.NET Framework: Developing Modern Web Apps with ASP.NET MVC – Workshop*PLUS*

Wael Kdouh

Senior Consultant

v1.0

Module 9: Security

Module Overview

Module 9: Security

Section 1: Security Fundamentals

Lesson: Overview

What Is Security? How to Think About It?

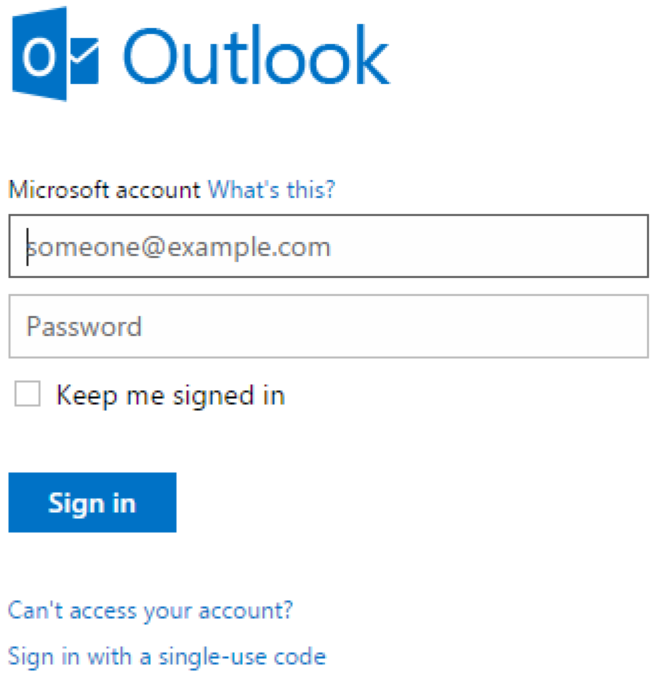
- **Prevention**
  - Prevent the system from reaching compromised state
  - For example, Secure Development Lifecycle
- **Detection and Recovery**
  - Detect that the system has been compromised and recover it to secure state
  - For example, Intrusion Detection Systems (IDS)
- **Resilience**
  - Ensure 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

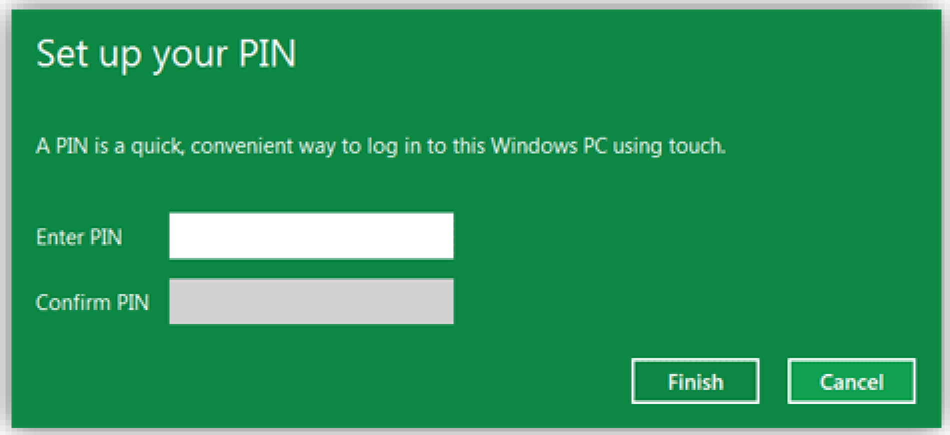
Security Principles

- 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)

Authentication

- Verifying the users are who they say they are

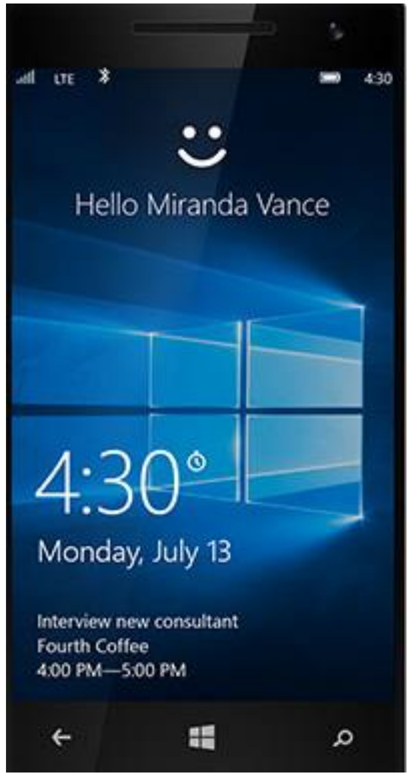

Outlook
Microsoft account What's this?
someone@example.com
Password
Keep me signed in
Sign in
Can't access your account?
Sign in with a single-use code


Set up your PIN

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

Enter PIN

Confirm PIN

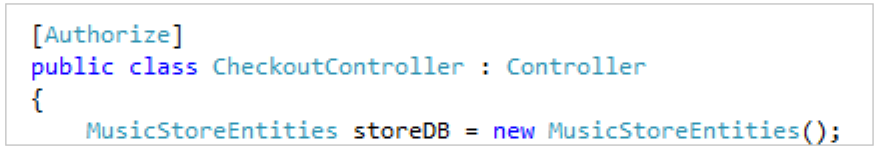


ASP.NET Authentication Methods

- No authentication
- Individual User Accounts (formerly ASP.NET Membership)
  - ASP.NET Identity
- Organizational Accounts
  - Windows Active Directory
  - Microsoft Azure Active Directory
- Windows Authentication
  - Internet Information Services (IIS) Windows Authentication module

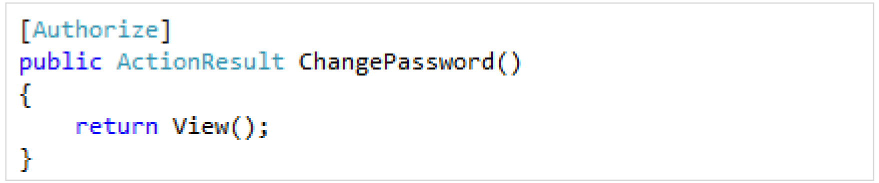
Authentication with [Authorize] Attribute

- [Authorize] attribute is used to require user authentication
- [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]
public class CheckoutController : Controller
{
    MusicStoreEntities storeDB = new MusicStoreEntities();
```

  - Action



```
[Authorize]
public ActionResult ChangePassword()
{
    return View();
}
```

Authorization and User Roles

- [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 multiple roles

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

    - Restricting controller/action to multiple users & roles

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

Demo: ASP.NET MVC Authentication

Module 9: Security

Section 2: ASP.NET Identity

Lesson: Overview

History of ASP.NET Account Services

- Nov 2005 ASP.NET 2.0 – Introducing Membership!
  - Microsoft SQL Server, Microsoft SQL Server Express
- May 2012 Universal Providers (First NuGet)
  - SQL CE, Azure, one provider to access all SQL
- Aug 2012 Simple Membership
  - Sourced in Web Pages, came to MVC / Web Forms
- Oct 2013 ASP.NET Identity v1
  - Completely new model
- Mar 2014 ASP.NET Identity v2
  - Two factor, account lockout, confirmation, reset, etc.

ASP.NET Identity

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

ASP.NET Identity

- **One ASP.NET Identity system**
  - ASP.NET MVC, Web Forms, Web Pages, Web API, and SignalR
- **Ease of plugging in profile data about the user**
  - 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 (continued)

- **Unit Testability**
  - Authentication/authorization logic independently testable
- **Social Login Providers**
  - Microsoft account, Facebook, Google, Twitter, and others…
- **Azure AD**
  - Single and multi-organization support
- **Open Web Interface for .NET (OWIN) Integration**
  - Based on OWIN middleware
- **NuGet package**
  - Agility in release of new features and bug fixes

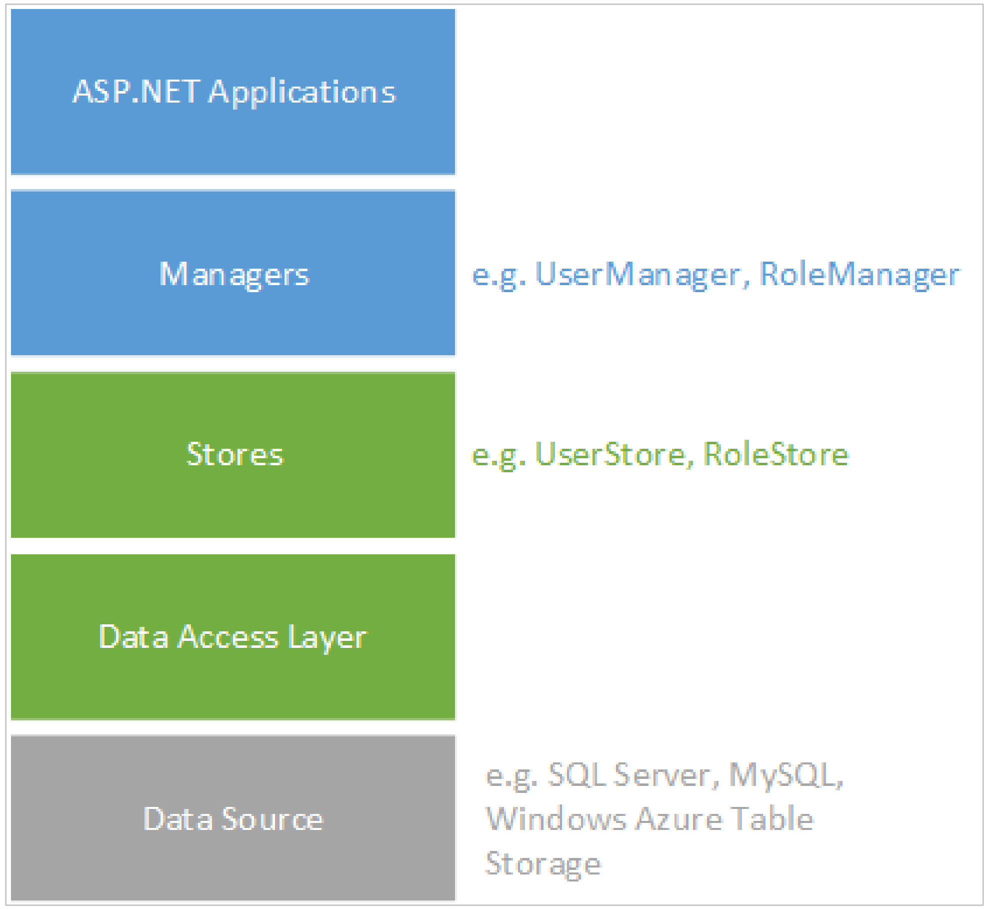
Features

- 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 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)

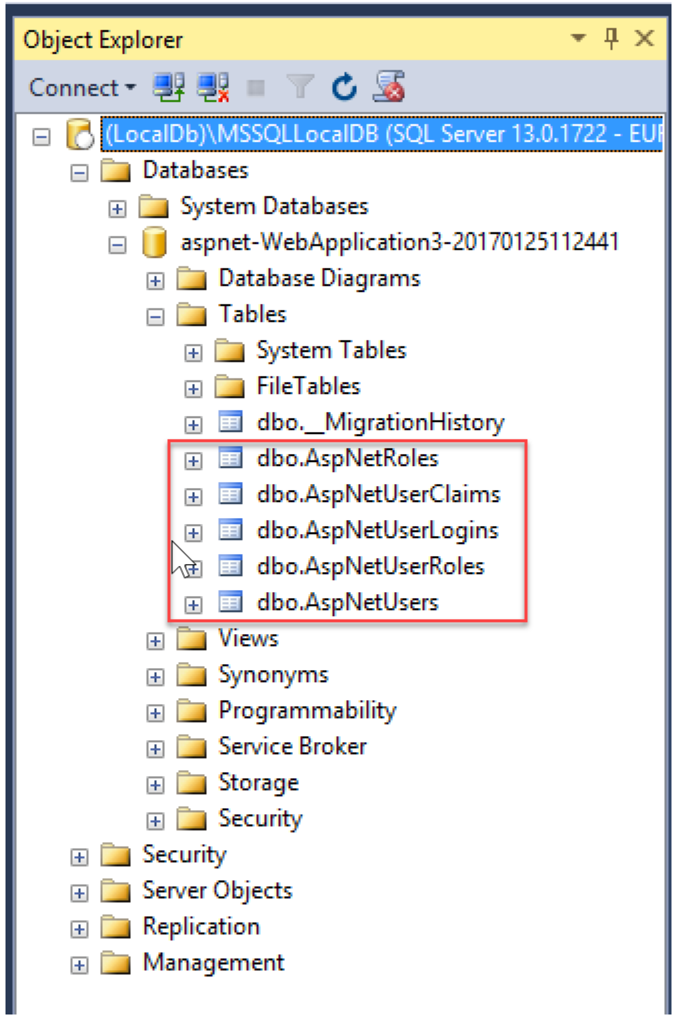

ASP.NET Applications
Managers
e.g. UserManager, RoleManager
Stores
e.g. UserStore, RoleStore
Data Access Layer
Data Source
e.g. SQL Server, MySQL, Windows Azure Table Storage

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").



```
(LocalDb)\MSSQLLocalDB (SQL Server 13.0.1722 - EU
  Databases
    System Databases
    aspnet-WebApplication3-20170125112441
      Database Diagrams
      Tables
        System Tables
        FileTables
        dbo.__MigrationHistory
        dbo.AspNetRoles
        dbo.AspNetUserClaims
        dbo.AspNetUserLogins
        dbo.AspNetUserRoles
        dbo.AspNetUsers
      Views
      Synonyms
      Programmability
      Service Broker
      Storage
      Security
  Security
  Server Objects
  Replication
  Management
```

Authentication with External Providers

- External providers
  - Facebook, Twitter, Microsoft, Google, etc.
- Configuration
  - Application ID
  - Application Secret
  - Website URL
- Storage of App Secret
  - **Do not** store in config file
  - [Best Practice] Azure Key Vault
  - [Best Practice] Application Settings in Azure

Module 9: Security

Section 2: ASP.NET Identity

Lesson: ASP.NET Identity Best Practices

Recommendations

- Utilize Secure Sockets Layer (SSL) 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
- Use Cross-Site Request Forgery (CSRF) tokens everywhere for post methods
- Do not allow for unlimited login attempts
  - Brute forcers dream. Script kiddies abound.
  - Use a slow, strong hashing algorithm (bCrypt)

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
    - Use single sign-on (SSO) with Azure for multi tenant
    - Shared across apps via shared SQL database with identity tables

Module 9: Security

Section 3: Security Threats and Defenses

Lesson: Web Attacks and Defenses

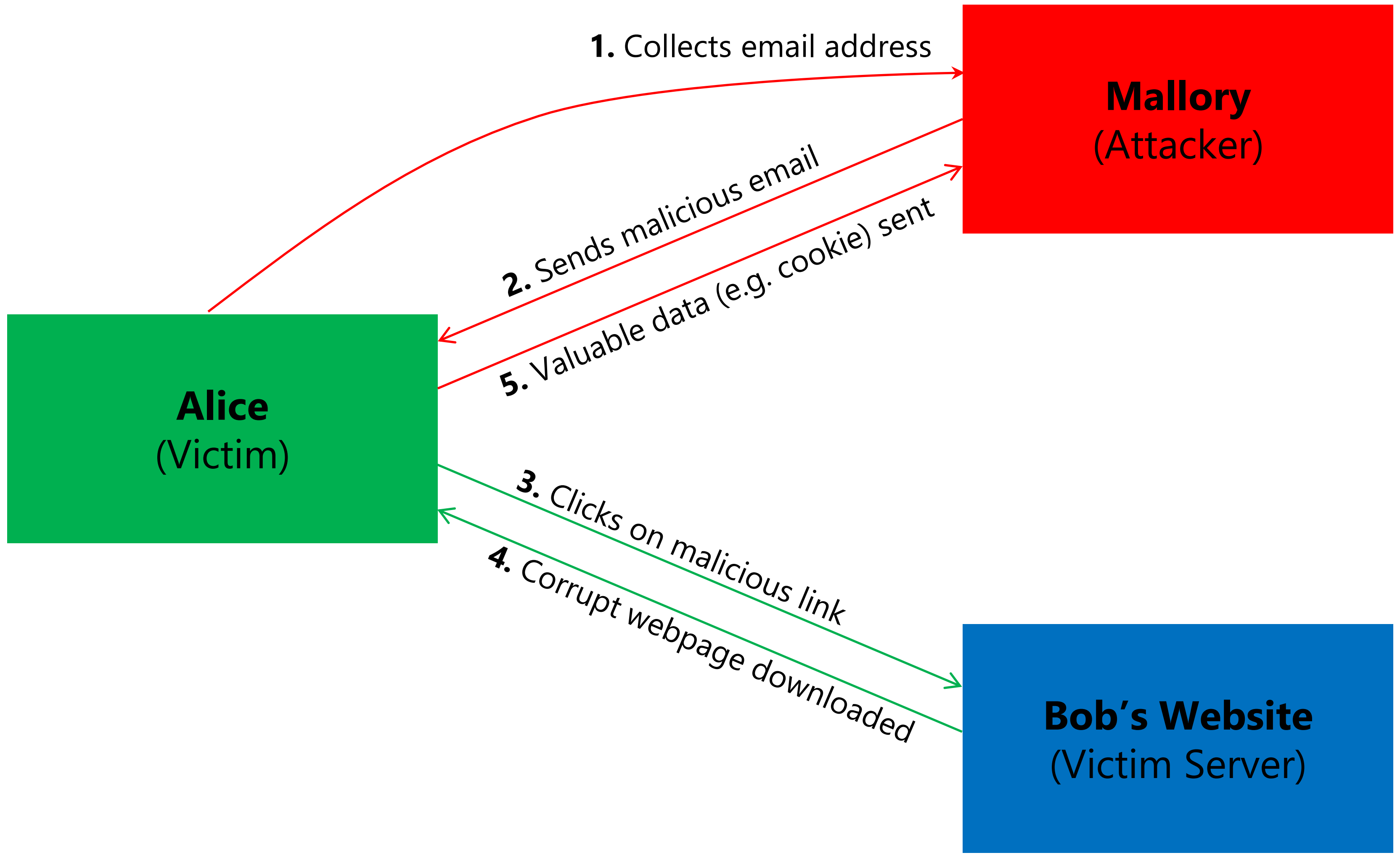
ASP.NET Web Forms vs. ASP.NET MVC

- ASP.NET Web Forms have automated protections
  - ViewState encryption and validation
  - Automatic request and event validation
- ASP.NET MVC provides more granular control over security protections
  - HTML-encoding
  - Request validation
  - Controller/action level authorization
  - Custom security attribute, etc.

Cross-Site Scripting (XSS) Attack

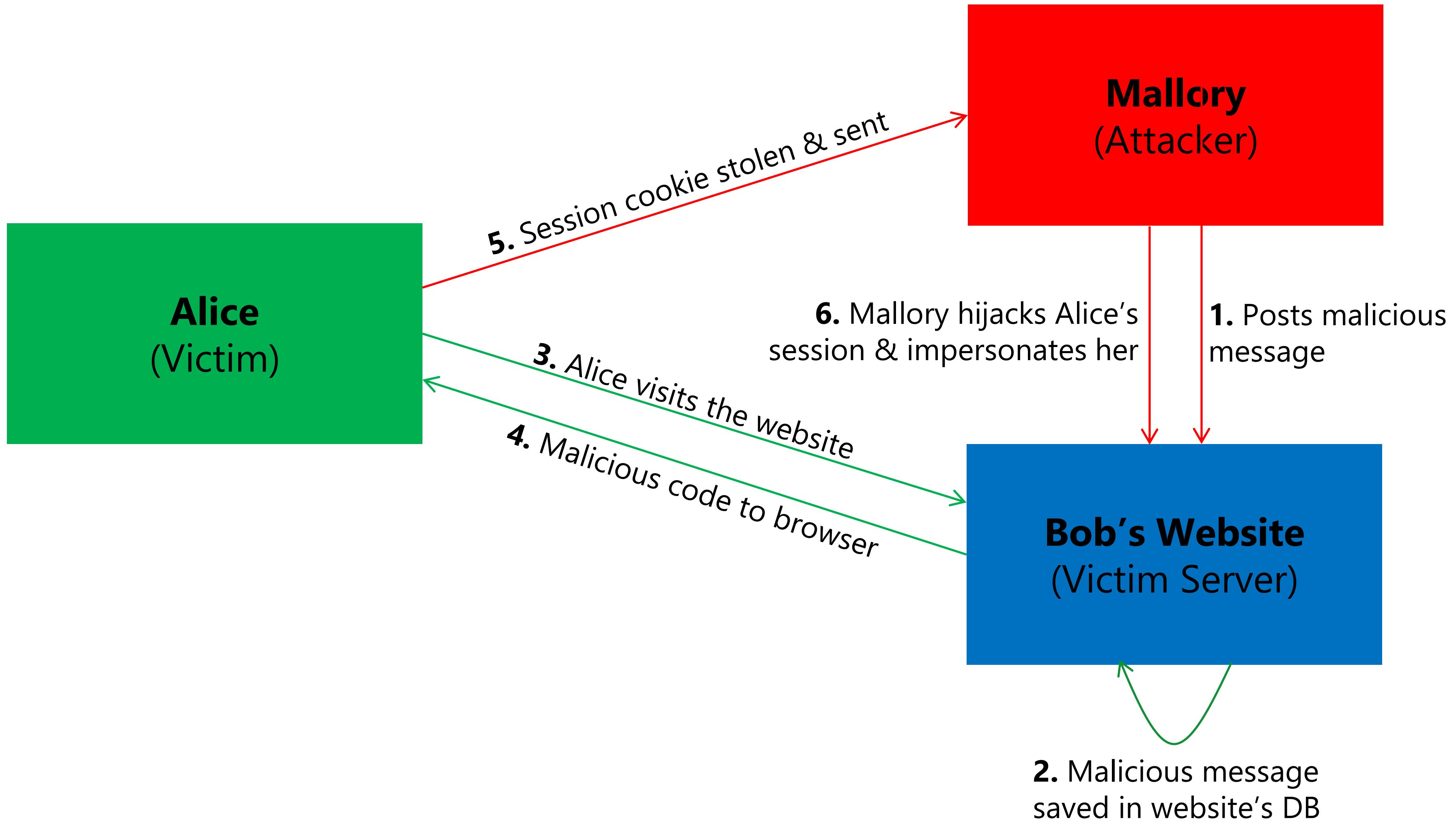
- XSS vulnerability allows an attacker to inject malicious JavaScript into pages generated by a web application
- Malicious script executes in victim client's browser
  - To gain access to sensitive webpage content, session cookies, etc.
- Methods for injecting malicious code:
  - **Active or Reflected 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



1. Collects email address
Mallory
(Attacker)
2. Sends malicious email
5. Valuable data (e.g. cookie) sent
Alice
(Victim)
3. Clicks on malicious link
4. Corrupt webpage downloaded
Bob's Website
(Victim Server)

XSS Stored Attack


Mallory
(Attacker)
Alice
(Victim)
Bob's Website
(Victim Server)
1. Posts malicious message
2. Malicious message saved in website's DB
3. Alice visits the website
4. Malicious code to browser
5. Session cookie stolen & sent
6. Mallory hijacks Alice's session & impersonates her

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
- Use standard security libraries
    - Anti-XSS library

HTML Encoding

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

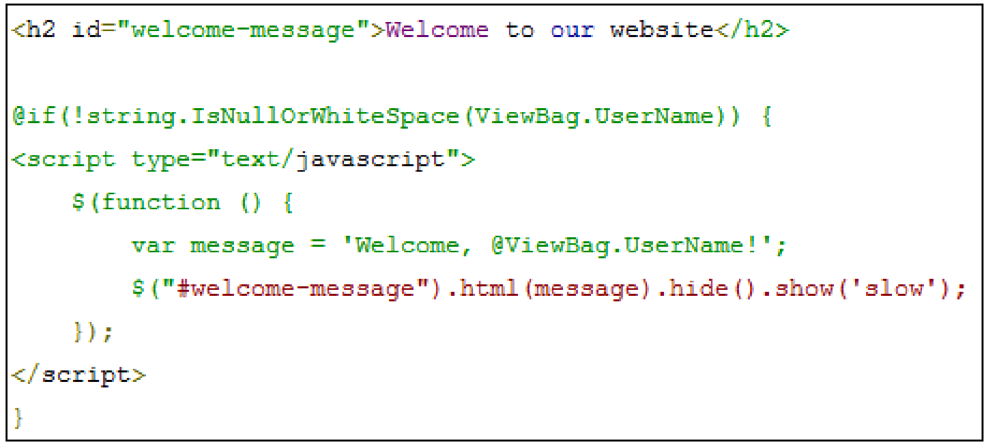
Malicious User Input (without encoding)

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

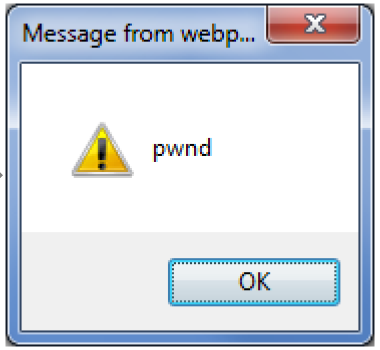
HTML-Encoded User Input

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

JavaScript Encoding



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

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

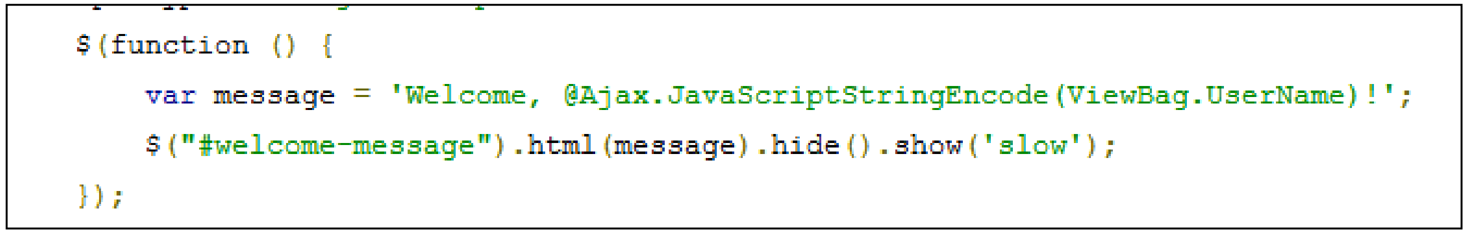


Message from webp...

pwnd

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

JavaScript Encoding Fix

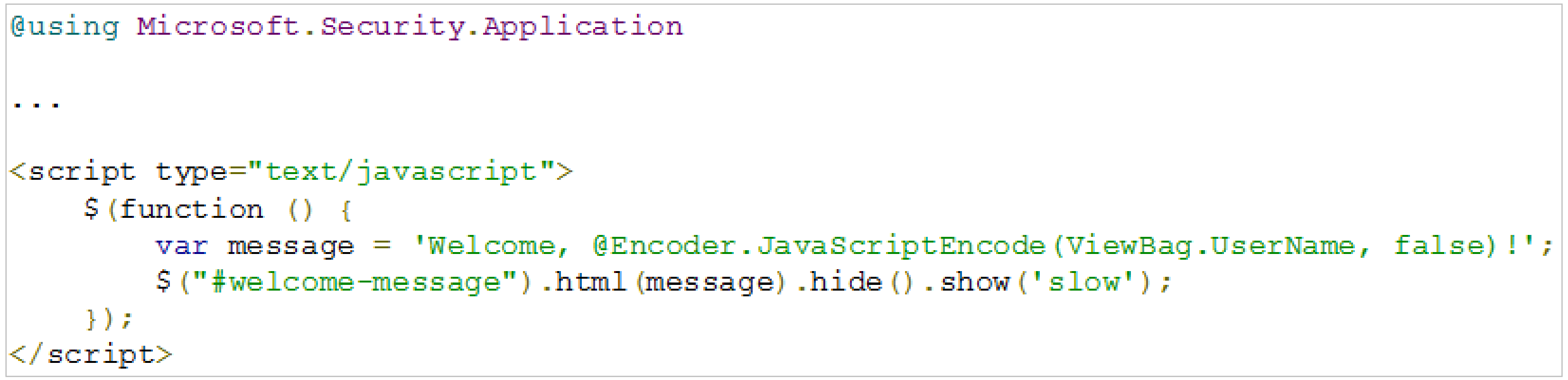


```
    $(function () {
        var message = 'Welcome, @Ajax.JavaScriptStringEncode(ViewBag.UserName)!';
        $("#welcome-message").html(message).hide().show('slow');
    });
```

Anti-XSS Library

- It provides a myriad of encoding functions for user input, including HTML, HTML attributes, XML, CSS and JavaScript
- Whitelist Approach:
    - All characters not on the whitelist are encoded

JavaScript Encoding Fix with Anti-XSS Library



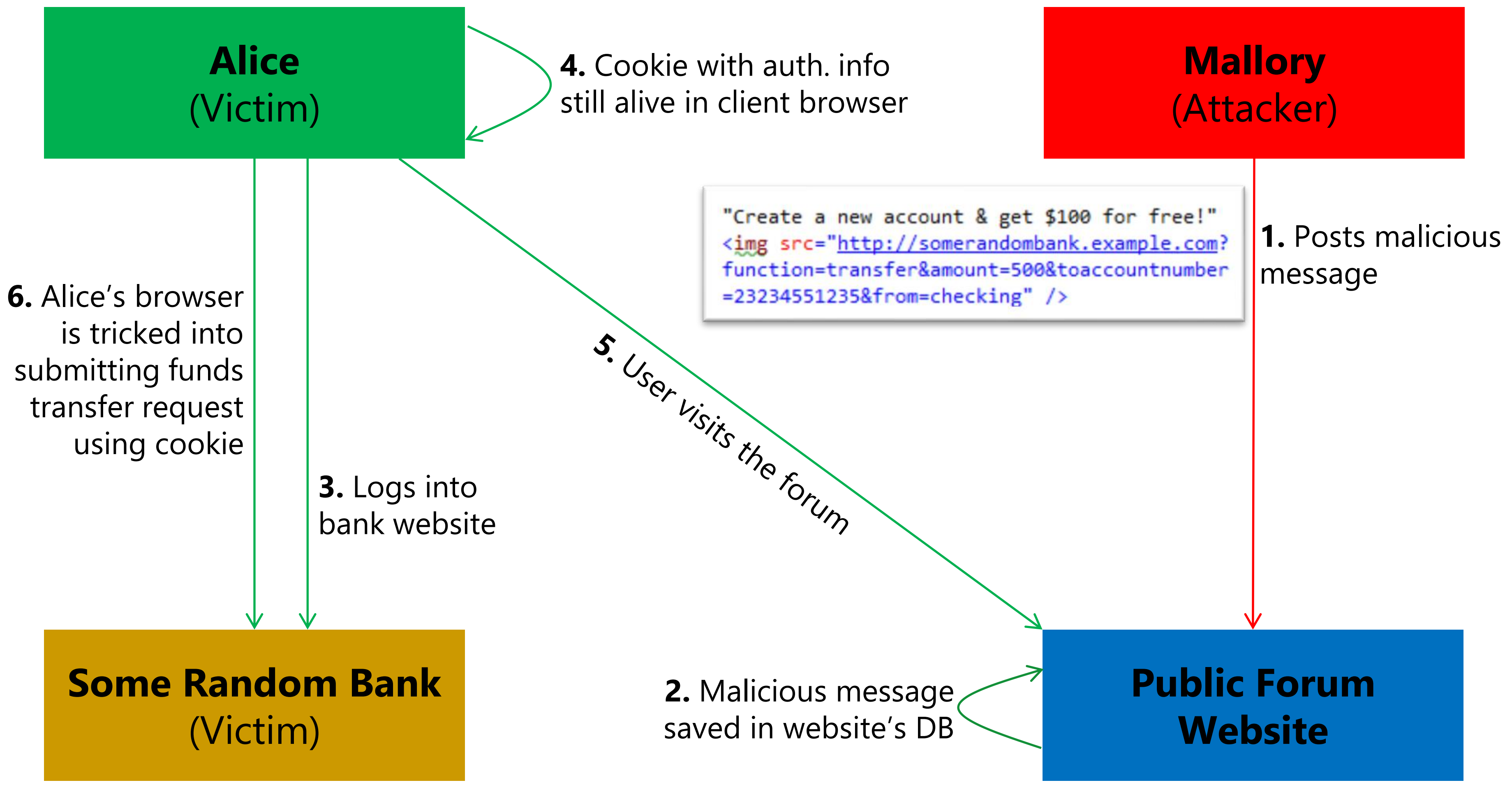
```
@using Microsoft.Security.Application

...

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

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)

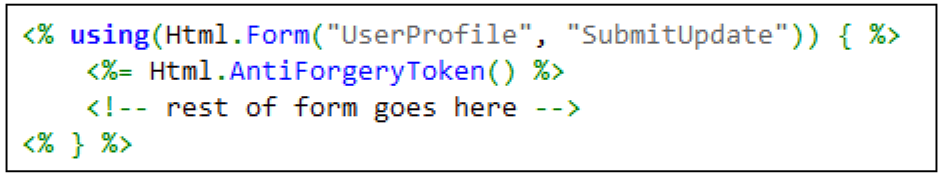


Alice
(Victim)
4. Cookie with auth. info still alive in client browser
Mallory
(Attacker)
1. Posts malicious message
6. Alice's browser is tricked into submitting funds transfer request using cookie
3. Logs into bank website
5. User visits the forum
2. Malicious message saved in website's DB
Some Random Bank
(Victim)
Public Forum Website

```
"Create a new account & get $100 for free!"
<img src="http://somerandombank.example.com?
function=transfer&amount=500&toaccountnumber
=23234551235&from=checking" />
```

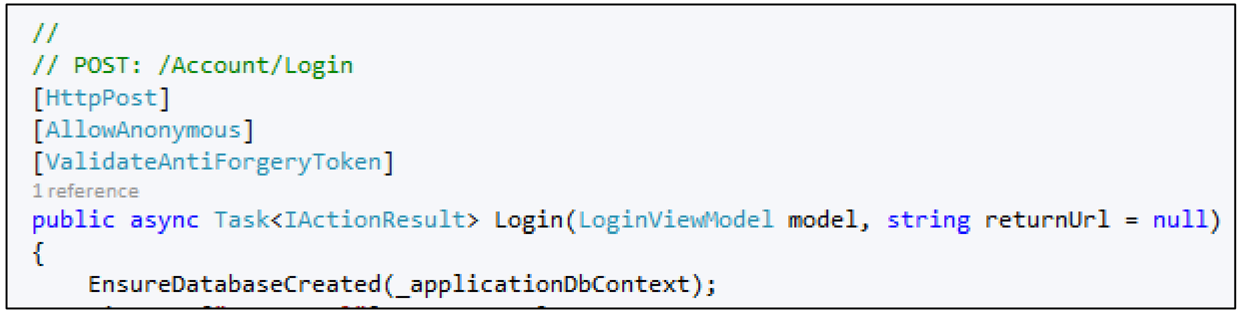
CSRF Defense

- **AntiForgery token**: A hidden form field that is validated when the form is submitted
  - Embed a hidden input into each form request



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

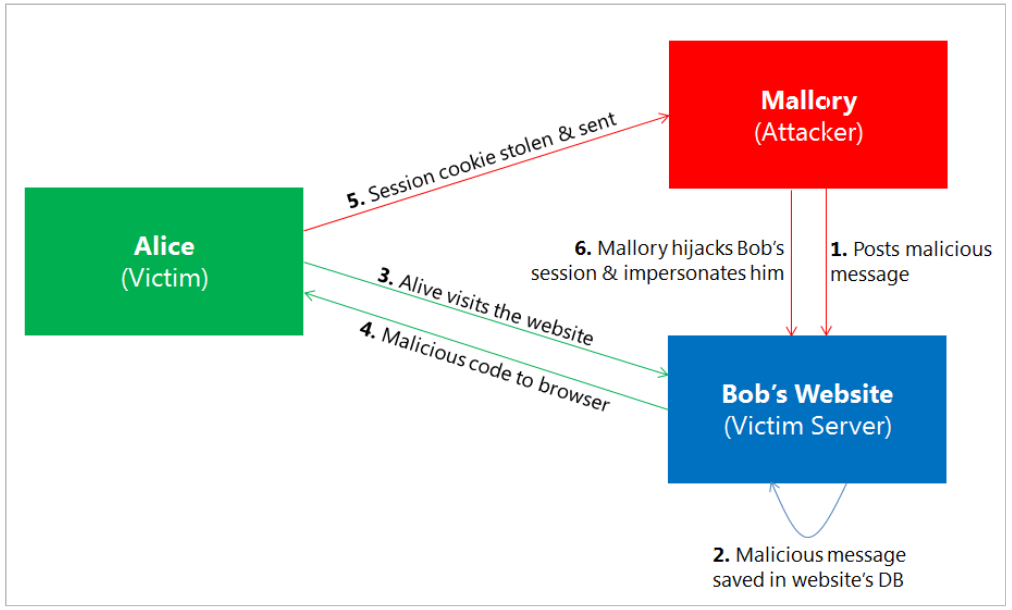
  - Mark the action method(s) to validate AntiForgery token



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

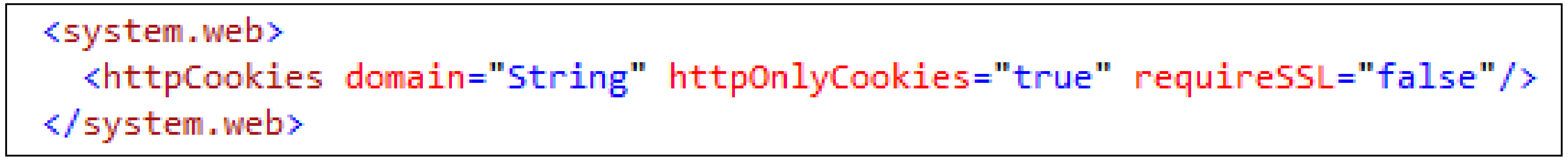
Cookie Stealing Attack

- 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


Mallory
(Attacker)
5. Session cookie stolen & sent
Alice
(Victim)
6. Mallory hijacks Bob’s session & impersonates him
1. Posts malicious message
3. Alive visits the website
4. Malicious code to browser
Bob’s Website
(Victim Server)
2. Malicious message saved in website’s DB

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

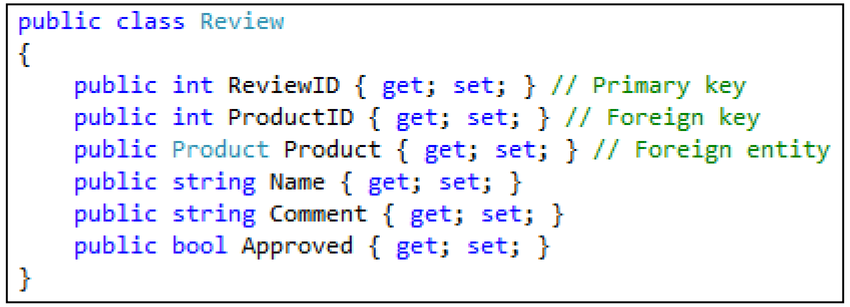


```
<system.web>
  <httpCookies domain="String" httpOnlyCookies="true" requireSSL="false"/>
</system.web>
```

Over-Posting Attack

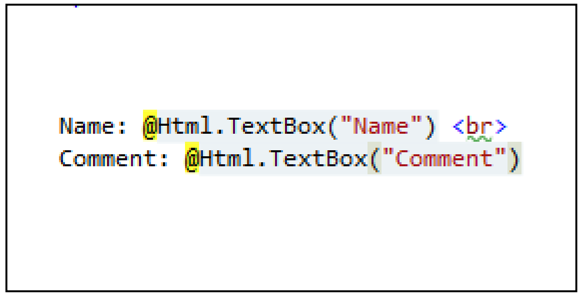
- An attacker can populate model properties that are not included in the View.

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

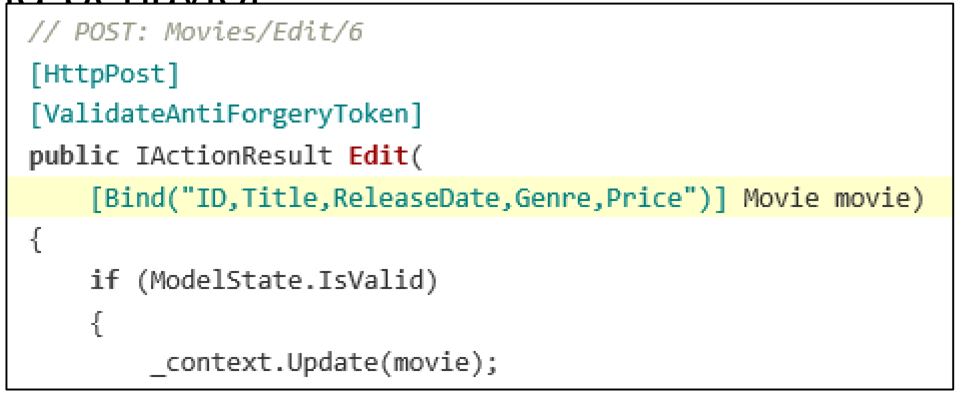


```
Name: @Html.TextBox("Name") <br>
Comment: @Html.TextBox("Comment")
```

- 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.

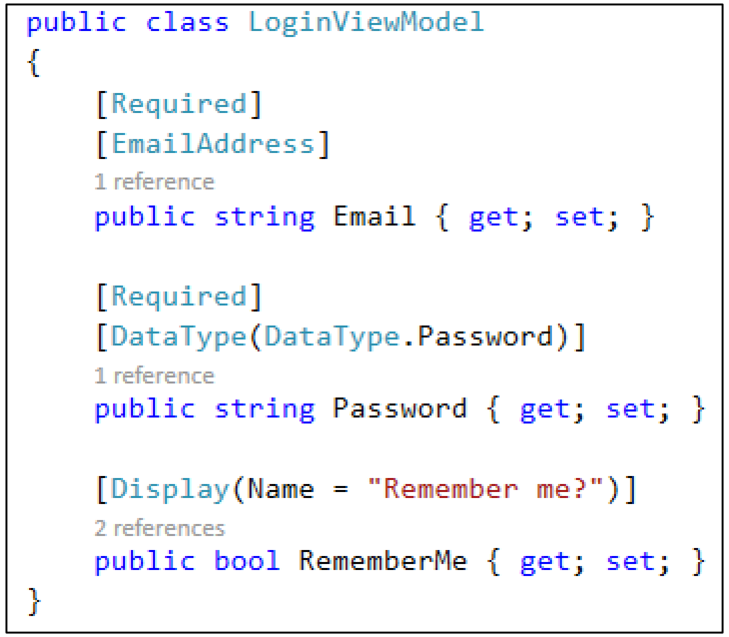
Over-Posting Defense

- Use [bind] attribute to explicitly control the binding behavior
    - Specifically list permitted properties
- Use View Model [recommended]
- Use DTOs to decouple your data objects from your view objects (View Models)



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

[Bind]



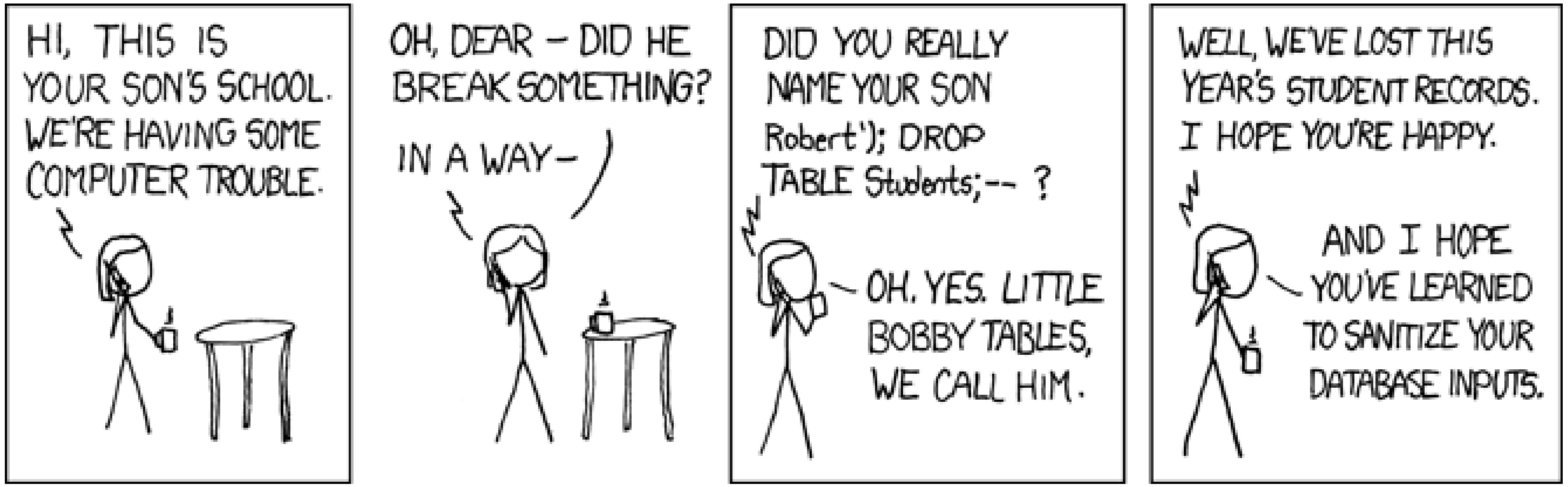
```
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    1 reference
    public string Email { get; set; }

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

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

View Model

Demo: Over-Posting Attack

Injection attack

- Malicious code is inserted into strings that are later passed to an instance of SQL Server (or other databases).
- Malicious code is injected through untrusted input methods to launch other attacks such as LDAP injection, XML injection, Memory exceptions, etc.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/327/

Threat Defense Summary

Threat	Solution
Cross-Site Scripting (XSS)	• HTML-encode all content • JavaScript encoding • Use Anti-XSS if possible
Cross-Site Request Forgery (CSRF)	• AntiForgery token • HTTPReferrer validation
Over-Posting	• Bind attribute; ViewModels
Cookie Stealing	• httpOnly cookies
Injection	• Validate all untrusted input • Use type-safe SQL parameters with stored procs or an ORM • Use parameters collection with dynamic SQL or no dynamic SQL • Use built-in escape routines for special characters • Least-privilege database account • Escape wildcard characters • Avoid disclosing error information • Use secure code recommended practices

Module 9: Security

Section 4: Trending Web Attacks

Lesson: OWASP Top 10

Open Web Application Security Project (OWASP) Top 10 Web Security Attacks (2013)

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

ASP.NET Defenses Against OWASP Top 10 Attacks

1. Injection
   - Use parametrized SQL queries or an ORM
   - Validate all untrusted input
   - Use parametrized APIs
   - Restricted binding of Action methods
   - White-list untrusted data
2. Broken Authentication and Session Management
   - Avoid using custom authentication modules
3. Cross-Site Scripting (XSS)
   - Encode HTML context (body, attribute, JavaScript, CSS, or URL)

ASP.NET Defenses Against OWASP Top 10 Attacks (continued)

4. Insecure Direct Object References
   - Use random-access reference maps for mapping database key with per-user indirect reference
   - Apply server-side access control for client-side calls
5. Security Misconfiguration
   - Apply repeatable hardening process – Application Lifecycle Management (ALM) and DevOps automation
   - Encrypt sensitive sections of config file(s)
   - Update Operating System/web server/.NET framework/third-party libraries
   - Perform random audits of deployment configuration

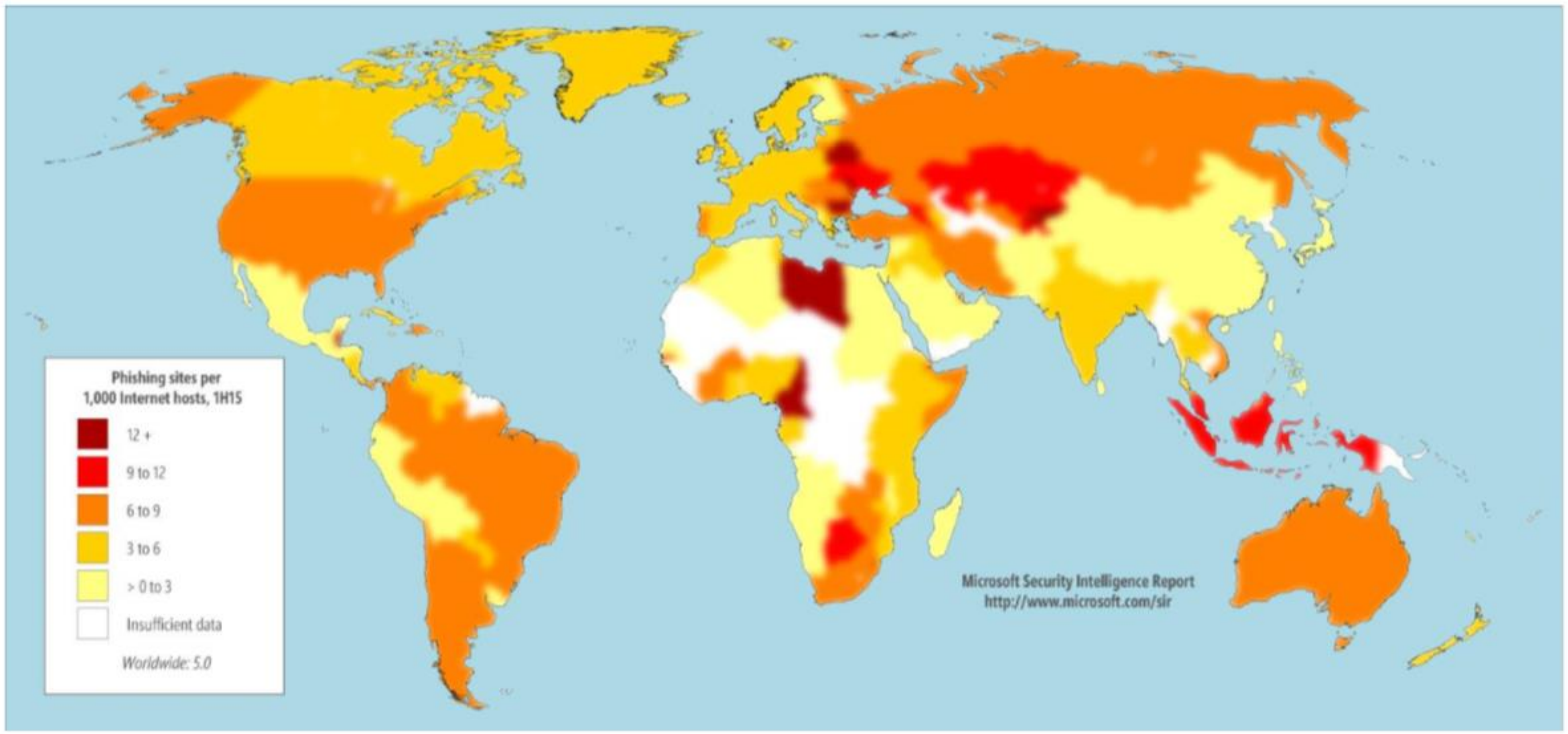
ASP.NET Defenses Against OWASP Top 10 Attacks (continued)

6. Sensitive Data Exposure
    - Use HTTPs
    - Encrypt data stored in application database(s)
    - Use strong encryption and hashing algorithms
    - Disable caching and autocomplete on sensitive forms
7. Missing Function/Method Level Access Control
    - Use ASP.NET Identity and Roles
8. Cross-Site Request Forgery (CSRF)
    - Generate and include the anti-XSRF tokens in all views
    - Validate tokens in controllers

ASP.NET Defenses Against OWASP Top 10 Attacks (continued)

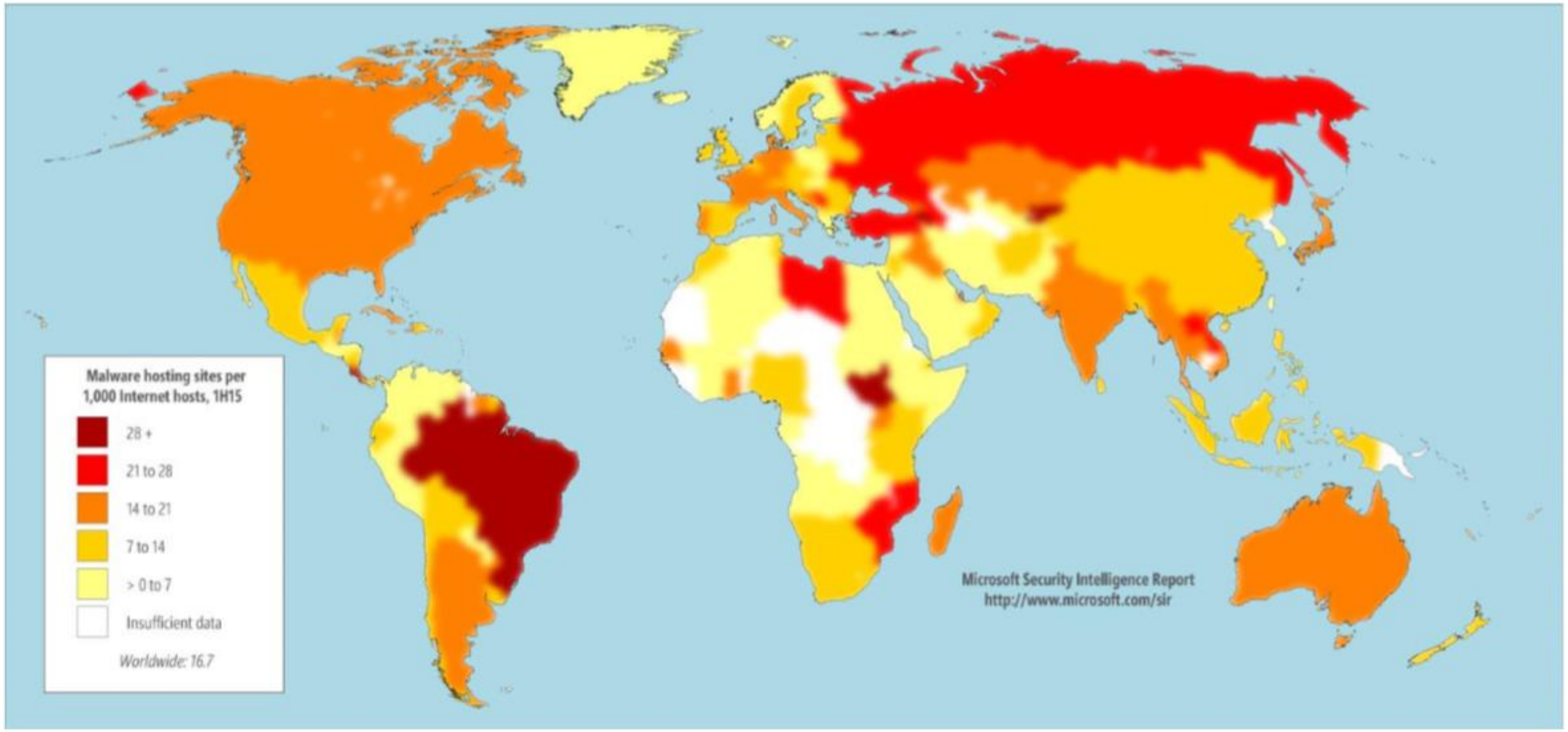
9. Using Components with Known Vulnerabilities
   - Regularly update application components
   - Formulate and enforce effective software security policy in your organization
   - OWASP Safe NuGet package
10. Unvalidated Redirects and Forwards
    - Do not involve user input or parameter in calculating the destination URL
    - If destination parameters are used, verify and authorize them per user

Phishing Sites Per 1000 Internet Hosts (2015 H1)


Phishing sites per
1,000 Internet hosts, 1H15
12 +
9 to 12
6 to 9
3 to 6
> 0 to 3
Insufficient data
Worldwide: 5.0
Microsoft Security Intelligence Report
http://www.microsoft.com/sir

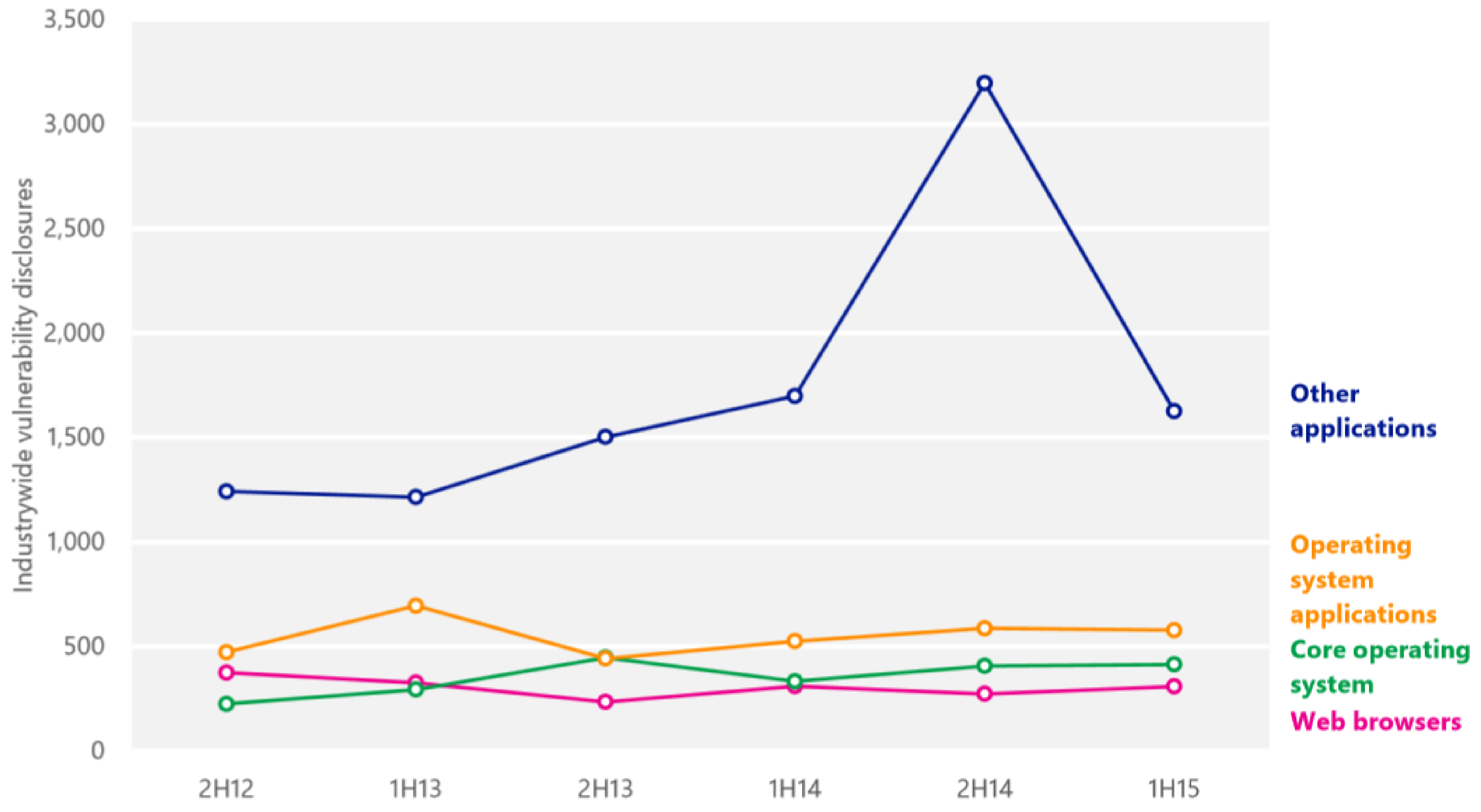
Reference: Microsoft Security Intelligence Report, Volume 19

Global Distribution of Malware Hosting Sites (2015 H1)


Malware hosting sites per 1,000 Internet hosts, 1H15
28 +
21 to 28
14 to 21
7 to 14
> 0 to 7
Insufficient data
Worldwide: 16.7
Microsoft Security Intelligence Report
http://www.microsoft.com/sir

Reference: Microsoft Security Intelligence Report, Volume 19

Vulnerability Trend



Reference: Microsoft Security Intelligence Report, Volume 19

Important Security Questions

- Does the application have different users who are allowed to do different things?
- How certain do we need to be that the user is who she/he claims to be?
- What is the security level required for different parts of the application?
- How to protect sensitive parts of the application?
- How to ensure that authenticated users only do what they are allowed to do?
- What should be done to ensure that only the right people have access to sensitive data?
- How will we detect malicious behavior?
- How long will the application be down after successful attack? What is the contingency plan?

Module Summary

- In this you learned about:
  - Security fundamentals
  - Authentication and authorization
  - ASP.NET Identity
  - Security threats and defenses
  - OWASP Top 10 web attacks
  - Latest web attacks trends

Lab: Security

Microsoft