OWASP Top 10

Showing how MVC, .NET and IIS can solve the low hanging fruit of the OWASP Top 10

James Davis

Overview

- Who is this guy?
- Quick Boot on MVC
- A1 – Injection
- A2 – Broken Authentication
- A3 – Cross Site Scripting (XSS)
- A4 – Insecure Direct References
- A5 – Security Configuration

Overview

- A6 – Sensitive Data Exposure
- A7 – Missing Function Level Access Control
- A8 – Cross-Site Request Forgery (CSRF)
- A9 – Using Components with Known Vulnerabilities
- A10 – Unvalidated Redirects and Forwards
- Wrap Up

Who is this guy?

- James Davis
  - @debugthings
  - LinkedIn
- Programmer for 10 years
  - .NET for 10 years
  - C++ for 5 years
  - Various other languages
- Soul developer for (now defunct) www.elawnguys.com
- Interested in Security and Brewing Beer
  - Working on the upper levels of IO in my spare time (stopped at 27 when the baby came)
  - Brewed a number of batches, love to make gadgets to help

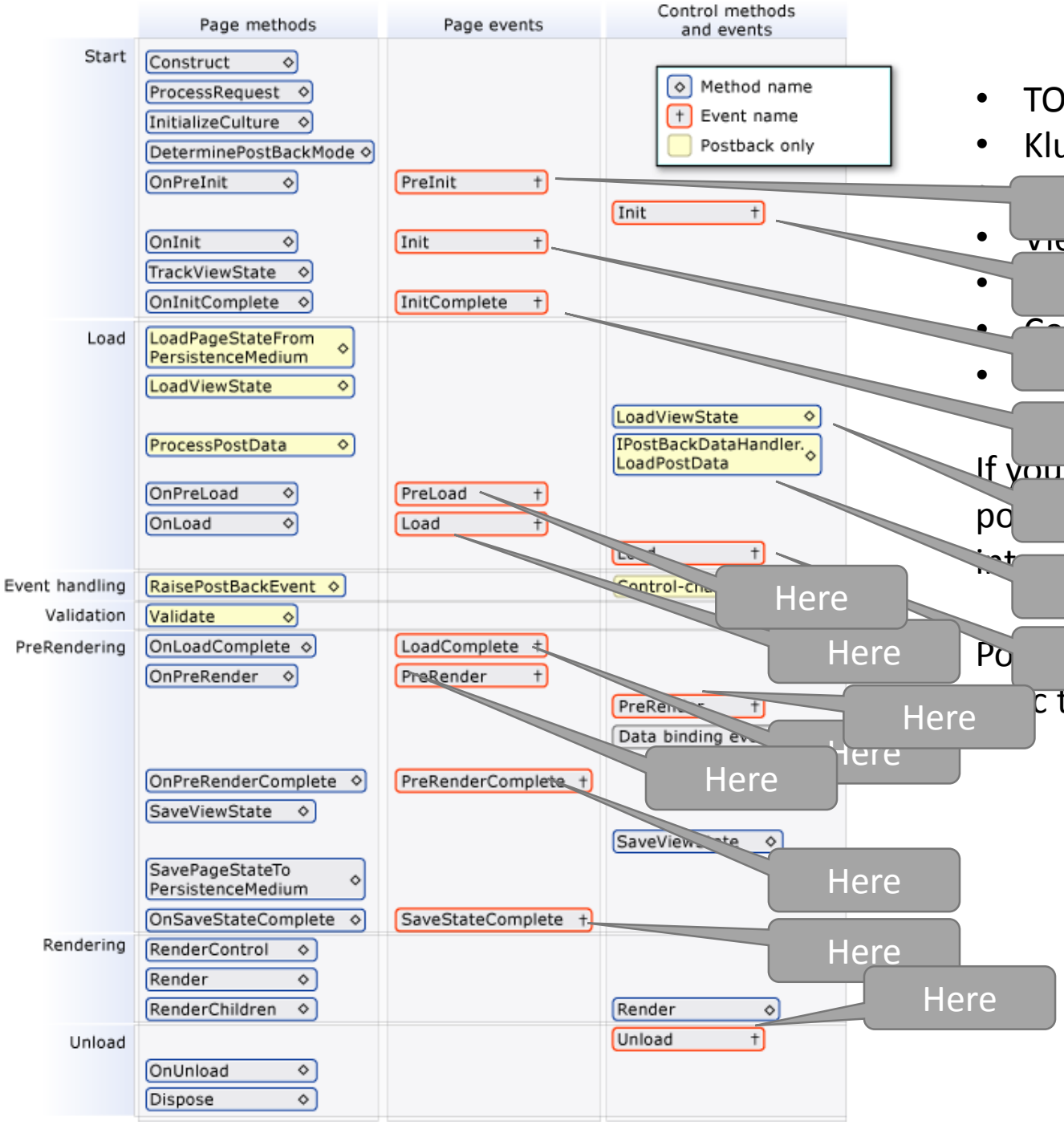
OWASP Site

- The OWASP MVC site is freely available on GitHub
  - https://github.com/jldgit/OWASPTop10Tampa
- All of the code you see displayed is available in the repository
- All of the code is free to use
- I do not guarantee this code will work for you
  - Please use them as guided examples.
- If you have issues running the site, feel free to contact me
  - There is limited documentation on the repository

MVC and ASP.NET Quick Boot

What is ASP.NET?

- Microsoft server side programming platform
  - It is language agnostic (can be used with C# and VB.NET)
- Generally is a module that is run in IIS
  - Executes in an IIS container called a worker process
  - Can be run on Linux with Mono (offshoot of CLI Rotor)
  - Can also be run from a custom container
- It’s versioned along with the main .NET assemblies
- Runs a few popular sites (microsoft.com)
- Primarily used in corporate environments



So, the old way of doing things?

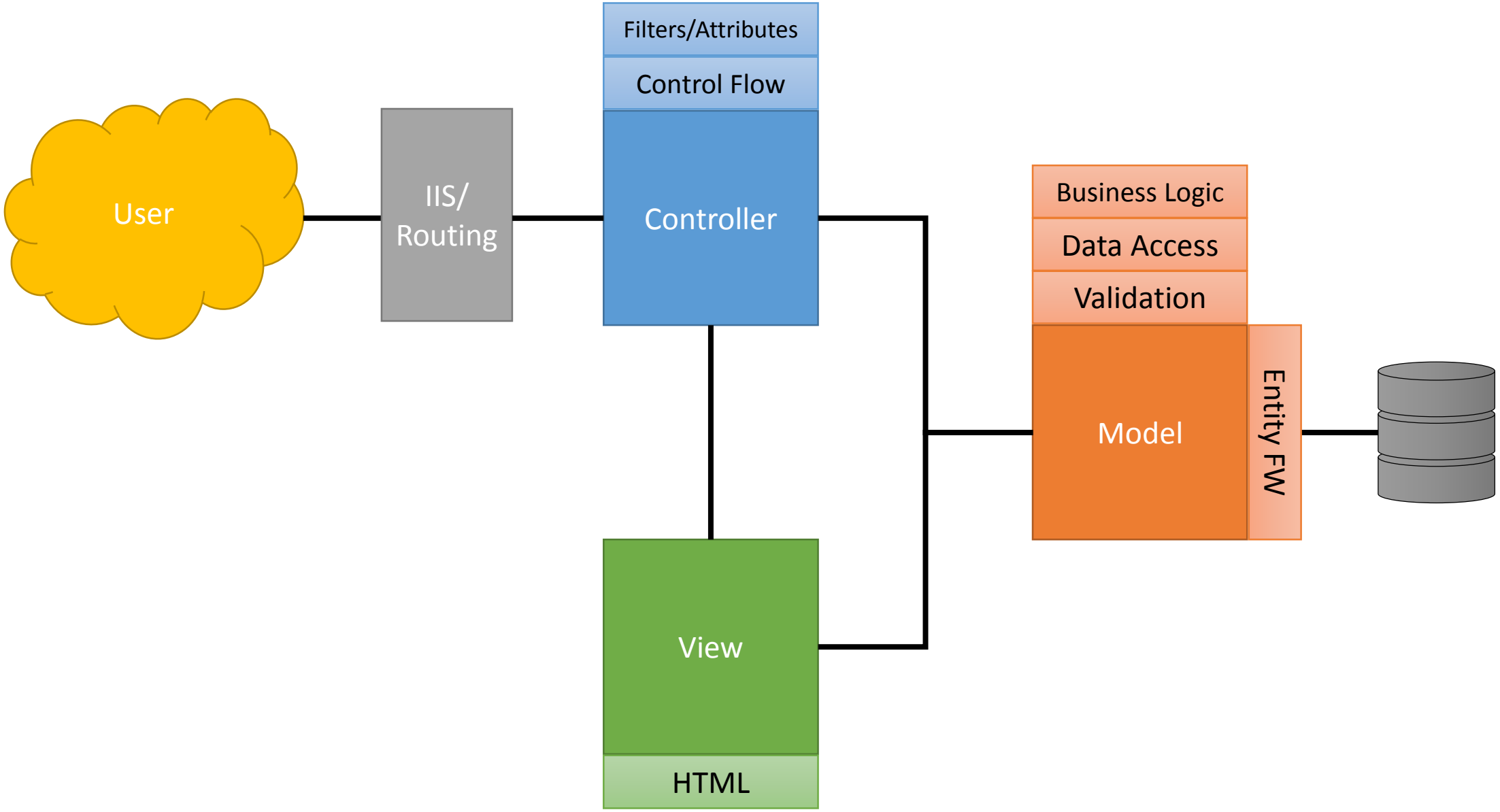
- TONS of events
- Kludgy insertion of data
- ere databound
- ViewStates (Ugh.)
- vas bound per control
- Complicated
- nformation

If you google how to bind data to a control, you get a bunch of po u also get a lot of hacks that--I'm sure--made it
n.

Po e a bit unruly and often required some complex
c to change the course of the page load.

What is MVC?

- MVC is not new. It’s an old pattern revived for .NET (supplanted MVVM)
- First iterations written by Scott Guthrie in 2009
- As of today latest version is 5.2
- Lots of built in goodies to help with security
- Model – Validation Logic, Business Logic, Data Access, Entity Framework
- View – HTML markup, javascript
- Controller – Control Flow, how the data is handled that is sent to the view


User
IIS/
Routing
Filters/Attributes
Control Flow
Controller
View
HTML
Business Logic
Data Access
Validation
Model
Entity FW

Legacy Application Design

- The Beginning (ASP, ASP.NET 1.0, old timers on new systems)
  - Handful of browsers, limited servers
  - A one to one relationship of all things
  - Code lived inline with the ASP and ASP.NET tags
  - Database calls were unchecked*
  - User input was also unchecked*
  - Lots of boilerplate code was needed, FOR EVERYTHING
  - Web pages were really just e-mail forms
  - Data access was limited

Legacy Application Design

- The Change (ASP.NET 2.0 takeover)
  - Everyone fears change
  - SOA became the “thing”
    - The application 7-layer burrito
  - Code is now separated from markup (mostly)
  - Content sourced from more than one place
  - Business developers became web developers
  - Large adoption for in-house
  - Low adoption for external
  - Solve hard problems with a webpage, not a fat client
  - Focus on complexity of systems
  - Little focus on best practices

Modern Application Design

- The Excitement (ASP.NET in full swing, MVC and MVVM)
  - Change is wildly embraced
  - By now code was robust and applications solid
  - WebForms (GOD AWFUL) were on the way out
    - Web development became civilized
  - Focus on UX and functionality
  - New toys to help speed development
  - Social media is big
  - Content comes from dozens of places
  - Best practices become accepted and used
  - Security is getting noticed

Modern Application Design

- The Reality
  - Things really don’t change
  - Legacy code sits every where; it’s the glue you can’t remove
  - WebForms are still around
  - Changing code takes time and money
  - Big companies are still the most prone
  - “It doesn’t face the outside. So, we’re good right?”
  - Business developers are still becoming web developers
  - People still take the easy route
  - People are still taking lessons from headlines

OWASP Top 10

A1 – Injection

Models

- Models provide built in validation for fields and parameters
  - Both for backend and frontend data (jQuery validate)
- Entity Framework provides some safe guards
  - Use Linq or Linq-to-SQL (properly performance tested of course)
- Use of custom model binders provides consistent validation

Controllers

- Controllers also provide a frontline for validation of POST data [ValidateInput]; the default value is true
- Custom Action filters to provide repeatable actions cleanly

Injection – Vulnerable Site

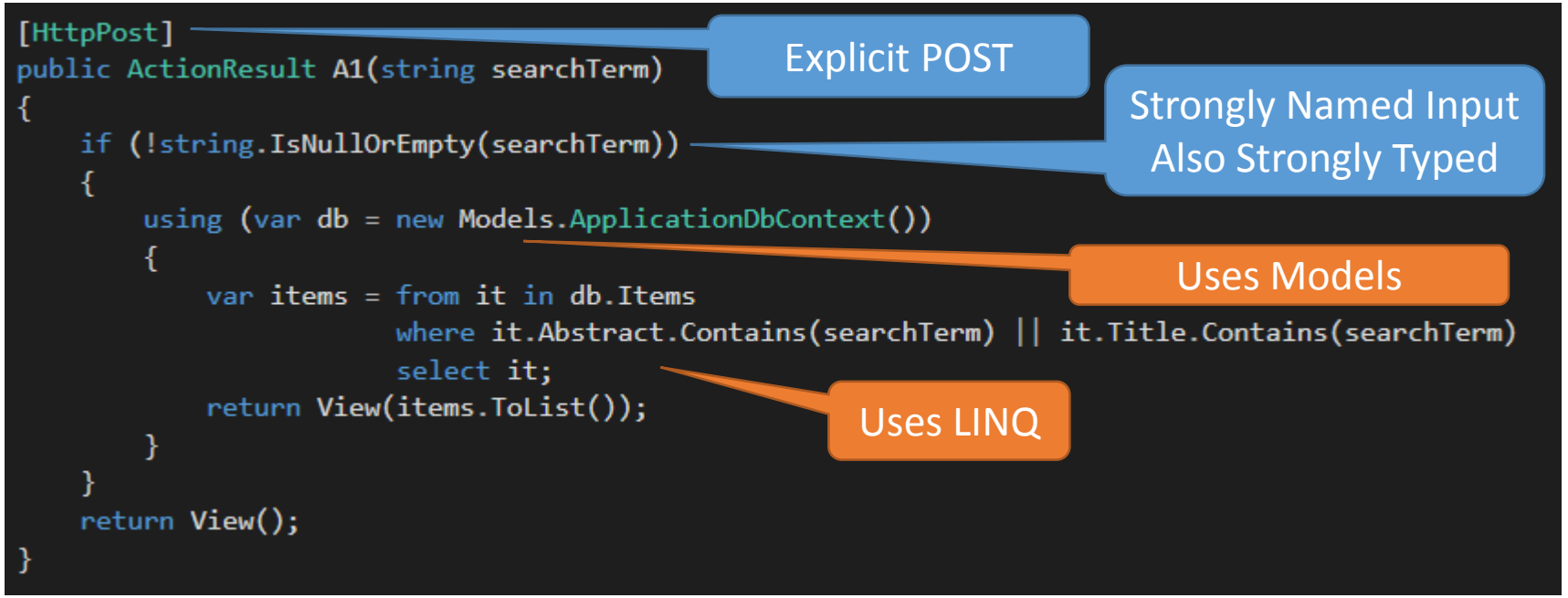

```
[HttpGet]
public ActionResult A1()
{
    if (HttpContext.Request.QueryString.AllKeys.Contains("searchTerm"))
    {
        string rawTerm = HttpContext.Request.QueryString["searchTerm"];
        using (var db = new System.Data.SqlClient.SqlConnection(
            ConfigMgr.ConnectionStrings["DefaultConnection"].ConnectionString))
        {
            // This can be broken with x' UNION ALL SELECT Email, PhoneNumber FROM AspNetUsers --
            using (var cmd = new System.Data.SqlClient.SqlCommand(
                string.Format(@"SELECT Title, Abstract
                                FROM Items
                                WHERE Title LIKE '%{0}%' OR Abstract LIKE '%{0}%'",
                rawTerm), db))
            {
                db.Open();
                System.Data.DataTable dt = new System.Data.DataTable();
                System.Data.SqlClient.SqlDataAdapter da = new System.Data.SqlClient.SqlDataAdapter(cmd);
                da.Fill(dt);
                ViewBag.SearchResults = dt;

            }
        }
    }
```

By using a raw query string and string concatenation we leave this vulnerable to an obvious injection.

What is more subtle is the fact that we allow search terms in the query string by only accepting a GET verb.

See my note on the next slide about using POST

Injection – Secure Site



*Note on using GET v. POST. There is no hard and fast rule to use either. In the spirit of MVC and clean URLs, using POST will get rid of the ?searchTerm= in the URL.

There is no inherent safety associated with using POST other than using the binding mechanisms that are available when using MVC. It also requires the attacker to use specific toolsets to mangle the HTTP request. These may not always be available from the attack location.

By making a few changes and leveraging the MVC pattern we build in an inherent type safety.

This allows for cleaner, more secure, and more maintainable code as your product grows.

A2 – Broken Authentication

Controllers

- No session information passed via query string
  - Use of cookie based ASP.NET session
  - Use web.config to enforce timeout rules
- Custom Filter to change session ID after logon

Extras

- Facebook, Twitter, Google, Microsoft?
  - Use of Oauth and these identity providers is recommended to keep your data secure
  - Built in support for things like 2 factor authentication
  - There’s a NuGet pacakage for that!

Peek at web.config for session

```
<system.web>
  <authentication mode="None" />
  <compilation debug="true" targetFramework="4.5.1" />
  <httpRuntime targetFramework="4.5.1" requestValidationMode="2.0" />
  <sessionState mode="InProc"
                cookieless="UseCookies"
                timeout="15" />
</system.web>
```

Storage Location

Default Value

Time in minutes that session will be active

The session timeout is perpetual. This means that it will timeout in the interval from the last request the session is being used.

See the code examples on creating a filter that will handle a hard timeout.

A3 – Cross Site Scripting (Server-Side)

Views

- Views default to HTML encoding.
- Built in helper methods
  - Html.Encode – Uses HTML Escaping >
  - Url.Encode – Uses URL Encoding %3E
  - Ajax.JavaScriptStringEncode – Uses JavaScript Escaping \u003e

Controllers

- If you are accepting data from a user you can also use these methods to encode from the Action; beware as you may end up double encoding
  - HttpUtility.UrlEncode
  - HttpUtility.HtmlEncode
  - HttpUtility.JavaScriptStringEncode

Extras

- Microsoft AntiXSS Library
  - Uses whitelisting and is updated regularly

Cross Site Scripting – Vulnerable Site

Disables all Validation

```
[HttpPost]
[ValidateInput(false)]
public ActionResult A3(Models.Comments comment)
{
    using (var db = new Models.ApplicationDbContext())
    {
        var apUser = from user in db.Users where user.UserName == User.Identity.Name select user;
        comment.ApplicationUserId = apUser.Single();
        comment.CommentDate = DateTime.Now;
        db.Comments.Add(comment);
        db.SaveChanges();
        var models = from mod in db.Comments select mod;
        return View("OWASP/A3", new List<Models.Items>());
    }
}
```

This set of pages is getting better as we can see that we're using Models. However the problem lies with the disabling of ALL validation for this model.

In the View we can also see that we're displaying RAW data to the user. This is un-escaped HTML data and will be inserted into the HtmlWirter() as is.

Improper Use of Url.Raw()

```
@foreach (var item in Model)
{
    <tr>
        <td class="col-md-2">
            @Html.DisplayFor(modelItem => item.CommentDate)
        </td>
        <td class="col-md-10">
            @Html.Raw(item.Comment);
        }
        </td>
    </tr>
}
```

Cross Site Scripting – Secure Site

```
public class CommentsModelBinder : IModelBinder
{
    public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
    {
        var comment = new Comments()
        {
            // Check to see if we've turned validation off
            Comment = controllerContext.Controller.ValidateRequest
            ? GetSanitizedValue(controllerContext, "Comment") : GetContextValue(bindingContext, "Comment")
        };
        return comment;
    }
```

Custom Model Binder

Use of a Model Binder and AntiXSS for both the Model and the View will ensure that you do not take in bad data as well as you will not display bad data.

```
private string GetSanitizedValue(ControllerContext controllerContext, string key)
{
    var result = Microsoft.Security.Application.Sanitizer.GetSafeHtmlFragment(
        controllerContext.HttpContext.Request.Unvalidated.Form[key]);
    return (result == null) ? null : result;
}
```

```
@foreach (var item in Model)
{
    <tr>
        <td class="col-md-2">
            @Html.DisplayFor(modelItem => item.CommentDate)
        </td>
        <td class="col-md-10">
            @Html.Raw(Microsoft.Security.Application.Sanitizer.GetSafeHtmlFragment(item.Comment));
        </td>
    </tr>
}
```

Use of AntiXSS Library

A4 – Insecure Direct Object References

http://www.mysite.com/4567/234/safe

Routing Engine

- The Routing Engine has a few tricks to help keep you safe
  - You can define the types of data accepted by a route as well you can even do (limited) validation and parsing of data before it hits your controller

Models

- The real meat of this vulnerability is taken care of when you use Models.
  - You can create tight relationships between your data and authenticated users

Controllers

- Use of the [Authorize] class attribute (Filter)
  - You can extend IAuthorizationFilter to create a custom authorization check for a page

Views

- **Proper logic and partial pages can hide sensitive areas**

Insecure Direct Object References – Vulnerable Site

```
routes.MapRoute(
    name: "A4Vuln",
    url: "vuln/A4/{id}",
    defaults: new { controller = "Vulnerable", action = "A4", id = UrlParameter.Optional }
);
```

```
public ActionResult A4(string id)
{
    ViewBag.ItemRequested = id;
    return View("OWASP/A4");
}
```

Generic Route

Generic Action

Both the generic Route and the generic Action lend to sloppy references.

Insecure Direct Object References – Secure Site

```
routes.MapRoute(
  name: "A4Normal",
  url: "sec/A4/{id}",
  defaults: new { controller = "Secure", action = "A4", id = UrlParameter.Optional },
  constraints: new { id = @"\d+"}
);

routes.MapRoute(
   name: "A4Sec",
   url: "sec/A4/USER/{id}",
   defaults: new { controller = "Secure", action = "A4User", id = UrlParameter.Optional },
   constraints: new { id = @"USER\d+"}
);

routes.MapRoute(
   name: "A4SecAdmin",
```

```
public ActionResult A4(string id)
{
    ViewBag.ItemRequested = id;
    return View("OWASP/A4");
}

public ActionResult A4User(string id)
{
    ViewBag.ItemRequested = id;
    return View("OWASP/A4");
}

[Authorize(Roles = "Admin")]
public ActionResult A4Admin(string id)
{
    ViewBag.ItemRequested = id;
    return View("OWASP/A4");
}
```

Specific Routes

Specific Authorization

Specific Actions

Using specific routes and specific actions together can help you control the users and role allowed to access specific areas of your application.

A5 – Security Misconfiguration

Non MVC Specific

- Be aware that your default application pool runs as IUSR which is a local (least privileged) account
- If you are in a AD environment create a non human account
  - You can revoke access at any time and also lock down more by policy
- Use Integrated security when possible for SQL connections
- IIS leaves anonymous access on by default; you have to explicitly turn it off
- Beware of the FullTrust trap
  - Not everything needs full trust. Sometimes you just need to create a specific security policy for your assemblies
- Try to use only signed assemblies from trusted sources

A6 – Sensitive Data Exposure

Controllers

- The most useful attribute for protecting sensitive data is [RequireHttps]
  - The name implies the obvious. It only allows HTTPs for a specific Controller or Action
- Disable output caching for secure pages
  - [OutputCache(NoStore = true)]
- Create secure cookies whenever possible
  - HttpCookie.Secure = true
  - HttpCookie.Sharable = false
  - HttpCookie.HttpOnly = true

Views

- Remove comments from markup
  - Use server side comments **@* *@**

IIS

- Turn off detailed exceptions and errors
- Turn off headers that identify your backend technology

Sensitive Data Exposure – Vulnerable Site

Allowing non HTTPS
Allowing
Client Side Cache

```
[Authorize]
public class VulnerableController : Controller
{
```

Not protecting
sensitive cookies

```
public ActionResult A6()
{
    if (!HttpContext.Request.Cookies.AllKeys.Contains("SecureBankInfo"))
    {
        HttpContext.Response.AppendCookie(new HttpCookie("SecureBankInfo")
        {
            Value = "RTNUM|0123456789"
        });
        HttpContext.Response.AppendCookie(new HttpCookie("UserToken")
        {
            Value = "TOK-987654-GRANT-ADMIN"
        });
    }
    return View("OWASP/A6");
}
```

Exposing
Sensitive Info in
Comments

```
13  <!-- We are storing our info client side for
14      transfer between non federated areas. This code below ... -->
15  <script>
16      function getCookie(name) {
17          var value = "; " + document.cookie;
18          var parts = value.split("; " + name + "=");
19          if (parts.length == 2) return parts.pop().split(";").shift();
20      }
```

Sensitive Data Exposure – Secure Site

```
[Authorize] // Must be logged in
[RequireHttps] // Must use https
[OutputCache(Duration = 0, NoStore = true)]
public class SecureController : Controller
{
```

HTTPS Only
No Caching

```
public ActionResult A6()
{

    if (!HttpContext.Request.Cookies.AllKeys.Contains("SecureBankInfo"))
    {
        HttpContext.Response.AppendCookie(new HttpCookie("SecureBankInfo")
        {
            Value = "RTNUM|0123456789",
            Secure = true,
            Shareable = false,
            HttpOnly = true
        });
        HttpContext.Response.AppendCookie(new HttpCookie("UserToken")
        {
            Value = "TOK-987654-GRANT-ADMIN"
            Secure = true
```

Protecting
sensitive cookies

Using Server
Side Comments

```
@* We are storing our info client side for
    transfer between non federated areas. This code below ... *@
<script>
    function getCookie(name) {
        var value = "; " + document.cookie;
        var parts = value.split("; " + name + "=");
        if (parts.length == 2) return parts.pop().split(";").shift();
    }
```

A7 – Missing Function Level Access Control

Controllers

- This is where the use of the [Authorize] filter shines
- You can use [Authorize] to limit by Roles or Users
  - [Authorize(Roles = “Admin”)]
  - This is text based and uses the roles provider
  - Can be applied to the Controller or the Action

Missing Function Level Access Control – Vulnerable Site

```
public ActionResult Admin()
{
    return View("OWASP/Admin");
}
```

Zero Protection

Only Validates User is Real

```
[Authorize]
public class VulnerableController : Controller
{
```

Almost no protection is on the Admin function.

The only protection available is the Authorize filter which just validates that a user (any user) can access this area.

Missing Function Level Access Control – Secure Site

```
[Authorize(Roles = "Admin")]
public ActionResult Admin()
{
    return View("OWASP/Admin");
}
```

Only Allows the Admin Role

Check the User Role Inside of the View

```
@if(User.IsInRole("Admin"))
{
    @Html.ActionLink("Admin Portal", "Admin");
}
```

Attach a User to a Model Item

```
public class UserBoundItem
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Display(Name = "User")]
    public virtual ApplicationUser ApplicationUserId { get; set; }
    [Display(Name = "Secure Secret")]
    public string Secret { get; set; }
}
```

Just by adding the Roles attribute to the Authorize filter we can limit an area to a specific user role. Or even a set of user roles.

By adding in simple checks in your views you can avoid displaying secure areas by mistake or omission.

A8 – Cross-Site Request Forgery

Controllers and Views

- Use [ValidateAntiForgeryToken] in conjunction with HtmlHelper.AntiforgeryToken (@Html.AntiForgeryToken)
  - Built in, works great.
  - By default this is a hidden form parameter and a cookie; this is the safest way to handle this.
- Create a filter that invalidates the session if they do not have a proper referrer header
- Try to use post parameters wherever possible to modify data
  - [HttpPost] attribute will limit the verbs to POST only

Cross-Site Request Forgery - Secure

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public asyn  Task<ActionResult> Login(LoginViewModel model, string returnUrl)
{
```

Require POST verb

Use of Anti-Forgery Token

```
@using (Htm  ginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl },
    FormMeth  .Post, new { @class = "form-horizontal", role = "form" }))
{
    @Html.AntiForgeryToken()
```

Custom Filter Attribute to Check the Referrer Header

```
public class ReferrerRedirectAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        if (filterContext.RequestContext.HttpContext.Request.UrlReferrer.Host.ToLower() != "localhost")
        {
            filterContext.Result = new RedirectToRouteResult(
                new System.Web.Routing.RouteValueDictionary(new { controller = "Home", action = "Index" }));
        }
    }
}
```

The Anti-Forgery Token built into MVC is very easy to use and helps prevent a number of attacks.

However, you don't have to stop there. You can extend a custom filter to check the origin of the page request as well.

A9 – Using Components with Known Vulnerabilities

- It's Microsoft, so, you know... Patch Tuesday
- Turn off services and components you don’t need
  - FTP (CVE-2010-3972) – Denial of Service (IIS7)
  - FastCGI (CVE-2010-2730) – Remote Code Execution (IIS 7 and 7.5)
  - MVC 2 – 5.1 (CVE-2014-4075) ☺ -- XSS Vulnerability
- Beware of using third party applications from an untrusted source
  - Use NuGet
  - If you can get the code, look it over and compile it yourself
  - Lock down permissions for 3rd party code, start at the most restrictive and turn it up if needed.

A10 – Unvalidated Redirects and Forwards

Controllers, Views and Routing

- Clever use of Views can help here for internal redirects
  - The old way of doing this was a Server.Redirect or Server.Transfer
  - The new way is just to return a View
  - Returning a View will not cause a redirect but instead change the content of the page
  - You can even tell your action to return another action; this has the added benefit of using the security filters you assign
  - You can validate the URL with the Url.IsLocalUrl() method
- Routing and Controllers also can help here
  - If you are taking in a parameter to redirect you can use your route to send it to a more secure/robust Action to determine if the location is really yours

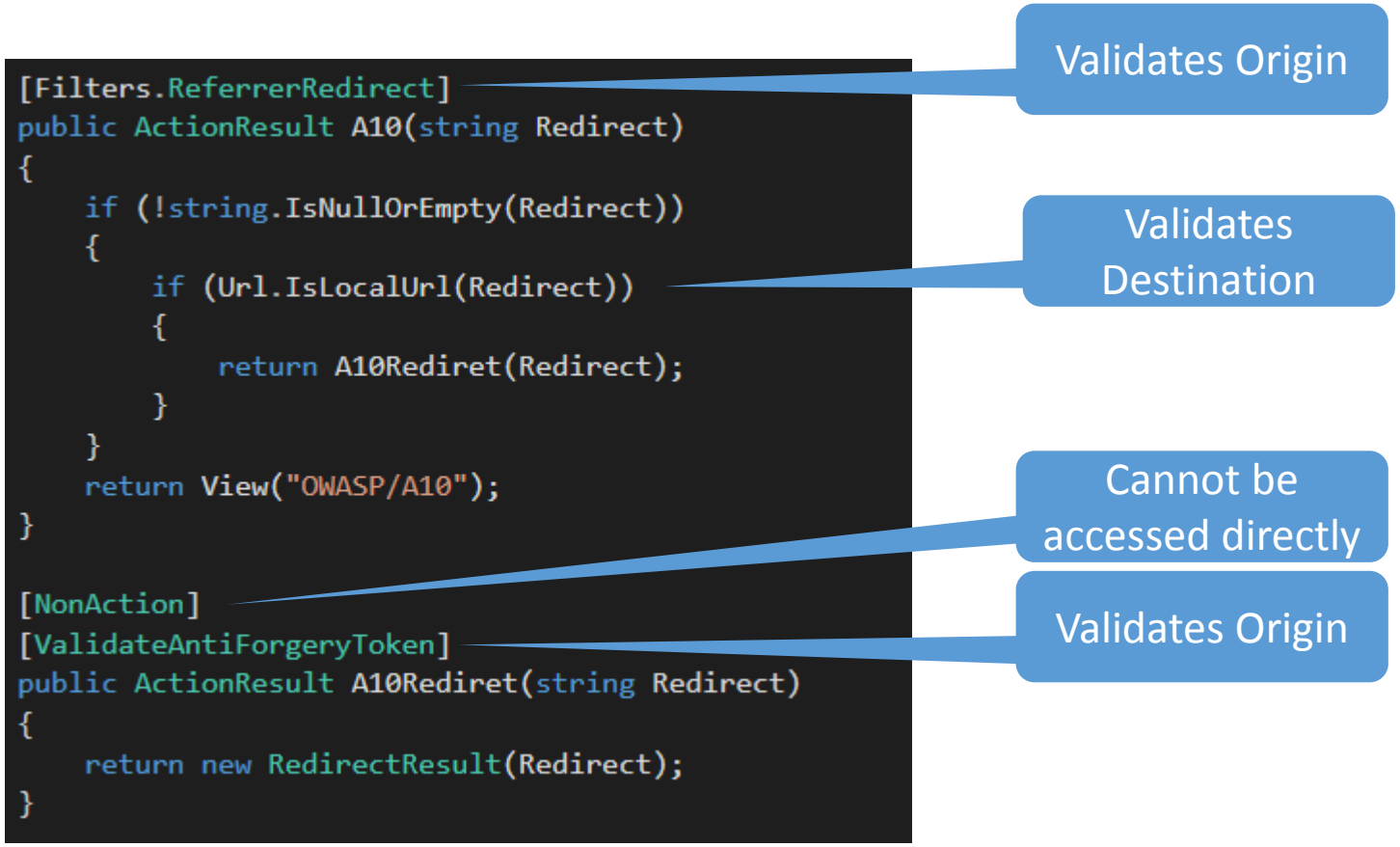
Unvalidated Redirects and Forwards – Vulnerable Site

```
public ActionResult A10(string Redirect)
{
    if (string.IsNullOrEmpty(Redirect))
    {
        return View("OWASP/A10");
    }
    return new RedirectResult(Redirect);
}
```

Raw Redirect

This contrived example shows that we're taking in a string parameter, whether it's a query string or POST parameter and blindly redirecting to the destination.

Unvalidated Redirects and Forwards – Secure Site



Here we can see that we're using the referrer action from A8 in conjunction with the Url.IsLocalUrl.

On the A10Redirect() method, you can see the use of the NonAction filter. This tells MVC that this is just a function and not something that can be navigated to.

This page has the potential for an infinite loop if we included a redirect query string along with the input.

In a nutshell

- Use Entity Framework and Models to simplify and secure data access
- Use available tools for XSS and Injection attacks
  - HtmlHelper library
  - Microsoft XSS Library (WPL)
- Limit types of input by using the routing engine
- Make use of built in security filters
  - [Authorize]
  - [RequireHttps]
  - [ValidateAntiForgeryToken]
- Properly configure other filters to be secure
  - [OutputCache(NoStore = true)]
- Write your own extensions to perform custom actions
  - Customer filters extended from FilterAttribute
  - Custom HTML Helpers
- When in doubt, check NuGet

Questions and Comments