memory Management in .NET framework by Adam Furmaek at May 7th

**Hosting**

Kestrel & IIS

Kestrel HTTPS

Can be config based – obeying environment setting

Selection on subject name & expiry date

IIS, Express, Self Signed Certs

Chrome changed how they validate certs

A fix is coming for IIS & IIS Express

In the meantime – https://gist.github.com/blowdart/....

“A brand new server is a rare thing.”

*Certificate Config*

“everything is in JSON file”

Certificates : {

“HTTPS

Source: Stpre

StoreLocation: :LocalMachine

StoreName: My

Subject: CN=localhost”

“AllowInvalid = True

}

*Kestrel Security Configuration*

Set via Limits property in KestrelOptions

*Version 1*

Keep-alive timeouts

Request header limits

Request/response buffer sizes

Request line size

Request header limits

Request header count limits

“version 1 vulnerabilities”

*Version 2*

Request/Response body timeouts & data rates.

Total client connections.

Internal request draining support. …

*Is Kestrel Edge Ready?*

We still recommend putting Kestrel behind a proxy

However with 2.0 RTM edge will no longer be unsupported

“currently safer to put in a proxy”

“if the load balancer terminates the connection you are good”

“It is slower now because it is now secure in 2.0”

**Authentication**

*Template Support*

No Authentication

Individual user accounts

Work and school accounts

Windows authentication

*Social authentication*

Facebook

Twitter

Google

Microsoft Account

OAuth

Community Driven project for GitHub, DropBox, LinkedIn, Foursquare, Papyal, etc.

https://github.com/aspnet-contrib/AspNet.Security.OAuth.Providers

"samba is a security failure”

Template Authentication

*Individual User Accounts*

Local database

Users can configure EF models for accounts, groups, etc.

Opinionated – we use what we feel is the most secure algorithms for storing data safely

Authentication via OIDC “plug in Facebook”

“primary keys are hard to change”

“hash it and store it”

*Work and School Accounts*

Azure Active Directory and OpenID connect based.

What about ADFS? OpenID Connect needs ADFS 2016.

WS-Fed – finally arrived in February

“you don’t want people crashes when logging in”

“banking, healthcare and government are usually lagging behind”

“you can wire up any number of authentication”

“rival is AuthO, better than Azure AD”

*Windows Authentication*

Needs IIS.

Local domain joined servers.

No automatic impersonation any more. “need call *impersonate* before you can impersonate”

“a secure hashing algorithm is a slow algorithm”

ASP.NET Core 1.0 Changes

*No more custom identity classes*

Everything is ClaimsPrincipal based

*Multiple authentication middleware*

*Cookie authentication*

Finally understands unauthenticated or forbidden

ASP.NET Core 2.0 Changes

*Authentication is now a single service*

No more individual authentication middlewares

Turned on with app.UseAuthentication() in Configure(IApplicationBuilder)

Each authentication type is added and configured in ConfigureServices()

*Active versus Passive*

This sucked

It now sucked less

https://github.com/aspnet/Announcement/issues/232

*TOTP*

ASP.NET Identity now has support for, and defaults to TOTP with authenticator apps

“removed SMS for 2 factor authentication”

“we have built in support for 2FA”

“no QR library”

TOTP Time-based One Token Protocol – RFC for authenticator.

“client side better, server use is memory use for QR -> verification code”

*1.0 v 2.0*

// v1.0

App.UseCookieAuthentication()

..

// v.2.0

ConfigureServices

Cookies

“one active middleware at a time in 2.0”

Setting Defaults

*DefaultScheme*

When everything Is the same handlers

*DefaAuthenticate*

Called to construct the identity for a request

*DefaultSIgnin*

*Called when a user triggers a sign-in.*

*DefaultChallenege*

Called when a challenge is triggerred, for example hitting[Authorize]

An example – Social Auth

Using Twitter

DefaultAuthneticate

Set to

Authenticate/Challengeg/Signin 1.0 v 2.0

// 1.0

Using Microsoft.AspNetCore.Http.Authentication;

Context.Autehtnication.

// 2.00

Using Micros

Context.Authentication.

Do it yourself

Cookie Authentication

Everything ends up in cookie middleware

Cookie middleware encrypts and signs cookies identifying a user principal

Cookie middleware uses data protection

*Caveats*

Once a cookie is dropped it’s the sole source of truth unless you implement a validator

Validators can reject principals, or replace them.

We use a validator to support signing out everywhere on password change in Individual Accounts.

*Implementing a Validator*

Public class CookieValidator

Public static async Task Validate Async(

// if (refreshNeeded)

Else

*Wiring up a validator*

Services.AddAuthentication(options =?

“five refreshes, it will log out the person”

“bearers token are more secure than cookies, however, the only easy way to do that is JavaScript, protect by cross-site protections, anti-forgery tokens for API calls”

public class AccountController : Controller

{

Static int \_issueCount = 0;

Public async Task<IActionResult> Login(string returnUrl = null)

{

const string Issuer = <https://contoso.com>;

var claims = new List<Claim>();

claims.Add(new Claim(ClaimTypes.Name, “barry #” +\_

}

}

*Limiting by Scheme*

[Authorize(ActivateAuthenticationSchemes = “Bearer”)]

Public class ApiController : Controller

“separate an APIs out an put it in another host”

“don’t do it based on path”

Individual user account

*Local database*

ASP.NET Identity is your database, models and everything around it

*Azure B2c*

Security is scary – let someone do it

*I don’t like Azure*

Use another 3rd part option: IdentityServer, ASOS, OpenIddict

Identity as a Service – 2.2 (?)

*Why OIDC?*

Standards Based

Decouples app code from authentication code

**Authorization**

*Goal*

No need to write your own authentication attribute

Removal of hard coded rules in attributes

*Features*

Code based authorization, policies, requirements, and requirement handlers

Resource based authorization

DI based

Authentication scheme filtering

Custom policy providers

(Roles supported for back compat. Please avoid it unless you’re using Integrate Auth.)

*Workshop*

https://github.com/blowdart/AspNetAuthorizationWorkshop

“replaced authorization by a call for authorization service”

*Requirements*

Using Microsf.aspNetCore.Au

Namespace AuthorizationLab

Public class Office Entry Requirement

*Handlers*

Using

Namespace

Public class HasBadgeHandler -: AuthorizationHanlder<OfficeEntryRequirement

{

Protected override Task HandleRequirementAsync(

AuthorizationHanlderContext context,

OfficeEntryRequirement requirement)

{

If

Return

}

// i.e. temporary access badges. Picture of space needles goes away after by 12 hours

*Policies*

Public void ConfigureServices()

Services.AddAuthorizatoin(options =>

{

Options.AddPolicy(“BuildingEntry”,

Policy

}

*Authorizing*

[Authorize(Policy = “BuildingEntry”)]

Public class BuildingController : Contorller

Public class IActionResult ()

View

“Multiple handlers, only 1 handler has to pass”

“no way to know what handler passed”

“no changing users inside a handler”

“when user has been authenticating, don’t mutate the user”

“cookie or the other goes to claims transformation service”

**Authorization Demo**

“dip tomatoes on fries, click “dip” “

“it then sents the log in”

“user Hugh Grant”

“it then successfully dips the tomato sauce”

“change condiment”

“doesn’t like it”

“user TinTin”

“change condiment to mayonnaise”

“can’t use tomato to dip”

“change to AmericanTourist”  
“dip to mayonaisse, can’t”

“dip to chili, success!”

Policy.RequireAuthenticatedUser

Add AuthorizationHandler

Look at Condiment database

“Handler will check the authenticated user”

“Will then check country to check against type of dip”

“If there are multiple handlers, 1 can succeed”

“if not success, forbidden”

“all requirements will be validated”

What should a handler return?

*Successful Evaluation*

Call context.Succeed(requirement);

*Unsuccessful Evaluation*

Do nothing.

*Horrific Circumstances*

The user has just been fired, but not everything has been updtated

My database server is on fire

Call context.Fail()

Imperative Checks

*Evaluate your policies in code*

Inject IAuthorizationService into your controller

Call AuthorizeAsync();

If failed return new ForbiddenResult();

“had someone worked on a policy, I have been credited, no you should be credited, no we don’t want to be accountable/asked, lol”

Things to Remember

*Register your handlers in DI*

As they’re in DI you can take repos.

Remember your DI scope if using EF

*Check your claims issuer*

Resource Based Authorization

*Acting on a resource or model*

For example check that the current user owns the resource requested

*Operation Based*

Resource Authorization Handlers

Public class DoesCurrentUserOwnHanlder

“checks for document owner, as another parameter”

Authorize Everywhere

Call PolicyBuilder

.Require

.Build

Add Bearer

.RequireAuthenticatedUser()

Adjusting your views

*Use Razor’s DI system*

@inject IAuthorizationService AuthorizationService

Put in \_ViewImports.cshtml to make globally available

Call Authorization.AuthorizeAsync()

*Imperative checks in views*

Remember to duplicated the checks in your controllers

Custom policy providers

*Extending policy resolution*

Takes a single string argument and returns a policy based upon it

“stringification”

**Data Protection***One stop shop for encryption*

No more machine key

Aimed at ephemeral data

Removes the ability to shoot yourself in the foot

Supports key rotation automatically

Provides isolation for applications automatically

Provides isolation based on purposes automatically

Attempts to figure out where to store keys based on app platform

Easy to write new key stores to match your customers’ environments

Custom algorithms supported (for Russia)

“the docs say, these are the things you care about losing”

“we make opinionated decisions for you (isolation purposes, encryptions)”

“we know what to do in Azure, we don’t know what AWS wants”

“server log, we don’t know what you want to do so we throw away your keys”  
“application 1 cannot decrypt application 2”

“key vault for Azure”

“read all keys or write new keys”

“certain governments require certain algorithms”

“US pushes for encryption backdoors”

Configuration

*Algorithms*

512bit master key

Rolled every 90 days

Derived keys based on purpose and every payload

AES-256 CBC for encryption

HMACSHA256 for authenticity

*Key Stores*

Azure Web Applications – Special synced folder, unencrypted

IIS with no user profile – registry, with machine DPAPI & worker process ACLed (with configuration)

IIS with user profile - %AppData%, user DPAPI

In memory, discarded

*Protection*

DPAI, DPAPI NG (with AD), X509 Certificate, Azure KeyVault, Plain Text

“get keys-> validate->keys refreshes every 24 hours”

“worst possible certificate – plain text”

Using Data Protection

*Manually*

*ASP.NET Core*

new EphemeralDataProtectionProvider()

CreateProtector

Purpose Strings

*Their purpose*

Provides isolation within an application

Configring Data Protection

*Configuration Points*

Key stores

Encryption

Key expiry policies

*Sharing Key Stores*

Apps are isolated by default

Forward compatibility package for .NET 4.5.2 (4.6.1 in Core 2.0)

**.PersistKeysToFileSystem**

Implementing a key store

New in 2.0

*X509 Certificate protection provider*

*KeyVault protection provider*

**Backwards Compatibility**

Sharing with ASP.NET 4.5.2

**General Enhancement**

*Encoders*

All encoders are safe list based

Encoders are in DI and can be injected into DI sourced classes

*CSTF*

Can now be automatic

Tokens can be accessed in Razor for use in Javascript

Tokens can be sent in a header, defaults to RequestVerificationToken

RazorPages inserts and validates CSRF tokens by default *CORS middleware*

*Environments*

Controlled by ASPNETCORE\_ENVIRONMENT environment variable. Defaults to Release

Enabling automatic CSRF Validation

Ignore tokens

**Getting CSRF tokens in biews**

@using Mircosoft.AspNetCore.AntiForgery

@inject IAntiforgery Antiforgery

// GetAndStore, or Get

@ {var antiforgeryTokens = Antiforgery.GetAndStoreTokens(Context); }

<script>

Var anti = ‘@antiforgeryTokens.RequestToken

</script>

Secrets

*Environment based*

Environment variables cant be checked into GitHub

Can develop custom settings providers

Azure KeyVault support

*Secret manager*

Developments use only

“if you pull an identity it pulls a NuGet package”