

# Developing Cross-Platform Web Apps With Blazor

Wael Kdouh - @waelkdouh

Senior Customer Engineer

v1.0

## Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

## Copyright and Trademarks

© 2013 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

<http://www.microsoft.com/about/legal/permissions/>

Active Directory, Azure, IntelliSense, Internet Explorer, Microsoft, Microsoft Corporate Logo, Silverlight, SharePoint, SQL Server, Visual Basic, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

# How to View This Presentation

- To switch to **Notes Page** view:
  - On the ribbon, click the **View** tab, and then click **Notes Page**
- To navigate through notes, use the Page Up and Page Down keys
  - Zoom in or zoom out, if required
- In the **Notes Page** view, you can:
  - Read any supporting text—now or after the delivery
  - Add notes to your copy of the presentation, if required
- Take the presentation files home with you

# Module 10: Security

## Module Overview

## Module 10: Security

### Section 1: Security Fundamentals

#### Lesson: Overview

# What Is Security? How to Think About It?

- **Prevention**

- Prevent the system from reaching compromised state
- For example, Secure Development Lifecycle

- **Detection and Recovery**

- Detect that the system has been compromised and recover it to secure state
- For example, Intrusion Detection Systems (IDS)

- **Resilience**

- Ensure minimum functionality in the compromised state
- For example, redundancy or diversity in physical infrastructure or technology

- **Deterrence**

- Deter the malicious users/mechanisms from malicious acts
- For example, Law enforcement, legislations, international collaboration

# Security Principles

- Do not trust anything (including user input)
- Know the weakest link
- Multiple layers of security
- Least privilege
- Secure fallback when things go wrong
- Universally check access permissions
- Minimize shared information
- Do not depend on secrecy
- Keep it simple (KISS)

# Application Architectures and Security

- Thick client applications
  - Windows authentication
- Server-side web applications
  - Windows or Forms authentication
- Not service-based



# Application Architectures and Security

- Service-oriented architecture
  - WS-Security (WCF)
  - IP-level configuration (firewall)
- SAML 2.0
  - Standard for exchanging authentication and authorization data between security domains

# Application Architectures and Security

- Http APIs came to the seen
  - Those no longer necessarily lived in the same domain
- We are no longer just building server-side web apps that live inside company walls
  - We started building client-side web application like Blazor that live in the browser and communicate with APIs
  - We started building mobile apps that communicate with APIs
  - We started building APIs that communicate with other APIs

# Application Architectures and Security

- Http APIs need to be public
  - Cannot be secured with forms authentication anymore
  - A mobile application doesn't even have a notion of forms authentication
  - Server-side web applications aren't limited to talking to APIs within the company
- Sending username/password on each request proved to be a bad idea

# Application Architectures and Security

- Token-based Security
  - Client applications send tokens
  - Tokens represent consent to access the API
- How do we create these tokens and how do we safely deliver them to the client applications that require them?
  - In-house token generating services emerged that would require your username and password and return a Json Web Token (JWT)

# Application Architectures and Security

Expiration

Token signing and  
validation

Token format

Authentication and  
authorization

Secure delivery of  
tokens to different  
application types

...

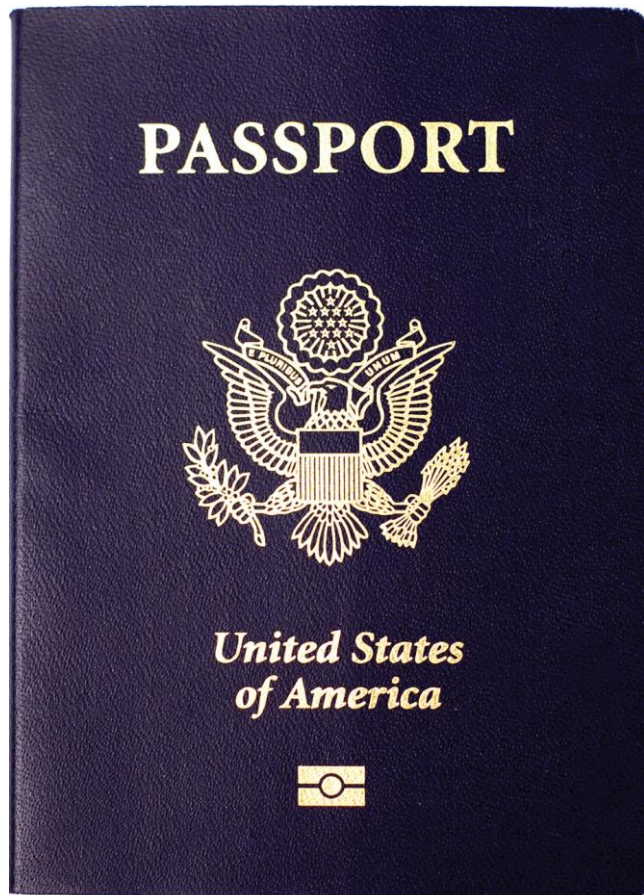
# What Is The Problem Statement?

- We are really trying to solve two issue here

# Authentication – Hotel Analogy

You arrive at the hotel and you are asked for some type of identity

**Formal Document - This proves to them your identity**



**They know that because there are certain claims on the document**

```
{  
  "userID": "83b6734e",  
  "username": "SuzyQ",  
  "Name": "Suzy",  
  "givenName": "Q",  
  "premiumMember": true  
}  
  
{  
  "userID": "ba35b637",  
  "username": "JohnDoe",  
  "Name": "John",  
  "givenName": "Doe",  
  "premiumMember": false  
}
```

# Identity

- How do we *represent* a user in our application?
- Typically: A collection of key : value pairs that describe a specific user
  - A pair is referred to as a **claim**
  - The collection of claims makes up an **Identity**
- Represented in code as a model we can create, store, and manipulate
- Can be unique to your app, or shared across apps (Single Sign On)

```
{  
  "userID": "83b6734e",  
  "username": "SuzyQ",  
  "Name": "Suzy",  
  "givenName": "Q",  
  "premiumMember": true  
}
```

```
{  
  "userID": "ba35b637",  
  "username": "JohnDoe",  
  "Name": "John",  
  "givenName": "Doe",  
  "premiumMember": false  
}
```



# Authentication

- Verifying the users are who they say they are



Microsoft account [What's this?](#)

☐ Keep me signed in

**Sign in**

[Can't access your account?](#)

[Sign in with a single-use code](#)

### Set up your PIN

A PIN is a quick, convenient way to log in to this Windows PC using touch.

Enter PIN

Confirm PIN

**Finish** **Cancel**





# ASP.NET Core Template Authentication Methods

X

## Create a new Blazor app

.NET Core 3.1

**Blazor Server App**  
A project that uses Blazor to create a rich user interface (UI) that runs in the browser.

**Blazor WebAssembly**  
A project that uses Blazor to create a rich user interface (UI) that runs in the browser.

**Authentication**  
Change Authentication  
For applications that don't require any user authentication.  
[Learn more](#)  
☒ No Authentication  
☐ Individual User Accounts  
☐ Work or School Accounts  
☐ Windows Authentication  
[Learn more about third-party open source authentication options](#)

☒ ASP.NET Core hosted  
☐ Progressive Web Application

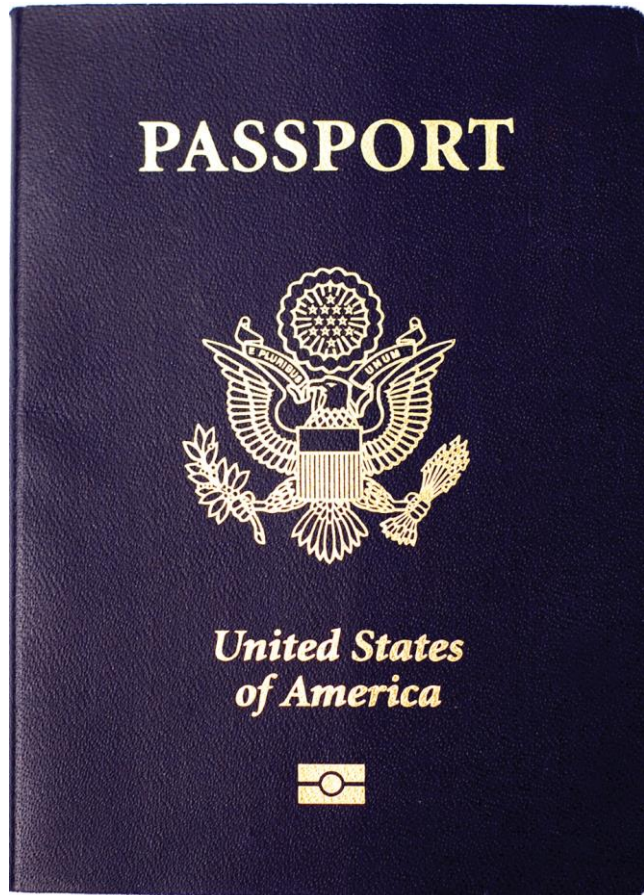
# ASP.NET Core Template Authentication Methods

- No authentication
- Individual User Accounts
  - Store user accounts in-app (ASP.NET Identity)
  - Connect to an existing user store in the cloud (OpenID compliant Identity Provider)
    - e.g., Azure AD B2C
- Work or School Accounts
  - Active Directory
  - Azure Active Directory
  - Office 365
- Windows Authentication
  - Internet Information Services (IIS) Windows Authentication module

# Authorization – Hotel Analogy

After the hotel confirms your identity and the claims are checked on the passport the hotel provides you with a key

**Formal Document - This proves to them your identity**



**Hotel key provides limited access**



# Authorization

- What can a user *do*?
- Needs authentication first
- Many strategies for approaching this important question:
  - Role-Based Authorization
  - Claims-Based Policy Authorization
  - Manual Custom Authorization

```
{  
  "userID": "83b6734e",  
  ...  
  "role": "SysAdmin",  
  "canEditForm": true,  
  "dob": "1/1/1985"  
}
```

```
{  
  "userID": "ba35b637",  
  ...  
  "role": "SDET2",  
  "canEditCode": true,  
  "dob": "1/1/1970"  
}
```

# Authentication with [Authorize] Attribute

- [Authorize] attribute by itself is used to require an authenticated user
- [Authorize] attribute can be used to restrict access to:
  - Specific action methods in a controller
  - Controller → every action method within the controller
- [Authorize] should be applied to each controller/action except login/register methods

- Controller

```
[Authorize]
3 references | 0 changes | 0 authors, 0 changes
public class HomeController : Controller
{
```

- Action

```
[Authorize]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public IActionResult About()
{
    ViewData["Message"] = "Your Employee application description page.";

    return View();
}
```

# Demo: Blazor Authentication

Module 10: Security

Section 2: ASP.NET Identity

Lesson: Overview



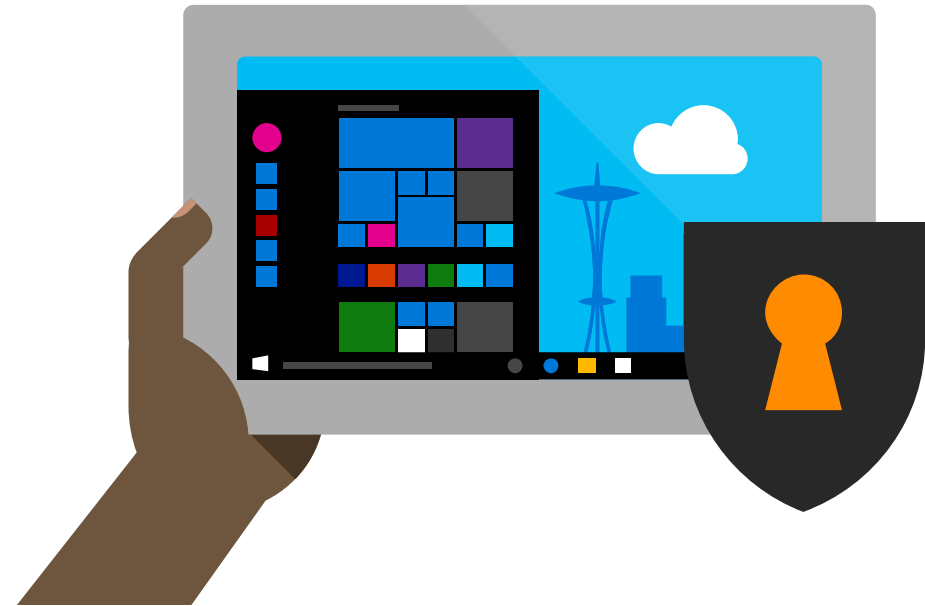
# ASP.NET Identity

Seamless and unified experience for enabling authentication in ASP.NET apps on-premises and in the cloud.



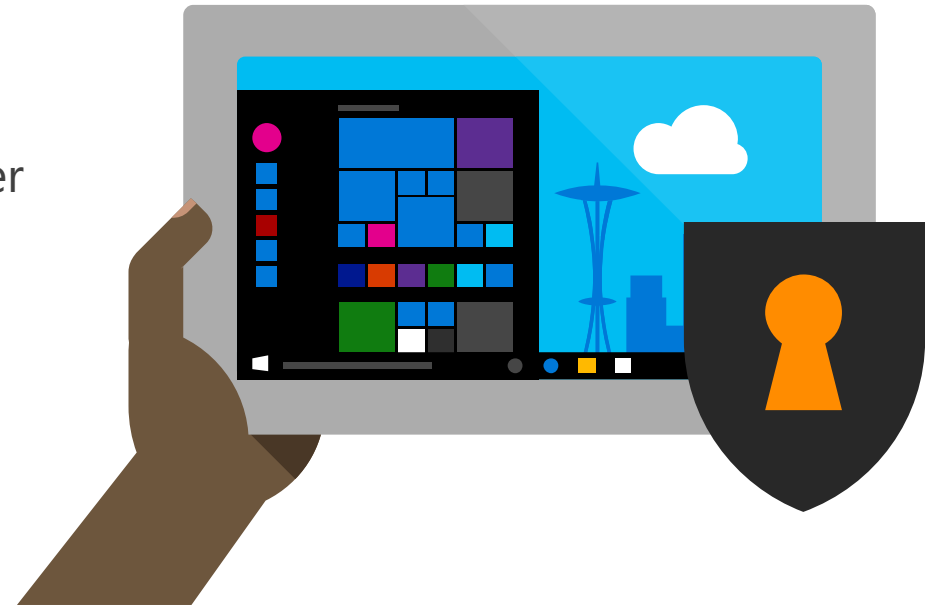
# ASP.NET Identity

- **Easily pluggable user profile**
  - Complete control over the schema of user and profile information
- **Persistence control**
  - SQL Server (Default), Microsoft SharePoint, Azure Storage Table Service, NoSQL databases
- **Role Provider**
  - Role-based authorization
- **Claims-based Authentication**
  - Includes rich information about user's identity



# ASP.NET Identity

- **Unit Testability**
  - Authentication/authorization logic independently testable
- **Social Login Providers**
  - Microsoft account, Facebook, Google, Twitter, and others...
- **Azure AD**
  - Single and multi-organization support
- **Azure AD B2C**
  - Managed OAuth/OpenID compliant Identity provider
- **NuGet package**
  - Agility in release of new features and bug fixes



# Features

- Two-Factor authentication
- Email/phone verification
- Roles and Claims
- Profile
- User Management
- Role Management
- Password policy enforcement
- User password management
- Account lockout
- Extensibility



# ASP.NET Identity Configuration

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseAuthentication();
    app.UseAuthorization();
}
```

Startup.cs

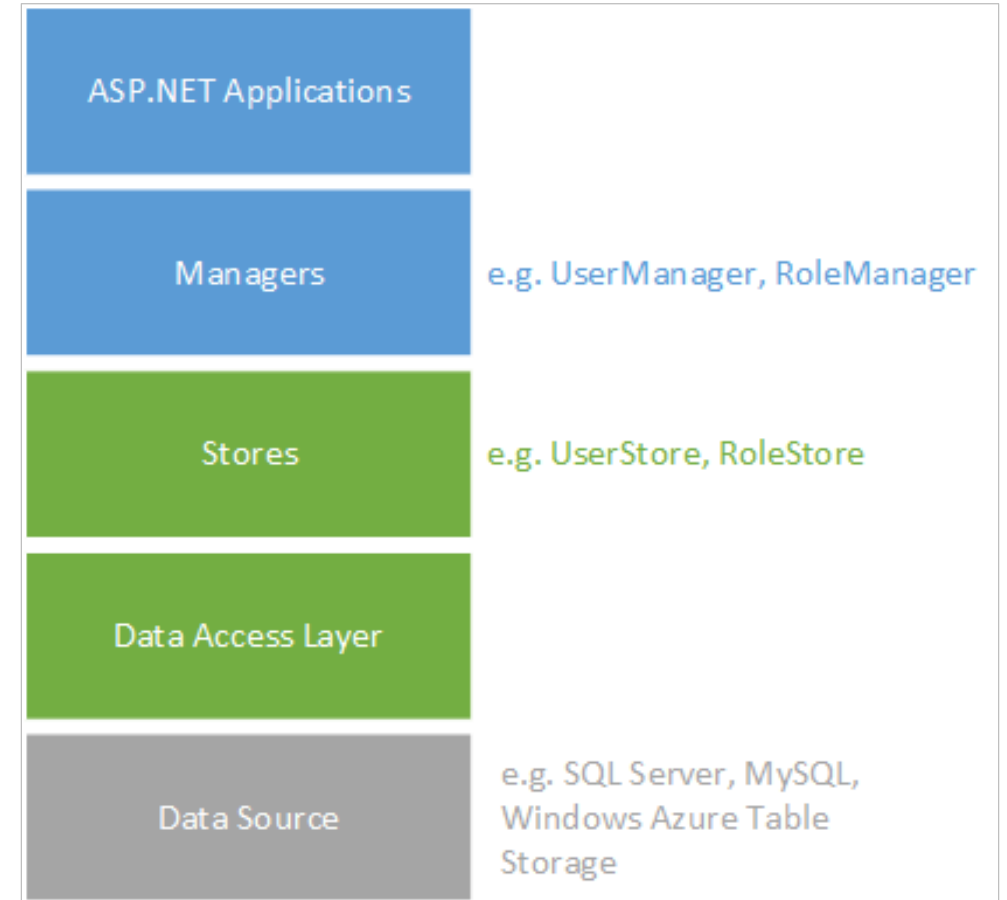
# ASP.NET Identity Architecture

- **Managers**

- High-level classes
- Operations such as create user
- Completely decoupled from stores

- **Stores**

- Lower-level classes
- Closely coupled with the persistent mechanism
- Store users, roles, claims through Data Access Layer (DAL)

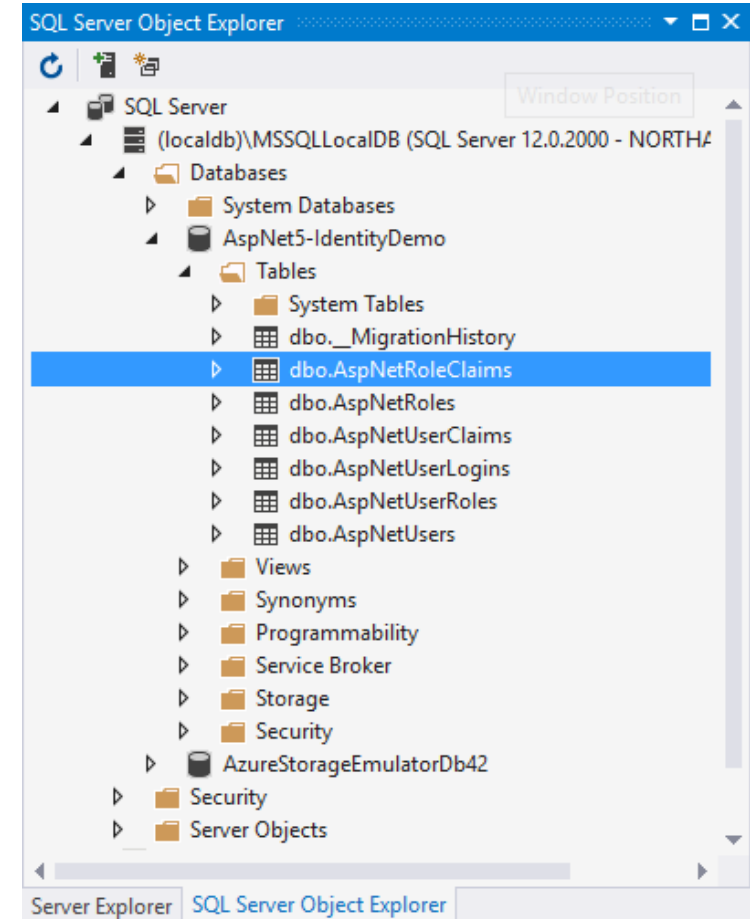


# ASP.NET Identity Key Classes

- **IdentityUser** – Represents web application user
- **EmailService, SmsService** – Notified during two-factor authentication
- **userManager** – APIs to CRUD (Create, Read, Update, and Delete) user, claim, and auth information via UserStore
- **RoleManager** – APIs to CRUD roles via RoleStore
- **UserStore** – Talks to data store to store user, user login providers, user claims, user roles,
  - IUserStore, IUserLoginStore, IUserClaimStore, IUserRoleStore
- **RoleStore** – Talks to the data store to store roles
- **SignInManager** – High level API to sign in (single or two-factor)

# ASP.NET Identity Database

Data	Description
Users	Registered users of your web site. Includes the user Id and user name. Might include a hashed password if users log in with credentials that are specific to your site (rather than using credentials from an external site like Facebook), and security stamp to indicate whether anything has changed in the user credentials. Might also include email address, phone number, whether two factor authentication is enabled, the current number of failed logins, and whether an account has been locked.
User Claims	A set of statements (or claims) about the user that represent the user's identity. Can enable greater expression of the user's identity than can be achieved through roles.
User Logins	Information about the external authentication provider (like Facebook) to use when logging in a user.
Roles	Authorization groups for your site. Includes the role Id and role name (like "Admin" or "Employee").





# Demo: ASP.NET Identity Setup in Project Template

Module 10: Security

Section 3: Authorization

Lesson: Authorization  
Methodologies

# Roles-Based Authorization

- [Authorize] attribute can be used to restrict access to specific users and roles

- Restricting StoreManagerController to Administrators only

```
[Authorize(Roles = "Administrator")]  
public class StoreManagerController : Controller
```

- Restricting controller/action to **any** of multiple roles (logical OR)

```
[Authorize(Roles = "Administrator, SuperAdmin")]  
public class StoreManagerController : Controller
```

- Restricting controller/action to **all** of multiple roles (logical AND)

```
[Authorize(Roles = "Administrator"), Authorize(Roles = "SuperAdmin")]  
public class StoreManagerController : Controller
```

- Restricting controller/action to multiple users & roles

```
[Authorize(Users = "User1, User2", Roles = "SuperAdmin")]  
public IActionResult Create(Album album)
```

# Claims-Based Policy Authorization - I

- [Authorize] attribute can be used to restrict access to users with specific claims
  - Create a policy for requiring a claim or claim value

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("EmployeeOnly", policy => policy.RequireClaim("EmployeeNumber"));

        options.AddPolicy("FounderOnly", policy =>
            policy.RequireClaim("EmployeeNumber", "1", "2", "3", "4", "5"));
    });
}
```

Startup.cs

# Claims-Based Policy Authorization - II

- [Authorize] attribute can be used to restrict access to users with specific claims

- Restricting controller/action to **all** of multiple Policies (logical AND)

```
[Authorize(Policy = "EmployeeOnly"), Authorize(Policy = "FounderOnly")]
```

```
public class StoreManagerController : Controller
```

- Restricting controller/action to any of multiple Policies (logical OR)

```
[Authorize(Policy = "EmployeeOnly, FounderOnly")]
```

```
public IActionResult Create(Album album)
```

# Demo: ASP.NET Core Identity

# Tokens

## Access Token and ID Token

- OIDC

## Access Token

- OAuth 2.0

## Refresh Token

- Can be obtained by both OIDC and OAuth 2.0 protocols



## Module 10: Security

### Section 6: ASP.NET Identity Strategies

### Lesson: ASP.NET Identity Strategies

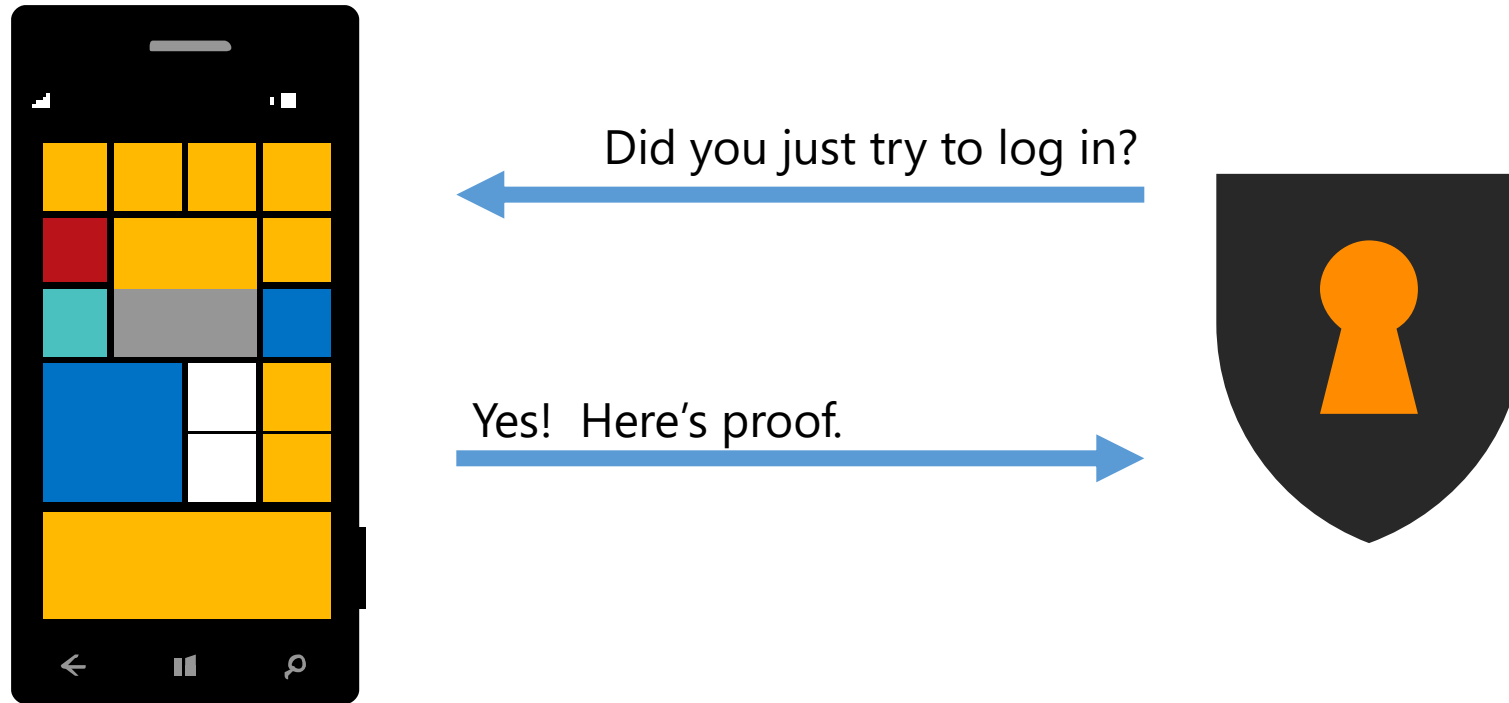


# Recommendations

- Utilize Secure Sockets Layer (TLS/SSL - HTTPS) everywhere
  - Attacker on network can steal your cookies and hijack your session
  - Yes, even login page needs to be protected
  - Any page user can access while logged in should be protected
- Enforce a strong password policy (more an art than a science)
- Use Cross-Site Request Forgery (CSRF) tokens everywhere for post methods
- Do not allow unlimited login attempts
  - Brute forcers dream. Script kiddies abound.

## Recommendations (continued)

- **If** security requirements demand it, you can change password hashing method
- Consider shortening `OnValidateIdentity` times to expire sessions
- Two-Factor authentication is highly recommended for enhanced security



## Note that...

- Password expiration is not built-in
  - It is not right for every system, a good policy but consider it carefully
- Identity is not multi-tenant or multi-app by default
  - Use Azure AD or add Tenant IDs to users for multi-tenancy
  - Put Identity in a separate SQL server to share across apps (*not* true SSO)

## Module 10: Security

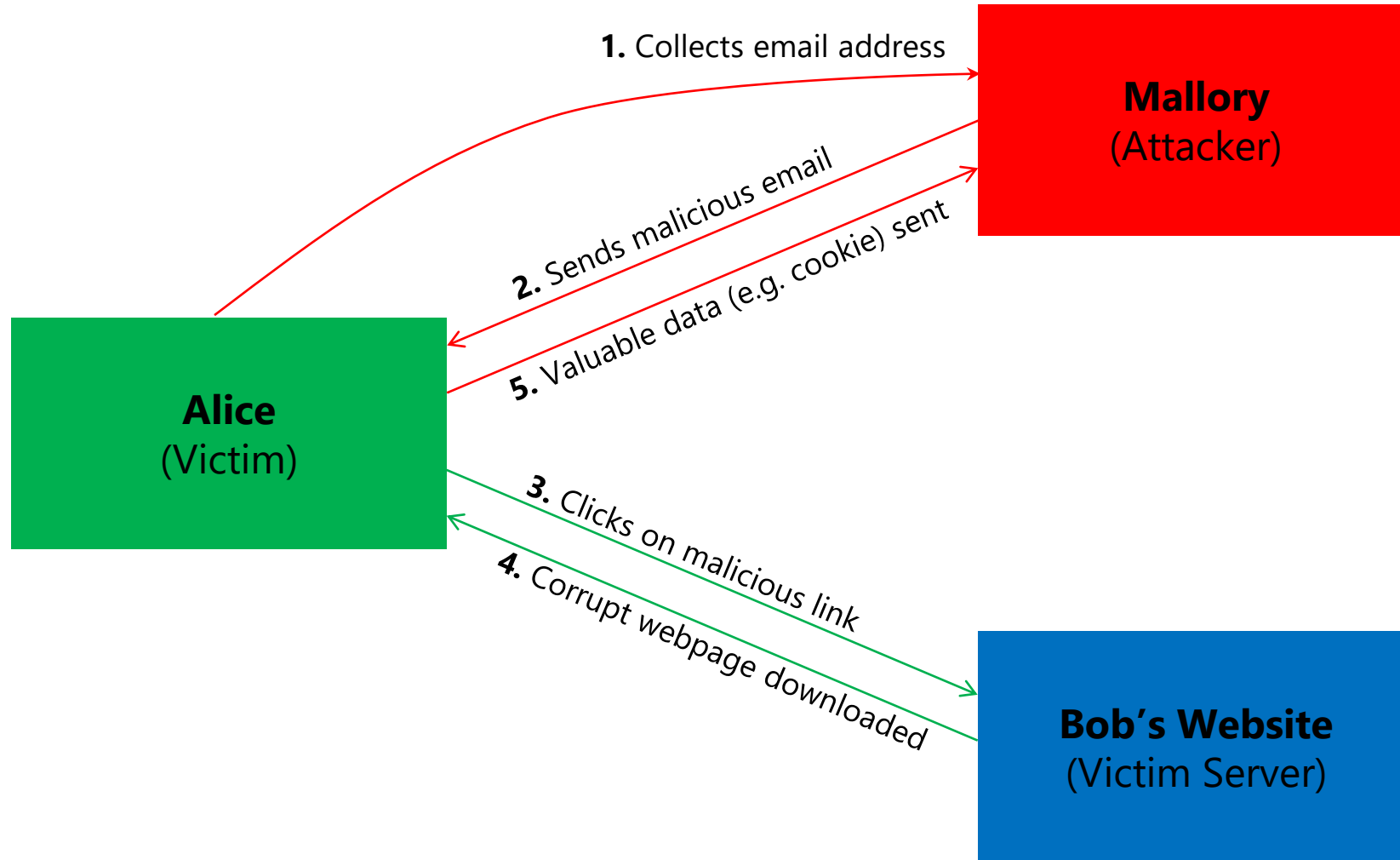
### Section 7: Security Threats and Defenses

#### Lesson: Web Attacks and Defenses

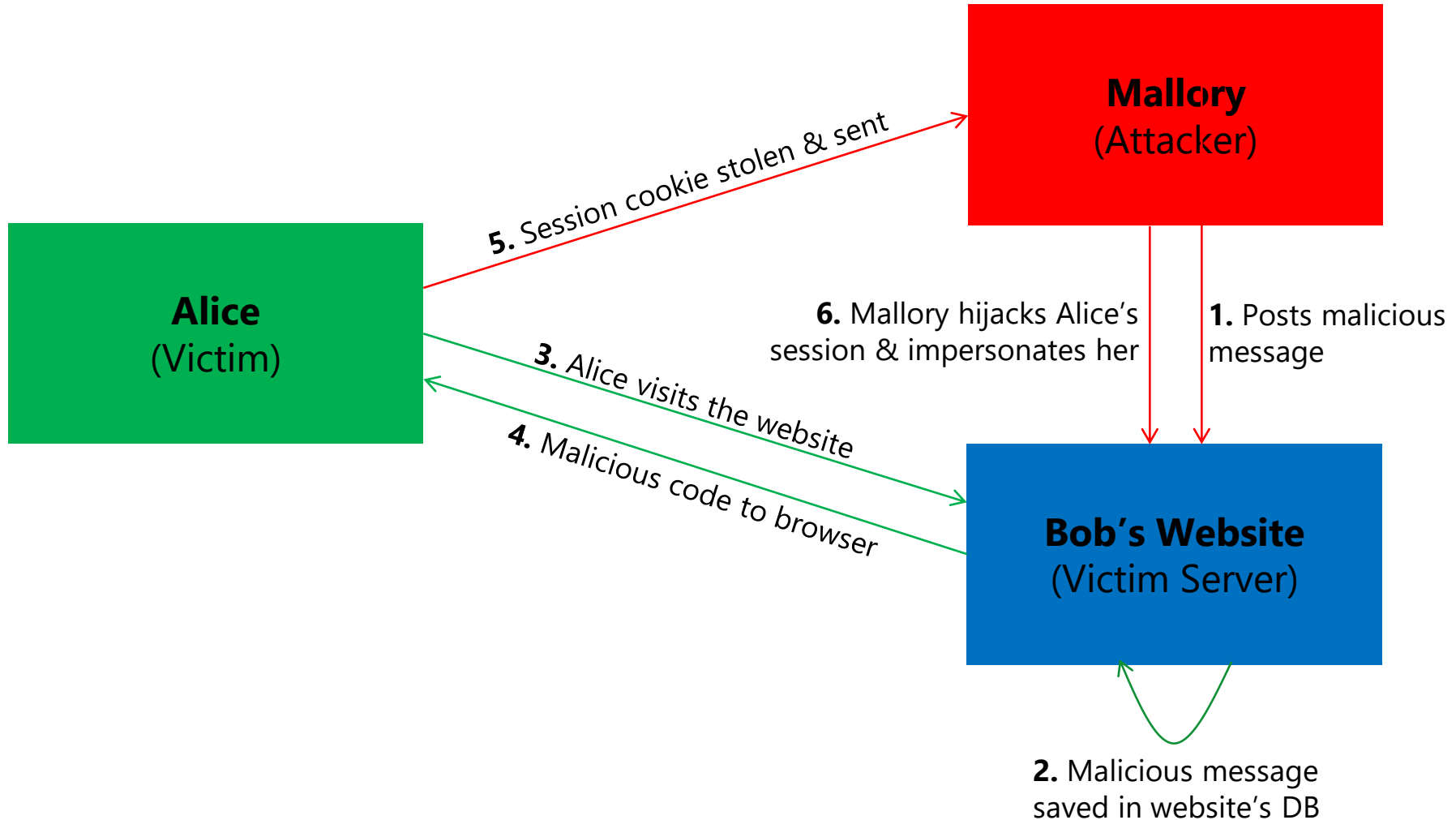
# Cross-Site Scripting (XSS) Attack

- XSS vulnerability allows an attacker to inject malicious JavaScript into pages generated by a web application
- Malicious script executes in victim client's browser
  - To gain access to sensitive webpage content, session cookies, etc.
- Methods for injecting malicious code:
  - **Active or Reflected Injection**
    - Attack script directly reflected back to the user from the victim site
    - Victim user participates directly in the attack
    - Often done through social engineering tricks, such as malicious email
  - **Passive or Stored Injection**
    - Malicious code is saved in the backend database using user input
    - Potentially more dangerous because all users of the web application may be compromised

# XSS Reflected Attack



# XSS Stored Attack



# XSS Defense

- Never trust any input to your website
- Ensure that your app validates all user input, form values, query strings, cookies, information received from third-party sources, for example, OpenID
- Use whitelist approach instead of trying to imagine all possible hacks
  - It is not possible to know all permutations
- Remove/encode special characters
  - HTML encoding
  - JavaScript encoding



# HTML Encoding

- All output on your pages should be HTML-encoded or HTML-attribute-encoded
  - `@Html.Encode(Model.FirstName)`
  - `@Model.FirstName`
- URL Encoding:
  - `@Url.Encode(Url.Action("index", "home", new {name=ViewData["name"]})))`
- Razor View Engine automatically HTML-encodes output

Malicious User Input (without encoding)

```
<script>alert("XSS!")</script>
```

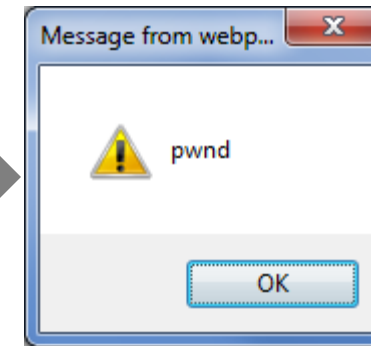
HTML-Encoded User Input

```
&lt;script&gt;alert('XSS!')&lt;/script&gt;
```

# JavaScript Encoding

```
<h2 id="welcome-message">Welcome to our website</h2>

@if(!string.IsNullOrEmpty(ViewBag.UserName)) {
<script type="text/javascript">
    $(function () {
        var message = 'Welcome, @ViewBag.UserName!';
        $("#welcome-message").html(message).hide().show('slow');
    });
</script>
}
```



`http://localhost:XXXXX/?UserName=Waqar\x3cscript\x3e%20alert(\x27pwnd\x27)%20\x3c/script\x3e`

## JavaScript Encoding Fix

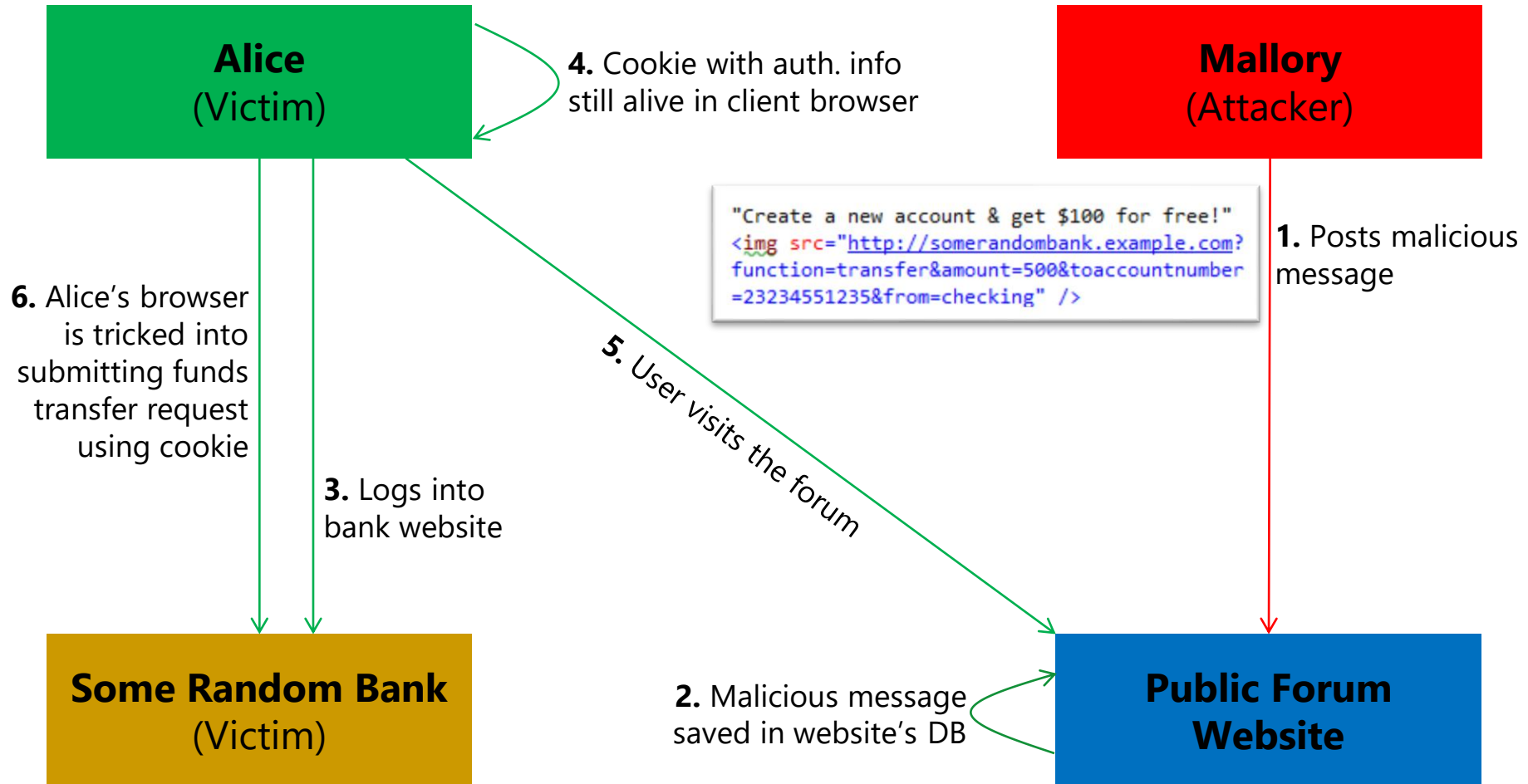
```
$(function () {
    var message = 'Welcome, @Ajax.JavaScriptStringEncode(ViewBag.UserName)!';
    $("#welcome-message").html(message).hide().show('slow');
});
```

# Demo: Cross-Site Scripting Attack

# CSRF Attack

- CSRF attack tricks a browser into misusing its authority to represent a user to remote website
- CSRF exploits user's trust in a browser
  - Confused Deputy Attack against a web browser
- Characteristics of "at-risk" sites:
  - Reliance on user identity
  - Perform actions on input from authenticated user *without* requiring explicit authorization

# CSRF Attack (continued)



# CSRF Defense

- **AntiForgery token:** A hidden form field that is validated when the form is submitted
  - Both Html Helper and Tag Helper based forms will *automatically* create an AntiForgery token and include it as a hidden field

```
<form asp-controller="Manage" asp-action="ChangePassword" method="post">  
  
</form>
```

```
@using (Html.BeginForm("ChangePassword", "Manage"))  
{  
  
}
```

# Syntax of the Anti-Forgery Token

```
<% using(Html.Form("UserProfile", "SubmitUpdate")) { %>  
    <%= Html.AntiForgeryToken() %>  
    <!-- rest of form goes here -->  
<% } %>
```

# CSRF Defense

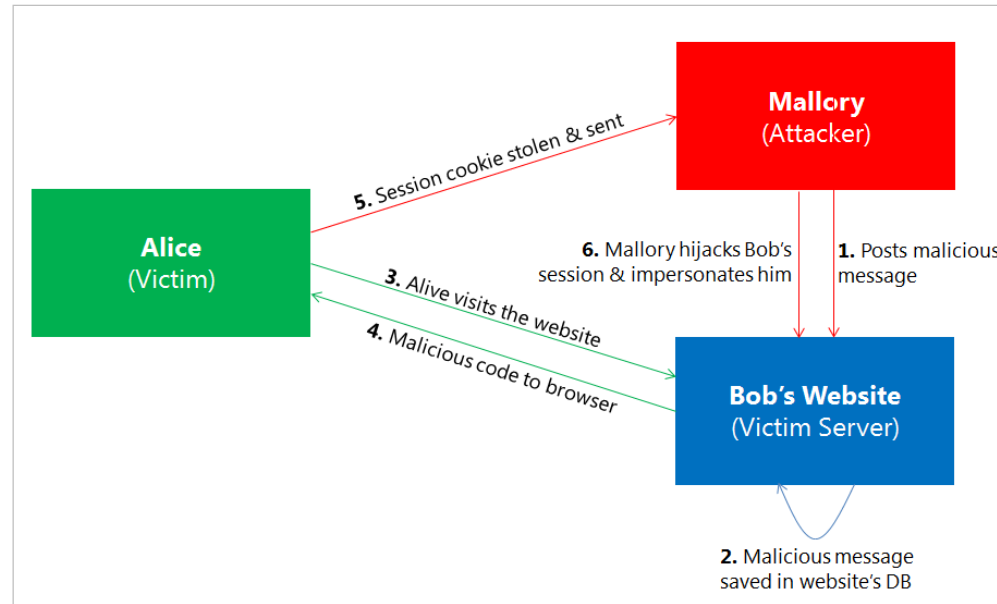
- **AntiForgery token:** A hidden form field that is validated when the form is submitted
  - Validate the token on the server side via the `[ValidateAntiForgeryToken]`

```
//  
// POST: /Account/Login  
[HttpPost]  
[AllowAnonymous]  
[ValidateAntiForgeryToken]  
1 reference  
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)  
{  
    EnsureDatabaseCreated(_applicationDbContext);  
}
```



# Cookie Stealing Attack

- Attacker steals user's authentication cookie for a website to impersonate user and carry out actions on user's behalf
- Dependent on XSS attack
  - Attacker must be able to inject script on the target site
  - Script sends user's authentication cookie to attacker's remote server



# Cookie Stealing Defense

- Prevent XSS attack on the website
- Disallow changes to the cookie from the client's browser
  - Browser will invalidate the cookie unless the server sets/changes it
  - Can be done from web.config if using IIS

```
<system.web>  
  <httpCookies domain="String" httpOnlyCookies="true" requireSSL="false"/>  
</system.web>
```

- Can also be set when configuring Cookies in Startup.cs

```
.AddCookie(opts => opts.Cookie.HttpOnly = true );
```

# Over-Posting Attack

- An attacker can populate model properties that are not included in the View.

## Model

```
public class Review
{
    public int ReviewID { get; set; } // Primary key
    public int ProductID { get; set; } // Foreign key
    public Product Product { get; set; } // Foreign entity
    public string Name { get; set; }
    public string Comment { get; set; }
    public bool Approved { get; set; }
}
```

## View

```
Name: @Html.TextBox("Name") <br>
Comment: @Html.TextBox("Comment")
```

- Attacker can add "Approved=true" to form post.
- Attacker can post values for Product, such as Product.Price, to change values in the persistent storage.

# Over-Posting Defense

- Use [bind] attribute to explicitly control the binding behavior
  - Specifically list permitted properties
- Use View Model [recommended]

```
// POST: Movies/Edit/6
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(
    [Bind("ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (ModelState.IsValid)
    {
        _context.Update(movie);
    }
}
```

**[Bind]**

```
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    1 reference
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    1 reference
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    2 references
    public bool RememberMe { get; set; }
}
```

**View Model**

# Lab 10: Security



# References

- [Microsoft Docs](#)
- [Blazor University](#)

