Microsoft

Security in ASP.NET Core 2.0

Barry Dorrans
.NET Security PM
https://idunno.org

Who am I

.NET Security “Czar”

Which means I annoy everyone about coding securely

And I get to deal with it when I miss things

I also ramble and swear on twitter in a non-work approved manner
@blowdart

Hosting

Kestrel & IIS

Kestrel HTTPS

Can be config based – obeying environment setting

Selection on subject name & expiry date

IIS, 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/1cb907b68ed56bcf8498c16faff4221c

Certificate Config

```
{
  "Certificates": {
    "HTTPS": {
      "Source": "Store",
      "StoreLocation": "LocalMachine",
      "StoreName": "My",
      "Subject": "CN=localhost",
      "AllowInvalid": true
    }
  }
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();
```

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 2 additions

Request/Response body timeouts & data rates

Total client connections

Internal request draining support

(if you configure a data rate without an upper timeout we ensure the request drains)

So … Kestrel on the Edge

Is Kestrel Edge Ready?

¯_(ツ)_/¯

We still recommend putting Kestrel behind a proxy

However with 2.0 RTM edge use will no longer be unsupported

Authentication

Authentication Options in ASP.NET Core

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

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

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

Windows Authentication

Needs IIS.

Local domain joined servers.

No automatic impersonation any more.

ASP.NET Core 1.0 Changes

No more custom identity classes

Everything is ClaimsPrincipal based

Multiple authentication middleware

They don’t have to run automatically.

Authorization can cherry pick which authentication scheme to use.

Only one middleware can run automatically.

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

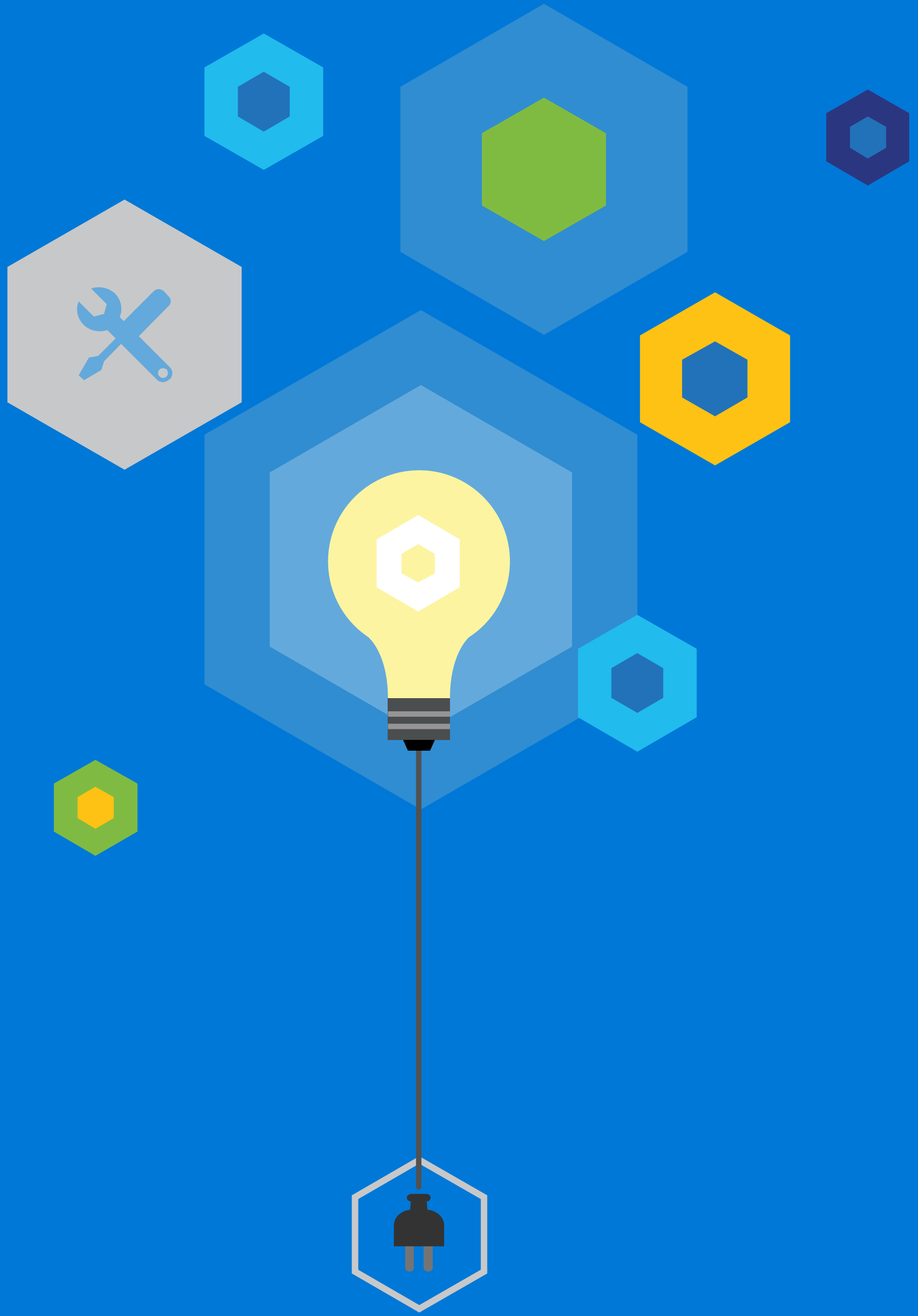
It now sucks less

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

TOTP

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

TOTP



1.0 v 2.0

```
// v1.0

app.UseCookieAuthentication(new CookieAuthenticationOptions()
{
        AccessDeniedPath = "/Account/Forbidden/",
        LoginPath = "/Account/Unauthorized/",
        AuthenticationScheme = CookieAuthenticationDefaults.AuthenticationScheme,
        AutomaticAuthenticate = true,
        AutomaticChallenge = true
});
```

1.0 v 2.0

```
// In ConfigureServices(IServiceCollection services)
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;
})
.AddCookie(options =>
{
        options.LoginPath = new PathString("/Account/Login/");
        options.AccessDeniedPath = new PathString("/Account/Forbidden/");
});

// In Configure(IApplicationBuilder app, IHostingEnvironment env)
app.UseAuthentication();
```

Setting Defaults

DefaultScheme

When everything is the same handler

DefaultAuthenticate

Called to construct the identity for a request

DefaultSignin

Called when a user triggers a sign-in.

DefaultChallenge

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

An example – Social Auth

Using Twitter.

DefaultAuthenticate

Set to cookie to rehydrate the identity set during Signin on each request.

DefaultSignin

Set to cookie, because the auth flow will call Signin, and you want your identity persisted.

DefaultChallenge

Set to Twitter, because you want the actual challenge to bounce to twitter to login there.

Authenticate/Challenge/Signin 1.0 v 2.0

```
// 1.0
using Microsoft.AspNetCore.Http.Authentication;
context.Authentication.
   Authenticate|Challenge|SignInAsync("scheme");


// 2.0
using Microsoft.AspNetCore.Authentication;
context.Authenticate|Challenge|SignInAsync("scheme");
```

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 - Encryption keys must be synced between servers

You can use the cookie middleware to serialize your own `ClaimsIdentity` and authenticate subsequent requests

Signing in and out in code

```
// Signin & Signout
await HttpContext.SignInAsync("SchemeName", principal);
await HttpContext.SignOutAsync("SchemeName");
```

Do it yourself

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.

You can use this to update claims when the backend changes.

Validators on raw Cookie Middleware run on every request.

Individual Accounts throttle the validator to 30 minutes
(configurable with `IdentityOptions.SecurityStampInterval`)

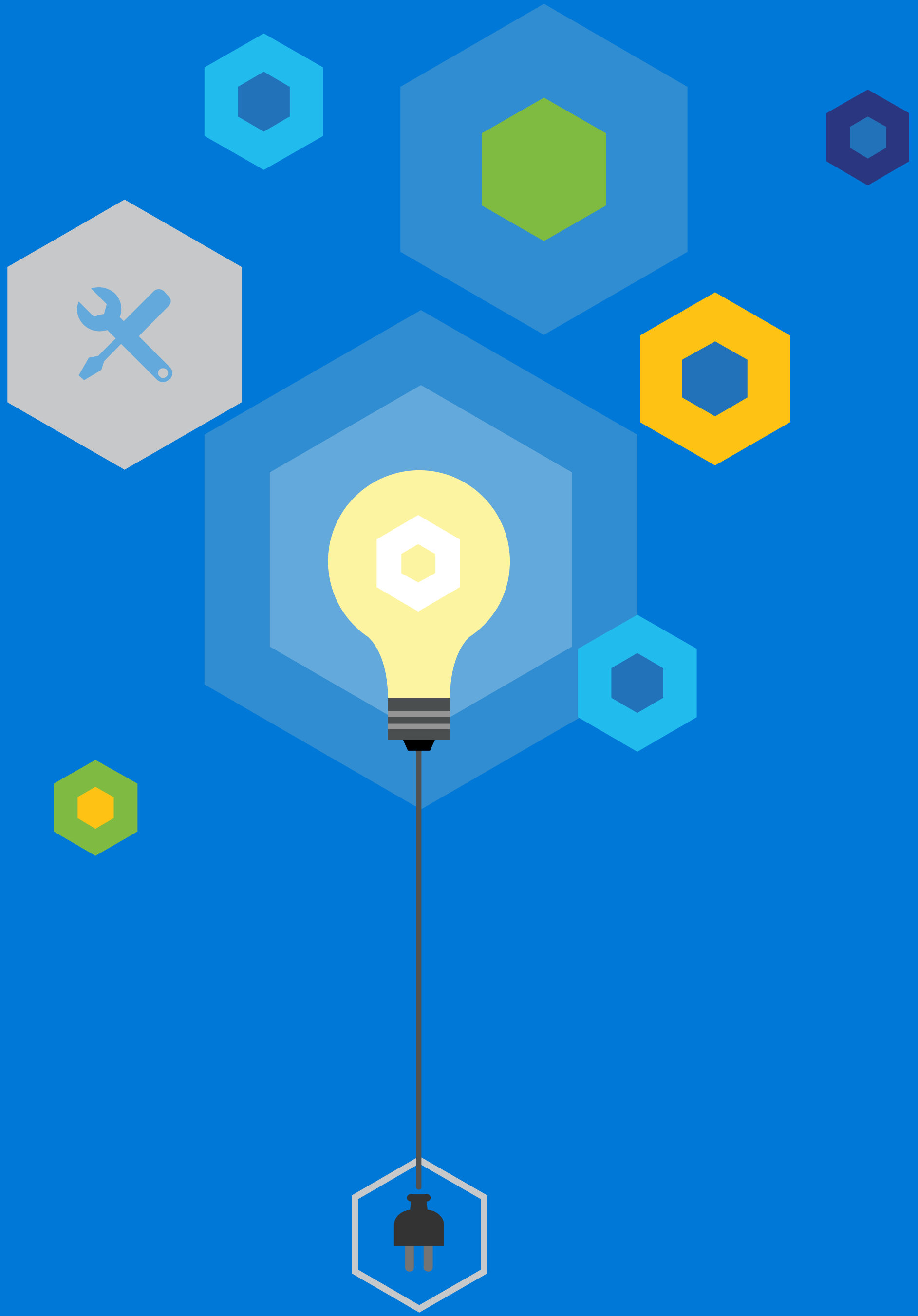
Implementing a validator

```
public class CookieValidator
{
    public static async Task ValidateAsync(CookieValidatePrincipalContext context)
    {
        // Do my validation/refresh checks.
        if (refreshNeeded)
        {
            context.ReplacePrincipal(newPrincipal);
            context.ShouldRenew = true;
        }
        else if (validationFailed)
        {
            context.RejectPrincipal();
            await context.HttpContext.SignOutAsync(
                "SchemeName");
        }
    }
}
```

Wiring up a validator

```
services.AddAuthentication(options =>
{
    // …
})
.AddCookie(options =>
{
    options.LoginPath = new PathString("/Account/Login/");
    options.AccessDeniedPath = new PathString("/Account/Forbidden/");
    options.Events = new CookieAuthenticationEvents()
    {
        OnValidatePrincipal = CookieValidator.ValidateAsync
    };
});
```

Cookie & Validator Demo



Limiting by scheme

```
[Authorize(ActiveAuthenticationSchemes = "Bearer")]
public class ApiController : Controller
```

Individual User Accounts

Local Database

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

Azure B2C

Security is scary – let someone else do it

I don’t like Azure

Use another 3rd party option

- IdentityServer
- ASOS
- OpenIddict

Identity as a Service – 2.2(?)

Why OIDC?

Standards Based

Decouples app code from authentication code

Single authentication flow for all providers, internal and external

Identity can be a separate app

Enables mobile & native applications

Uplift to Azure B2C

Replaceability

IDaaS only supports Authorization & Hybrid Code Flows (Implicit is coming for SPA apps)

Need more than we give you? Swap out for IdentityServer, OpenIddict, ASOS

Authorization

Authorization

Goal

No need to write your own authorization 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 Integrated Auth.)

Workshop

https://github.com/blowdart/AspNetAuthorizationWorkshop

Requirements

```
using Microsoft.AspNetCore.Authorization;

namespace AuthorizationLab
{
    public class OfficeEntryRequirement : IAuthorizationRequirement
    {
    }
}
```

Handlers

```
using Microsoft.AspNetCore.Authorization;

namespace AuthorizationLab
{
    public class HasBadgeHandler : AuthorizationHandler<OfficeEntryRequirement>
    {
        protected override Task HandleRequirementAsync(
          AuthorizationHandlerContext context,
          OfficeEntryRequirement requirement)
        {
            if (!context.User.HasClaim(c => c.Type == "BadgeNumber" &&
                                        c.Issuer == "https://contoso.com"))
            {
                return Task.FromResult(0);
            }

            context.Succeed(requirement);
            return Task.FromResult(0);
        }
    }
}
```

Policies

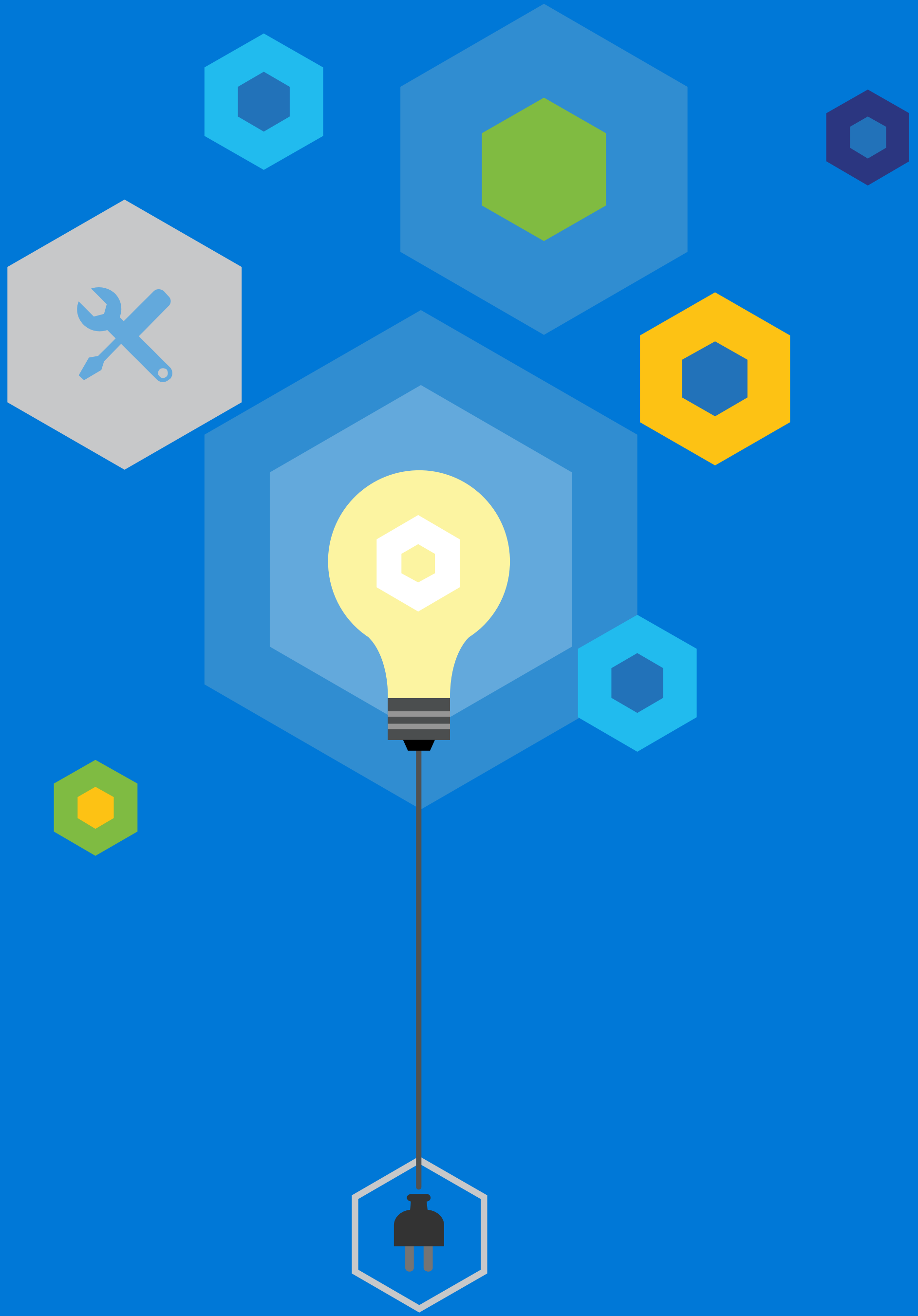
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthorization(options =>
    {
        options.AddPolicy("BuildingEntry",
                          policy => policy.Requirements.Add(
                                        new OfficeEntryRequirement()));
    });

    // Register the handler for the requirement.
    services.AddSingleton<IAuthorizationHandler, HasBadgeHandler>();
}
```

Authorizing

```
[Authorize(Policy = "BuildingEntry")]
public class BuildingController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Authorization Demo



Policies, Requirements & Handlers

Policies

Made up of one or more requirements

All requirements must succeed for the policy to pass

All requirements are evaluated, even if previous ones fail

Requirements

`IAuthorizationRequirement`

Marker interface, no data is required

Can have multiple handlers

Handlers

`AuthorizationHandler<IAuthorizationRequirement>`

Must be registered in DI system

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 updated

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();`

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

Base class provided – `OperationAuthorizationRequirement`

Remember multiple handlers – a common administrator handler could be combined with resource ownership checks

Resource Authorization Handlers

```
using Microsoft.AspNetCore.Authorization;

namespace AuthorizationLab
{
    public class DoesCurrentUserOwnHandler :
        AuthorizationHandler<OperationAuthorizationRequirement, Document>
    {
        protected override Task HandleRequirementAsync(
          AuthorizationHandlerContext context,
          OperationAuthorizationRequirement requirement,
          Document resource)
        {
            if (document != null && document.Owner == context.Identity.Name)
            {
                context.Succeed(requirement);
                return Task.CompletedTask;
            }
        }
    }
}
```

Authorize everywhere

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(config =>
    {
        var policy = new AuthorizationPolicyBuilder()
                         .RequireAuthenticatedUser()
                         .Build();
        config.Filters.Add(new AuthorizeFilter(policy));
    });
}
```

Limiting by scheme in policy

```
options.AddPolicy("Api", policy =>
{
    policy.AuthenticationSchemes.Add("Bearer");
    policy.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 duplicate the checks in your controllers.

Custom policy providers

Extending policy resolution

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

Can be used to support old style custom attribute arguments via “stringification”.

Data Protection

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)

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

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

Using Data Protection

Manually

Create a `IDataProtectionProvider`.

Create a `DataProtector` with a purpose from the `IDataProtectionProvider`.

ASP.NET Core

`services.AddDataProtection();`

Take `IDataProtectionProvider` in your controller constructors.

Use

Call `protector.Protect();`

Call `protecter.Unprotect();`

Using Data Protection

```
var dataProtectionProvider = new EphemeralDataProtectionProvider();
var protector = dataProtectionProvider.CreateProtector(purpose);

Console.Write("Enter input: ");
string input = Console.ReadLine();

string protectedPayload = protector.Protect(input);
Console.WriteLine($"Protect returned: {protectedPayload}");

string unprotectedPayload = protector.Unprotect(protectedPayload);
Console.WriteLine($"Unprotect returned: {unprotectedPayload}");
```

Purpose Strings

Their purpose

Provides isolation within an application

Used to derive keys from the master key

`CreateProtector("Authentication")` cannot unprotect `CreateProtector("Data")`

Caveats

Do not use user input as the sole source of a purpose

A good purpose would be something like `Contoso.Component.1.0`

Configuring Data Protection

Configuration Points

Key stores

Encryption

Key expiry policy

Sharing Key Stores.

Apps are isolated by default

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

Configuring Data Protection

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDataProtection()
        .PersistKeysToFileSystem(
                new DirectoryInfo(@"\\server\share\directory\"))
        .ProtectKeysWithCertificate("thumbprint")
        .SetApplicationName("my application");
}
```

Implementing a key store

IXmlRepository

- `GetAllElements`
- `StoreElement`

New in 2.0

X509 Certificate protection provider

Before was only when targeting desktop

KeyVault protection provider

Acts like the X509 protection provider

You still need a backing store

Back Compatibility

Sharing with ASP.NET 4.5.2

Data Protection

Forward compatibility package

Enables ASP.NET 4.5.1 to use data protection

Cookie sharing

Between Katana based applications and ASP.NET Core

Replacing machine key

Needs configuration to share the key ring and set a shared application name.

Will require 4.6.2 to talk to .NET Core 2.0

Updated docs at `https://github.com/blowdart/idunno.CookieSharing`

ASP.NET Security Enhancements

ASP.NET Enhancements

Encoders

All encoders are safe list based

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

CSRF

CSRF can now be automatic

CSRF tokens can be accessed in Razor for use in JavaScript

CSRF 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

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
        options.Filters.Add(
        new AutoValidateAntiforgeryTokenAttribute()));
}

[IgnoreAntiforgeryToken]
public MyController : Controller
{
}
```

Getting CSRF tokens in views

```
@using Microsoft.AspNetCore.Antiforgery
@inject IAntiforgery Antiforgery

// GetAndStore, or Get
@{ var antiforgeryTokens =
       Antiforgery.GetAndStoreTokens(Context); }

<script>
var antiForgeryToken = '@antiforgeryTokens.RequestToken';
</script>
```

Secrets

Environment based

Environment variables can’t be checked into GitHub

Can develop custom settings providers

Azure KeyVault support

Secret Manager

Development use only

Secrets held outside of source directory in user profile directory

Q&A

Microsoft