

SECURING ASP.NET CORE APPLICATIONS

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
Consultant, Teacher, Writer

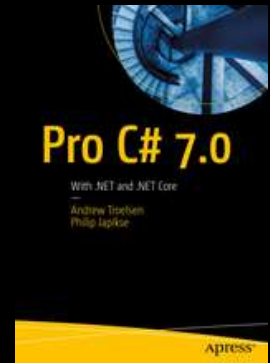


<https://github.com/skimedic/presentations>

All slides copyright Philip Japikse <http://www.skimedic.com>

Phil.About()

- Consultant, Coach, Author, Teacher
 - Lynda.com (<http://bit.ly/skimedicylndacourses>)
 - Apress.com (<http://bit.ly/apressbooks>)
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
 - <http://www.dayofagile.org>
- President, Cincinnati .NET User's Group



WHAT DOES IT MEAN TO “SECURE”?

- More than just “logging in”
- Authentication
- Authorization
- Transport Layer Security
- Cross Origin Resource Sharing (CORS)
- Cross Site Scripting (XSS)
- User and access control management

TRANSPORT LAYER SECURITY (TLS)

- Provides communications security
 - SSL was proven to be easy to hack
 - SSL is now prohibited by the Internet Engineering Task Force (IETF),
- TLS aims to provide privacy and data integrity between two communicating computer applications

TLS SECURE CONNECTION PROPERTIES (MUST HAVE 1+)

- Symmetric cryptography encrypts the data transmitted
- The identity of the communicating parties can be *authenticated* using public-key cryptography.
- Each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.

CROSS ORIGIN RESOURCE SHARING (CORS)

- CORS defines a way in which a browser and server can interact to determine whether or not it is safe to allow request from a different domain.
 - It is more secure than simply allowing all cross-origin requests.
- It describes new HTTP headers which provide browsers and servers a way to request remote URLs only when they have permission.
 - Built in to all modern browsers
- Simple CORS
 - GET/POST, form encoded, no additional header
 - Sends Origin header in request, expects Access-Control-Allow-Origin in response

DEALING WITH CORS

- Most CORS sends “preflight” OPTIONS request specifying what is being requested (Verb, headers, cookies, etc)
- Destination server decides who gets in
- Have to populate appropriate headers in your \$http service calls
- Automatic with Angular \$http service with right configuration
- Configurable with ASP.NET Core Middleware

CROSS SITE REQUEST FORGERY (CSRF/XSRF)

- Attack where unauthorized commands are executed unwilling by user that the web application (browser) trusts.
- Commonly involves the following:
 - Sites that rely on user's identity
 - Exploits that sites trust
 - Tricks the browser into sending HTTP requests to target site
- Typically target state change attacks since the response can't be captured
- Can be executed through Image tags, JS Ajax Requests, hidden forms, etc.

CROSS SITE SCRIPTING (XSS)

- XSS are attacks where malicious scripts are injected into trusted web sites.
- Can be used to bypass CORS rules or other access controls
- Can access cookies, session tokens, or other sensitive information
- Account for roughly 84% of security vulnerabilities documented by Symantec in 2007

PROTOCOLS

➤ OAuth2

- Just about authorization
- Issued access token after user is authenticated “somehow”
- Includes provisions for user consent

➤ OpenID Connect

- Builds on OAuth2
- Just about authentication
- Issued id token after presenting valid credentials

TERMINOLOGY

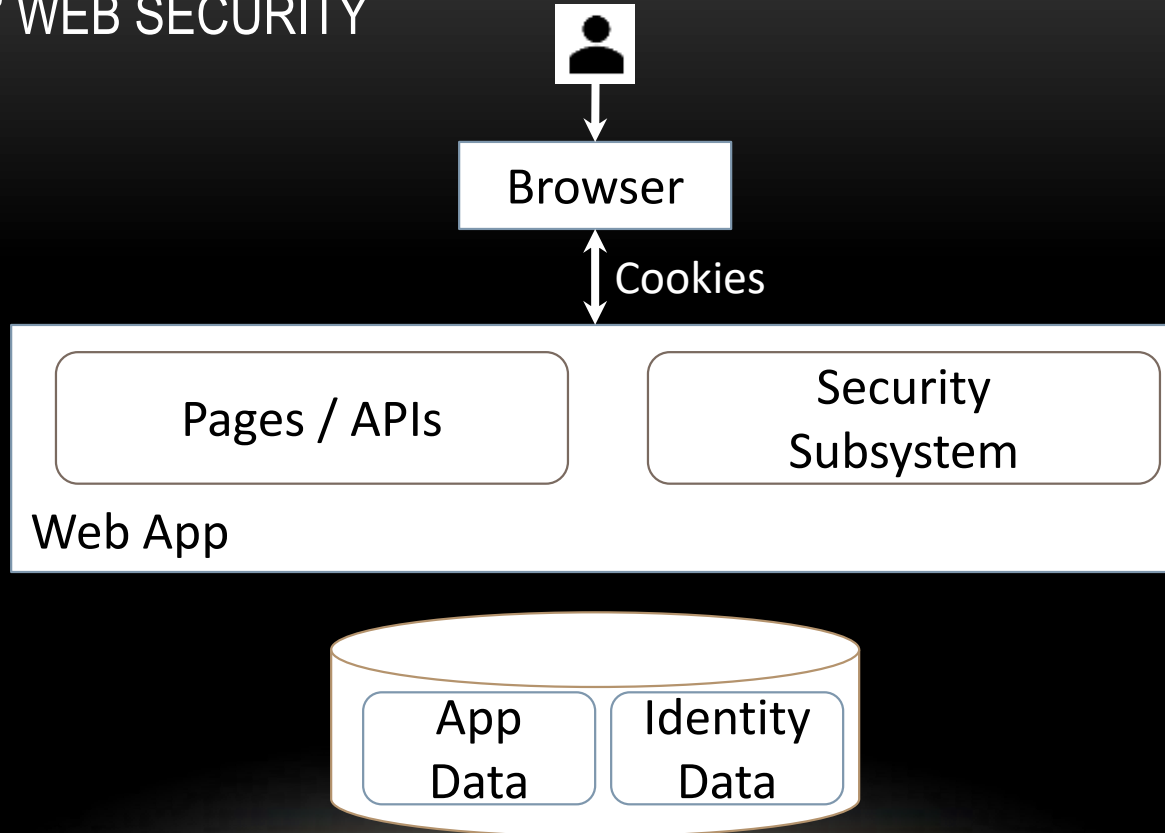
- Client – application requesting access to a Resource
- Resource / Relying Party – a secured API/app that Client wants to call
- Resource Owner – end user using the Client
- Scope – a named resource that authorization is needed for
- Identity Provider (IdP) / Security Token Service (STS) / SSO server / Authentication Server / Authorization Server
 - App that manages identities, authenticates users, returns ID and Access tokens for use by Client
 - IdentityServer, Azure AD, ADFS, Domain Controller, Auth0 server
- JWT – “jwt” – token format used for OpenID Connect and OAuth2

ASP.NET CORE IDENTITY

ASP.NET (CORE) IDENTITY

- Handles creation and management of identities, roles, and claims
- Handles password hashing / creation, crypto protocols, etc.
- Supports Oauth providers and Security Token Service (STS) providers (like Identity Server)

“CLASSIC” WEB SECURITY



AUTHENTICATION OPTIONS

- Windows authentication
- Basic authentication
 - Locally stored user accounts
 - Clouds (e.g. Azure AD) user accounts
 - Work or School accounts
- Cookie-based authentication with host site
- Token-based authentication (STS)

ASP.NET CORE IDENTITY PACKAGES

- `Microsoft.AspNetCore.Identity.EntityFrameworkCore`
- `Microsoft.EntityFrameworkCore.SqlServer`
 - Can use `.InMemory` for testing
- `Microsoft.AspNetCore.Authentication.Cookies`
 - Enables cookie based authentication

IDENTITY DATABASE TABLES

- Users – Registered users of your application
- User Claims – Claims for individual users
- User Logins – External logins for users
- Roles – Authorization Groups
- Role Claims – Claims for authorization groups

IDENTITY OPTIONS

- Configured in Startup/ConfigureServices
- Configures:
 - Password requirements
 - Lockout options (check AccountController.Login if lockout is enabled)
 - UserName/Email restrictions
 - Signin restrictions
 - ClaimsIdentity options settings
 - Tokens provider options (password reset, email confirmation, change email)

COOKIE SETTINGS

- Configured in Startup/ConfigureServices
- Options:
 - LoginPath (Account/Login)
 - AccessDeniedPath (Account/AccessDenied)
 - ExpireTimeSpan (defaults to 14 days)
 - SlidingExpiration (defaults to true)

CHANGE PRIMARY KEYS

CONVERT PRIMARY KEY TO INT FROM STRING

- Modify ApplicationUser
- Create ApplicationRole
- Update Startup
- Update UrlHelperExtensions, Account/ConfirmEmail, Account/ResetPassword, MC.LinkLoginCallback,, AC.ConfirmEmail
- Delete Migrations, add new

```
public class ApplicationUser : IdentityUser<int> {}  
public class ApplicationRole : IdentityRole<int> {}  
  
services.AddIdentity<ApplicationUser, ApplicationRole>()  
    .AddEntityFrameworkStores<ApplicationDbContext>()  
    .AddDefaultTokenProviders();
```

EXTEND IDENTITY TABLES

EXTENDING IDENTITY TABLES

- ApplicationUser and ApplicationRole wer changed to use <int>
 - This cascaded to all other Identity Tables
- Identity tables have FK relationships defined in database but no navigation properties
- Adding navigation properties through the fluent API allows for C# traversal of Identity models

EMAIL CONFIRMATION AND PASSWORD RESET

CONFIGURE EMAIL PROVIDER

- Default template pretends to send email
- Add MailKit to send email
- Use Secrets Manager Tool to save secrets
 - VS or dotnet user-secrets set <key> <value>
 - Stored in
 - %appdata%/Microsoft/UserSecrets/<WebAppName-userSecretsId>
- Use Azure KeyVault in production

ACCOUNT CONFIRMATION AND PASSWORD RECOVERY

- Accounts can be required to have confirmed email address
 - Users can register but not login until email is validated
 - Immediate logon can be disabled in Account/Register by commenting out:

```
//await _signInManager.SignInAsync(user, isPersistent: false);
```

- Be careful of changing this setting after site has users
 - Set all existing users as confirmed

TWO FACTOR AUTHENTICATION

CONFIGURE 2 FACTOR AUTHENTICATION WITH QR CODE

- Download qrcode.js and qrcode.min.js from <https://davidshimjs.github.io/qrcodejs/>
- Add the following to the scripts section of `Manage\EnableAuthenticator.cshtml`

```
<script src="~/js/qrcode/qrcode.js" asp-append-version="true"></script>
<script type="text/javascript">
  new QRCode(document.getElementById("qrCode"),
  {
    text: "@Html.Raw(Model.AuthenticatorUri)",
    width: 150,
    height: 150
  });
</script>
```

OAuth AUTHENTICATION

REQUIRE SSL

➤ RequireHttpsAttribute - Globally/Controller/Actions

```
services.Configure<MvcOptions>(options =>  
{  
    options.Filters.Add(new RequireHttpsAttribute());  
});
```

➤ URL Rewriting Middleware

```
//Use the global filter instead  
var options = new RewriteOptions().AddRedirectToHttps();  
app.UseRewriter(options);
```

➤ Kestrel and SSL is convoluted. See <https://www.lynda.com/ASP-NET-tutorials/ASP-NET-Core-New-Features/656815-2.html> for full details.

SUPPORTING PROCESSES

- Create Privacy Policy URL
- Create Terms of Service URL
- SignInManager loads login button for each registered OAuth provider
- External Logins are stored in `AspNetUserLogins`
 - Still need to register in `AspNetUsers`

FACEBOOK

- Create/Register your app with <https://developers.facebook.com/apps/>
- Create a new app id - choose Facebook Login
- Under settings, select
 - Client OAuth Login, Web OAuth Login, Enforce HTTPS, Strict Mode
 - Enter `https://localhost:<portnumber>/signin-facebook` for redirect URL
- From the Settings menu, note the App ID and Client Secret
- Add to secrets.json

```
dotnet user-secrets set Authentication:Facebook:Appid <app-id>  
dotnet user-secrets set Authentication:Facebook:AppSecret <app-secret>
```


FINISH SETUP

➤ Update Startup/Configure:

```
services.AddAuthentication().AddFacebook(facebookOptions =>
{
    facebookOptions.AppId = Configuration["Authentication:Facebook:AppId"];
    facebookOptions.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
});
```

➤ Login page now has a button for Facebook login

TWITTER

- Create/Register your app with <https://apps.twitter.com/>
- Create a new app with
 - App Name, Description,
 - Website: Valid URL (e.g. <https://www.skimedic.com>)
 - Callback Url: `https://localhost:<portnumber>/signin-twitter`
 - Can only get email with PrivacyPolicy and TermsOfService
- From the Settings menu, note the Consumer Key and Secret
 - Add to secrets.json

```
dotnet user-secrets set Authentication:Twitter:ConsumerKey <key>  
dotnet user-secrets set Authentication:Twitter:ConsumerSecret <secret>
```

FINISH SETUP

- Add Microsoft.AspNetCore.Authentication.Twitter package
- Update Startup/Configure:

```
services.AddAuthentication().AddTwitter(twitterOptions =>
{
    twitterOptions.ConsumerKey = Configuration["Authentication:Twitter:ConsumerKey"];
    twitterOptions.ConsumerSecret = Configuration["Authentication:Twitter:ConsumerSecret"];
    twitterOptions.RetrieveUserDetails = true;
});
```

- Add Terms of Service and Privacy Policy URLs to access user's email address
 - Must be valid URL Format (i.e. no port number)
- Login page now has a button for Twitter login

GOOGLE

- Create/Register your app with <https://console.developers.google.com/projectselector/apis/library>
- Select Google+ API click Enable
- Select Create Credentials, calling from Web Server, select User Data
- Create OAuth2.0 client Id and call back URL
 - <https://localhost:<portnumber>/signin-google>
- Setup OAuth Consent Screen
- Download Credentials
 - Add to secrets.json

```
dotnet user-secrets set Authentication:Google:ClientId <key>  
dotnet user-secrets set Authentication:Google:ClientSecret <secret>
```

FINISH SETUP

➤ Update Startup/Configure:

```
services.AddAuthentication().AddGoogle(googleOptions =>
{
    googleOptions.ClientId = Configuration["Authentication:Google:ClientId"];
    googleOptions.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
});
```

➤ Login page now has a button for Google login

MICROSOFT

- Create/Register your app with <https://apps.dev.microsoft.com/>
 - Select Converged Apps and click Add App
 - Add name, skip guided setup
 - Add Web platform, enter call back URL, TOS and PP.
 - <https://localhost:<portnumber>/signin-microsoft>
 - Generate password – it's only displayed once
 - Add to secrets.json

```
dotnet user-secrets set Authentication:Microsoft:ApplicationId <key>  
dotnet user-secrets set Authentication:Microsoft:Password <secret>
```

FINISH SETUP

➤ Update Startup/Configure:

```
services.AddAuthentication().AddMicrosoftAccount(microsoftOptions =>
{
    microsoftOptions.ClientId = Configuration["Authentication:Microsoft:ApplicationId"];
    microsoftOptions.ClientSecret = Configuration["Authentication:Microsoft:Password"];
});
```

➤ Login page now has a button for Microsoft login

AUTHORIZATION

AUTHORIZATION

➤ Simple - Authorize, AllowAnonymous attributes

➤ Declarative role

```
[Authorize(Roles = "Administrator, SysAdmin")] //or  
[Authorize(Roles = "Administrator"), Authorize(Roles = "SysAdmin")] //and
```

➤ Policy based

➤ [Authorize(Policy="ITAdmin")]

```
services.AddAuthorization(options =>  
{  
    options.AddPolicy("ITAdmin", policy =>  
    {  
        policy.RequireAuthenticatedUser();  
        policy.RequireRole("Admin");  
        policy.RequireClaim("Department", "IT");  
    });  
});
```

CLAIMS

- Name Value pairs issued by a trusted party
 - Represents what the subject is, not what it can do
- Claims are policy based.
 - Either check for the presence of a claim or the presence and specific value

```
services.AddAuthorization(options =>
{
    options.AddPolicy("Employees",
        policy => policy.RequireClaim("EmployeeNumber"));
    options.AddPolicy("Founders",
        policy => policy.RequireClaim("EmployeeNumber", "1", "2", "3", "5"));
});
```

AUTHENTICATION IN VIEWS

- Role Authorization

 - `User.IsInRole("Dept")`

- Claims Authorization

 - `User.HasClaim("Department", "IT")`

- Policy Authorization

 - Inject the authorization service into the view

 - `@inject IAuthorizationService AuthorizationService`

CORS

ENABLE CROSS-ORIGIN REQUESTS (CORS) IN ASP.NET CORE

- Add CORS in Startup/ConfigureServices

 - `services.AddCors()`

- Enable CORS middleware

 - `app.UseCors(builder=>builder.WithOrigins("<origins>"));`

- In MVC, apply to controller, action, or globally

 - `[EnableCors]`, `[DisableCors]`

IDENTITYSERVER OVERVIEW

- IdentityServer is...
 - Open standards security protocols server
 - An OpenID Connect, WS-Federation, and SAML2p authentication server
 - And OAuth2 authorization server
 - Identity Provider (IdP)
 - Single Sign On (SSO) server

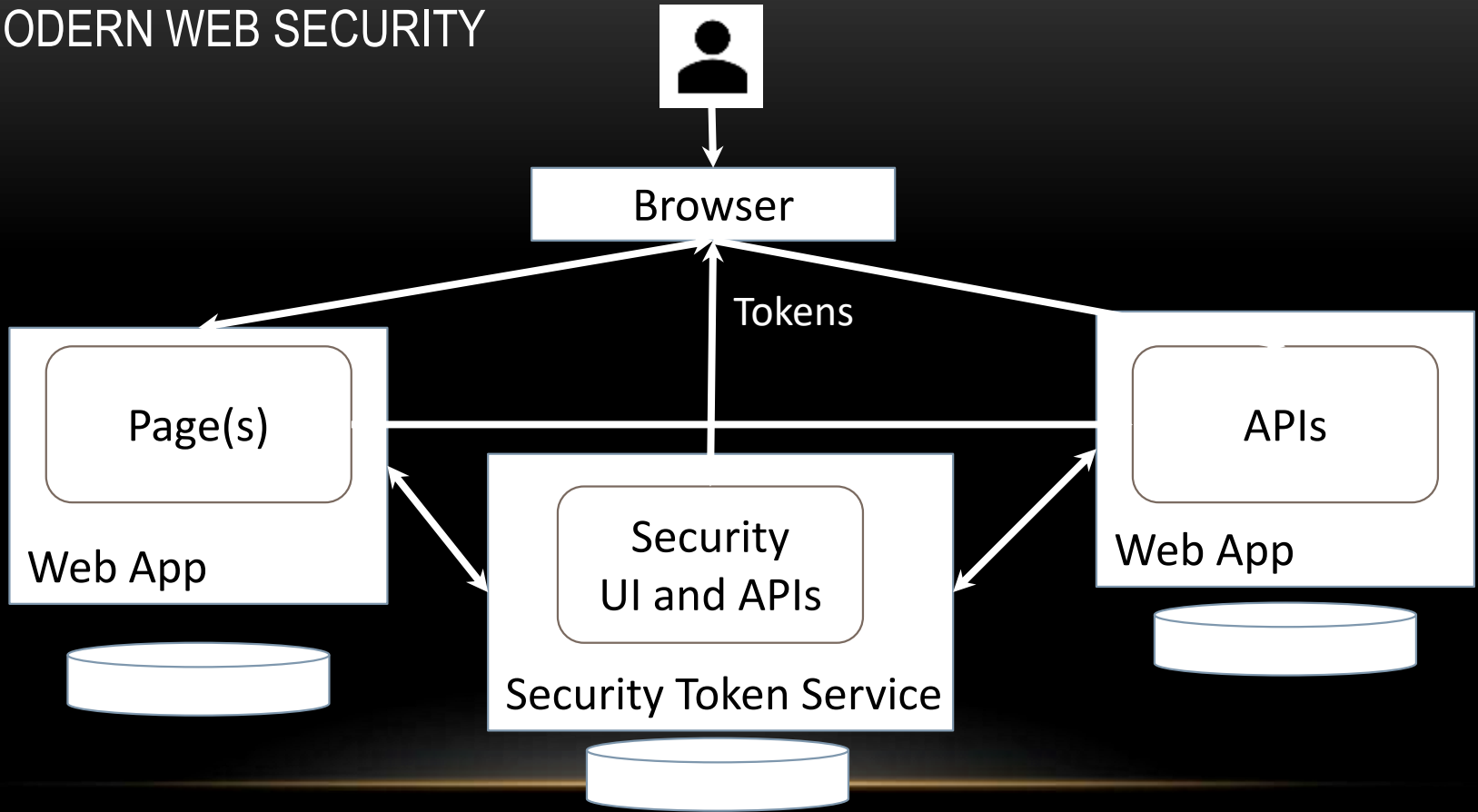
IDENTITYSERVER OVERVIEW

- Security Token Service (STS)
 - Encompasses all of those responsibilities
- Two versions
 - IdentityServer3 – ASP.NET 4.x basis
 - IdentityServer4 – ASP.NET Core basis

OPENID CONNECTION CLAIMS

- aud – Audience (recipient)
- Auth_time – When auth happened (nbf)
- exp – Expiration time
- nbf – Not Before (expiration)
- scope – Identity Scope
- sub – Subject (identity principal)
- idp – Identity Provider
- Iss – Issuer (URI)
- Client_id – Identity Client
- amr – Authentication method

MODERN WEB SECURITY



Contact Me

skimedic@outlook.com

www.skimedic.com/blog

www.twitter.com/skimedic

<http://bit.ly/skimediclyndacourses>

<http://bit.ly/apressbooks>

www.hallwayconversations.com

<https://github.com/skimedic/presentations>



Thank You!