ASP.NET Security

Scott Guthrie
ASP.NET Team
Microsoft Corporation

Agenda

- Overview
- ASP.NET Security Concepts
  - Process Identity
  - Authentication
  - Authorization
  - Role-Based Security
  - Encryption
- Defending against Common Web Hacks
  - Client Side Script Injection Attacks
  - SQL Injection Attacks

ASP.NET Security Concepts

- Process Identity
  - NT Account server code *runs under*
- Authentication
  - *Identifying username identity* of a client
- Authorization
  - *Controlling access* of an identified user
- Role Based Security
  - Organizing identities into *custom groups* and controlling access by those groups
- Encryption
  - *Protecting traffic* between server and client

Process Identity

Process Identity

- Process Identity refers to the Windows Account that your server code is running under
  - “ASPNET” account default on Win2000 and XP
  - “Network Service” account default under Win2003
- Recommendation:
  - Give process account as few permissions as possible (ex: ASP.NET can’t by default write to files)
  - Strongly recommend keeping the out of the box process default process identity unless you have a very good reason to change it

Setting Process Identity

- ASP.NET on Win 2003 enables per-application process identities (configured via app pools)
  - Each application can run under unique account
  - Easily configured via IIS MMC Admin Tool
- ASP.NET on Win 2000 and Win XP enables per-machine process identity (shared for all apps)
  - Can enable per application impersonated identity – but worked process identity shared for all apps
  - Process account configured in machine.config file
  - ASPNET_SetReg.exe allows the machine.config process username/password to be encrypted (new feature with ASP.NET V1.1)

Authentication

Authentication

- Authentication is the process of identifying and verifying “who is” a visiting browser
  - Example: REDMOND\scottgu
  - Example: scottgu@microsoft.com
  - Example: puid:8934839938439839843
- Three built-in authentication options:
  - Windows Authentication
  - Forms Based (Cookie) Authentication
  - Microsoft Passport Authentication
- You can create your own modules for custom authentication approaches

Authentication Code

- Application security code the same regardless of authentication mode used
  - "User" component provides same OM
  - "Request.IsAuthenticated" property

```
‘ Output custom welcome message to user
If (Request.IsAuthenticated = true) Then
    WelcomeMsg.Text = “Hi “ & User.Identity.Name
End If
…….
<asp:label id=“WelcomeMsg” runat=server/>
```

Windows Authentication

- Authenticates usernames/passwords against NT SAM or Active Directory
  - Ideal for *Intranet* security scenarios
- Credential resolution handled directly by browser/server
  - NTLM (under the covers)
  - Basic/Digest dialog pop-up
- User.Identity.Name returns NT account:
  - DOMAIN\username: REDMOND\scottgu

Windows Authentication

- Enable windows authentication by placing web.config file in app root:

```
<!– Application’s Root Web.Config File -->

<configuration>
    <system.web>
        <authentication mode=“Windows”/>
    </system.web>
</configuration>
```

Windows Authentication Demo

Forms Authentication

- Utilizes html based sign-in login form to prompt users for username/password
  - Login page UI completely customizable
- Username/password store flexibility
  - Can be stored anywhere, including database
- Ideal for *Internet* scenarios
  - Works with any browser and any OS
  - Doesn't require any NT accounts on server

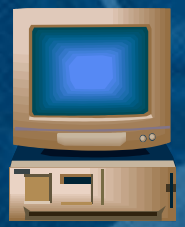
How Forms Authentication Works

1. HTTP GET securepage.aspx
2. HTTP 302 Redirect
   Location: login.aspx
3. HTTPS POST login.aspx
   <form data containing credentials>
5. HTTP 200 Status OK
   Set-Cookie: .ASPXAUTH Auth Ticket
6. HTTP GET securepage.aspx
   Cookie: .ASPXAUTH Auth Ticket

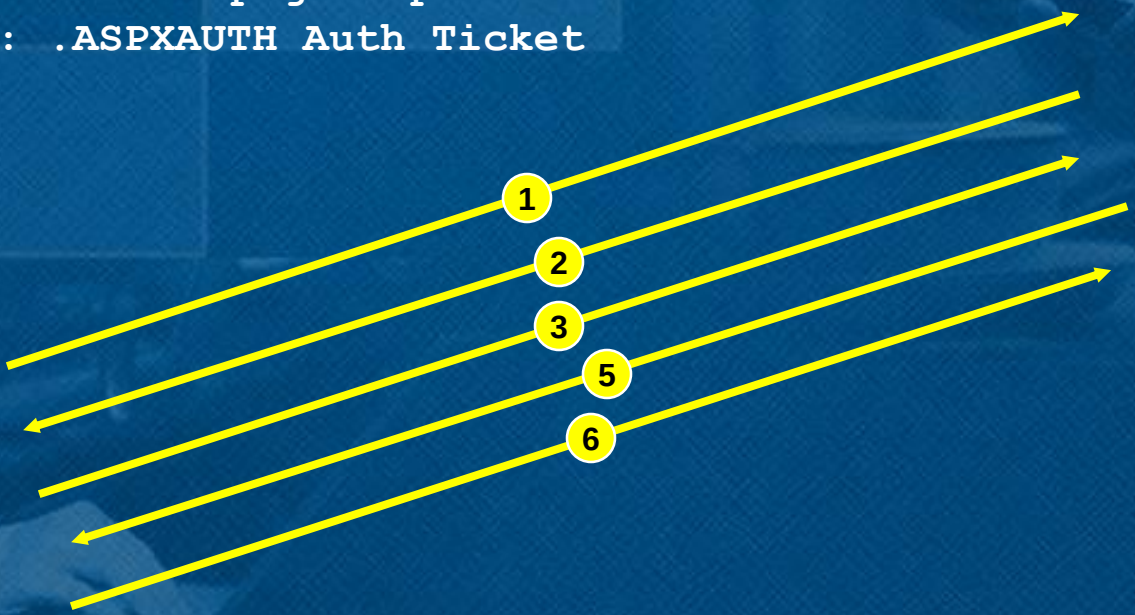
Database

4. App authentication

IIS/
ASP.NET



Web Browser



Implementing Forms Auth

- Developer Steps:
  - 1) Configure Web.Config for Forms auth
  - 2) Write your Login page
  - 3) Implement password check in login page

Forms Auth Web.Config

```
<configuration>
    <system.web>
        <authentication mode="Forms">
            <forms name=".MyAppCookieName"
                    loginUrl="login.aspx"
                    protection="all"
                    timeout="30"
                    requireSSL="false"
                    slidingExpiration="true"
                    path="/"  />
        </authentication>
    </system.web>
</configuration>
```

Forms Auth Web.Config

- Consistent machine keys must be set for web farm scenarios

```
<configuration>
    <system.web>
        <machineKey validationKey="autogenerate"
                    decryptionKey="autogenerate"
                    validation="SHA1" />

                    <!– Validation = [SHA1|MD5|3DES] -->

    </system.web>
</configuration>
```

Writing A Login Page

- 1) Provide your Custom HTML UI
  - Typically have textboxes + checkboxes
- 2) Login button event handler
  - Validate username/password however you want (database call, AD call, etc)
- 3) Call ASP.NET APIs to:
  - Issue authentication cookie
  - Redirect to original URL

FormsAuthentication Class

- RedirectFromLoginPage Method
  - After authentication, redirects back to original request URL
- GetAuthCookie Method
  - Retrieves the authentication cookie (doesn't add it to the outgoing response)
- SetAuthCookie Method
  - Appends the authentication cookie to the outgoing response (no redirect)

Forms Authentication Demo

.NET Passport Authentication

- Single sign-in across member sites
  - No separate usernames/passwords required
  - Large installed based: 165 million users today
  - Built-in support within Windows XP
  - Ideal for *Internet* security scenarios
- Integrated into ASP.NET authentication
  - Requires Passport SDK installation
- More details at http://www.passport.com

Custom Web Authentication

- Application.AuthenticateRequest event
  - Implemented in Global.asax or
  - Http Module (implement IHttpModule)
- Scenarios:
  - Custom SOAP authentication
  - Non-cookie forms auth for mobile devices
  - Customize forms authentication

Authorization

Authorization Strategies

- 1) Windows Security & ACLs
  - ACLs checked for Windows authentication
  - Independent of impersonation
- 2) URL Authorization
  - Imperative “allow” or “deny” tags
  - Supports non-Windows accounts
  - Easy XCopy Deployment Solution
- 3) Custom Authorization
  - Role your own (database calls, etc)

Using URL Authorization

- Example: deny user “fred”, allow users “scott” and “mary”

```
<configuration>
    <system.web>
        <authorization>
            <deny users=“fred”/>
            <allow users=“scott”/>
            <allow users=“mary”/>
        </authorization>
    </system.web>
</configuration>
```

Using URL Authorization

- Example: deny user “fred”, allow all other users

```
<configuration>
    <system.web>
        <authorization>
            <deny users=“fred”/>
            <allow users=“*”/>
        </authorization>
    </system.web>
</configuration>
```

Using URL Authorization

- Example: deny anonymous users (force authentication to take place)

```
<configuration>
    <system.web>
        <authorization>
            <deny users="?"/>
            <allow users="*"/>
        </authorization>
    </system.web>
</configuration>
```

Using URL Authorization

- Example: force authentication only on the Checkout.aspx page

```
<configuration>
   <location path="Checkout.aspx">
      <system.web>
         <authorization>
            <deny users="?"/>
            <allow users="*"/>
         </authorization>
      </system.web>
   </location>
</configuration>
```

URL Authorization Demos

Custom Web Authorization

- Application.AuthorizeRequest event
  - Implemented in Global.asax or
  - Http Module (implement IHttpModule)
- Scenarios:
  - Implement per-request billing system
  - Restrict access based on time of day or other custom parameters
  - Restrict access based on behaviors (e.g. implement a per-day access limit, etc).

Role Based Security

Custom Roles

- Role based security allows application devs to define custom identity groups
  - Roles not tied to NT domain groups
  - Examples: “Brokers”, “SalesPeople”, “Admins”, “VP”, “Premium”, “Partners”
- Enables more flexible authorization of resources and code than per user checks
  - Declaratively through Web.Config
  - Through code: User.IsInRole method
- Goal: Application administrators can modify role members once app deployed
  - No code or configuration changes required

Defining Roles

- Roles are specified programmatically using Application_Authenticate event
  - Implemented in Global.asax or
  - Http Module (implement IHttpModule)

```
‘ Global.asax Authenticate Event Handler

Sub Application_Authenticate(Sender as Object, E as EventArgs)
    Dim roles() as String  = GetRolesFromMyDB(User.Identity.Name)
    Context.User = new GenericPrincipal(User.Identity, roles)
End Sub
```

Authorizing against Roles

- Roles can be used to grant/deny access within Web.Config files:

```
<configuration>
    <system.web>
        <authorization>
            <allow roles="Admins"/>
            <allow roles="Premium"/>
            <deny users="*"/>
        </authorization>
    </system.web>
</configuration>
```

Roles and Code

- User.IsInRole() method can be used to check roles within code at runtime

```
‘ Restrict who can make expensive purchase

If ((amount < 10000) Or (User.IsInRole(“VP”)) Then
    ‘ Do purchase
Else
    Throw New Exception(“You require VP expense approval!”)
End If
```

Role Based Security Demo

Encryption

Encryption

- ASP.NET supports wire encryption of network traffic using SSL through IIS
  - https://www.foobar.com/login.aspx
- Request.IsSecureConnection
  - Indicates whether request is SSL based
- System.Security.Cryptography
  - .NET Namespace provides cryptographic encoding/decoding of arbitrary data

Encryption

- **Recommendations:**
  - **Use SSL when passing username/ password credentials over the web**
  - **Encrypt or one-way hash passwords stored within databases (secures in event of DB penetration)**
  - **Never store secrets or passwords in clear text – use framework to encrypt within a secret store (example: DAPI)**

Common Web Hacks

Client Side Script Injection

- Very common hacking technique used on the web today
- Hacker Technique:
  - Find place on website where input is taken from users, and then redisplayed on a page
  - Provide client-side script for input, unless developer html encodes it on the server, the script will execute when redisplayed
- Note: All web applications (PHP, ASP, JSP and ASP.NET) susceptible to this

CSS Injection Example

```
<script language="VB" runat="server">

    Sub Page_Load()
        Label1.Text = "Hello " & Request.QueryString("name")
    End Sub

</script>

<html>

    <body>

            <asp:label id="Label1" runat="server/>

     </body>

</html>
```

```
Home.aspx?name=<script>alert('Gotcha!');</script>
```

Client Side Script Injection

- Prevention Techniques:
  - HtmlEncode all inputs from the browser
  - Server.HtmlEncode(input)
  - HttpUtility.HtmlEncode(input)

- ASP.NET V1.1 ValidateRequest feature
  - Enabled by default in ASP.NET V1.1
  - Detects and raises error when some common CSS attacks are passed to server
  - Still use HtmlEncode in addition though!

Client Side Script Injection Demo

SQL Injection Attacks

- Very dangerous hacking technique – leads to data loss/corruption/penetration
- Hacker Technique:
  - Find place on website where input is taken from users (not necessarily redisplayed)
  - Assume input is being used in a database operation, try to escape out of a developer's late-bound database query and cause alternative query to be executed
- Note: All web applications (PHP, ASP, JSP and ASP.NET) susceptible to this

SQL Injection Example

```
<script language="VB" runat="server">

  Sub Page_Load()

      Dim connection As SqlConnection
      Dim command As SqlCommand
      Dim query As String

      query = "SELECT * from Products Where QtyInStock > " & Request ("qtyinstock")

      connection = New SqlConnection(ConfigurationSettings.AppSettings("products"))
      command = New SqlCommand(query, connection)

      connection.Open()

      DataGrid1.DataSource = command.ExecuteReader()
      DataGrid1.DataBind()

      connection.Close()

   End Sub

</script>
```

SQL Injection Prevention

- Always, Always, Always use type-safe SQL parameters for data access -> no lazily constructed SQL statements
- Use stored procedures for data access and avoid dynamic SQL statements
  - Make sure you use parameters when calling the SROCS or still be susceptible to attacks!
  - Disable dynamic SQL statement in DB – require all access through SPROCs you write
  - Limit the ASP.NET account to only have access to the SPROCs it needs

SQL Injection Demo

Summary

- Security is a critical feature of every app
  - Design and incorporate it up front
  - Always be vigilant about potential attacks
- ASP.NET provides a rich and flexible security architecture
  - Built-in support for common scenarios
  - Flexible enough for custom adapting

Additional Resources

- Online Discussion Groups:
  - www.asp.net Security Forum
  - www.aspadvice.com Security Listserv
- Microsoft Prescriptive Guidance Books:
  - http://msdn.microsoft.com/practices/
- Watch for:
  - Improving Web Application Security – Threats and Countermeasures patterns & practices book (currently in beta)


Microsoft®