Security in ASP.NET Core Applications

By

Narasimha Rao T

Microsoft.Net FSD Trainer

Professional Development Trainer

tnrao.trainer@gmail.com

1. Introduction to Security in ASP.NET Core

- ASP.NET Core provides a **flexible and robust security framework**.
- Key pillars of application security:
    - **Authentication**: Verifying the identity of users or services.
    - **Authorization**: Determining what authenticated users are allowed to do.
    - **Data protection**: Ensuring sensitive data is stored/transmitted securely.
    - **Secure communication**: Using HTTPS.

2.1. What is Authentication in ASP.NET Core?

- **Definition**: Authentication is the process of verifying the **identity** of a user or application.
- **Purpose**: Ensures that the user is who they claim to be.
- **Mechanisms in ASP.NET Core**:
  - Cookies (for web apps)
  - Tokens (for APIs, like JWT)
  - OAuth/OpenID Connect

- **Authentication Middleware**:
  - Configured in `Program.cs` or `Startup.cs`
  - Uses `AddAuthentication()` and `UseAuthentication()` methods

Example:

```
builder.Services.AddAuthentication("Bearer")
    .AddJwtBearer(options => {
        options.Authority = "https://your-auth-server";
        options.Audience = "api1";
    });
```

2.2. What is Authorization in ASP.NET Core?

- **Definition**: Authorization is the process of **determining what resources an authenticated user can access**.
- **Purpose**: Ensures users have the correct permissions to perform actions.
- **Types**:
    - Role-based (e.g., Admin, User)
    - Policy-based
    - Claims-based

Example:

```
[Authorize(Roles = "Admin")]
public IActionResult GetAdminData() {
    return Ok("Admin data");
}
```

3. Authentication vs Authorization

- **Authentication**: Who are you? (Identity verification)
- **Authorization**: What can you do? (Access control)
- **Order of Execution**: Authentication happens before authorization.

4. What is ASP.NET Core Identity?

- **Definition**: ASP.NET Core Identity is a membership system that provides:
  - User registration
  - Password hashing
  - Role management
  - Claims support
- **Components**:
  - `UserManager`, `SignInManager`, `RoleManager`
  - EF Core integration for database persistence

5. How to Involve ASP.NET Core Identity in Security

1. **Install NuGet Packages**:

   `Microsoft.AspNetCore.Identity.EntityFrameworkCore`

2. **Configure Identity in Services**:

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

3. **Use Authentication Middleware**:

```
app.UseAuthentication();
app.UseAuthorization();
```

6. What is JWT (JSON Web Token)?

- **Definition**: JWT is an open standard (RFC 7519) for securely transmitting information as a JSON object.
- **Structure**:
  - Header (algorithm, token type)
  - Payload (claims: user data, roles)
  - Signature (to verify integrity)
- **Advantages**:
  - Stateless (no server session storage)
  - Works well for APIs and distributed systems

7. Implementing Role-Based Access with JWT

- Add roles to JWT claims when issuing the token.
- Protect endpoints using `[Authorize(Roles = "Admin")]`.

Token Generation Example:

```
var authClaims = new List<Claim>
{
    new Claim(ClaimTypes.Name, user.UserName),
    new Claim(ClaimTypes.Role, "Admin"),
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
};
```

8. Steps to Implement JWT in ASP.NET Core

1. **Install Package**: `Microsoft.AspNetCore.Authentication.JwtBearer`
2. **Configure JWT Authentication**:

```
builder.Services.AddAuthentication(options => {
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options => {
    options.TokenValidationParameters = new TokenValidationParameters {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("your_secret_key"))
    };
});
```

3. **Generate JWT Token** when user logs in.
4. **Add `[Authorize]` Attribute** to secure endpoints.

9. Securing Endpoints in ASP.NET Core

- Use `[Authorize]` attribute on controllers or actions.
- Use `[AllowAnonymous]` where security is not required.
- Implement policies:

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminPolicy", policy => policy.RequireRole("Admin"));
});
```

Then apply:

```
[Authorize(Policy = "AdminPolicy")]
```

10. Other Security Best Practices

- Always store **passwords as hashed** (Identity does this automatically).
- Use **data protection API** for secure key storage.
- Implement **refresh tokens** for long-lived sessions.
- Protect against **CSRF** (for browser-based apps).
- Enable **logging & auditing** for security events.

11. Conclusion

- Authentication verifies **who you are**.
- Authorization verifies **what you can do**.
- **ASP.NET Core Identity + JWT** provides robust security for APIs.
- **Role-based and policy-based authorization** ensures fine-grained control.

Self-Check Questions

1. Difference between Authentication and Authorization?
2. What are Claims in JWT?
3. How does ASP.NET Core Identity work internally?
4. What is the difference between Cookie-based and Token-based Authentication?
5. How do you secure APIs in ASP.NET Core?
6. How to refresh JWT tokens?
7. What is the difference between Policy-based and Role-based Authorization?
8. What is the role of middleware in authentication/authorization?