Building Secure ASP.NET Web Sites

Ben Hickman
ben@sftsrc.com
Overview

- Kinds of Threats: STRIDE
- Principles To Code By
- Balancing Security
- ASP.NET Security Architecture
- Authentication & Authorization
- SQL Server Communication
- Handling User Data
- Encryption & Privacy
- Resources
Kinds of Threats: STRIDE

- **Spoofing Identity**
- **Tampering with data (integrity)**
- **Repudiation**
- **Information disclosed**
- **Denial of service**
- **Elevation of privilege**
Principles to Code By

- Run with least privilege
- Don’t trust user input
- Defend with depth
- If you don’t use it, disable it
- Assume external systems are insecure
Balancing Security

- Tradeoffs
  - Security
  - Performance
  - Ease of use
  - Administration effort
ASP.NET Security Architecture

HTTP Requests → IIS → ASP.NET → COM+ Application → Database → Other Services
Authorization: Resource Based

- Resources are secured with ACLs
- Caller is impersonated to perform access checks

Pros
- Uses built-in security mechanisms (admin, auditing)

Cons
- Limits connection pooling
- Admin effort
- Requires a Windows identity for each user
- Kerberos required for delegation
Authorization: Role Based

- Operations are secured with roles
- Users have membership in roles

Pros
- Scalability: permits fixed identities to access services
- Caller’s security context doesn’t have to flow
- Users cannot directly access services

Cons
- Auditing
- Increased risk if servers are compromised
## Authentication: IIS

<table>
<thead>
<tr>
<th>Mode</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Broad browser support&lt;br&gt;Supports firewalls &amp; proxy servers</td>
<td>Credentials not encrypted&lt;br&gt;Log on Locally user right</td>
</tr>
<tr>
<td>Digest</td>
<td>Credential hashes are transmitted&lt;br&gt;Supports firewalls &amp; proxy servers</td>
<td>IE 5.0 or later&lt;br&gt;Log on Locally user right&lt;br&gt;Active Directory required</td>
</tr>
<tr>
<td>Integrated Windows</td>
<td>No credentials on the wire</td>
<td>Firewall &amp; proxy server problems&lt;br&gt;IE only</td>
</tr>
<tr>
<td>Certificate</td>
<td>Uses secure certificates</td>
<td>Requires client certification installation</td>
</tr>
<tr>
<td>Anonymous</td>
<td>No security</td>
<td>No security</td>
</tr>
</tbody>
</table>
Authentication: ASP.NET Identities

- IPrincipal interface
  - Determine role membership
  - Access identity
- IIdentity
  - Name, authentication info
- Page.User
Authentication: ASP.NET

- Windows
  - Uses IIS authentication settings
  - Can impersonate the caller

- Passport
  - Use Microsoft Passport authentication services
  - Microsoft Passport SDK

- None
  - No authentication or using a custom authentication module
Authentication: ASP.NET (continued)

- Forms
  - HTML login form
  - You write code to authenticate the user
  - Users are automatically redirected to your login page and back
  - Credentials are passed in the clear
  - protection attribute controls cookie validation and encryption

```xml
<authentication mode="Forms">
  <forms
    loginUrl="login.aspx"
    timeout="30"
    protection="All"
  />
</authentication>
```
Authorization: ASP.NET

- File Authorization
  - ACL checks on resources

- URL Authorization
  - Web.config settings to control access
  - * = all users
  - ? = the anonymous user
  - First allow or deny match for the user is used

```xml
<authorization>
  <allow users="Jane"/>
  <deny users="*"/>
</authorization>
```
Authorization: ASP.NET (continued)

- **.NET Roles**
  - Do not require a Windows identity
  - Page.User.IsInRole
  - <allow roles="Managers"/>

- **Principal Permission Demands**
  - Permit fine-grained authorization of classes and members
  - PrincipalPermission class
  - PrincipalPermission attribute
SQL Server Communication: Overview

- Authentication
- Identities
- Connection strings
SQL Server Communication: Authentication

- SQL Server
  - Works without Windows auth
  - Credentials are visible in:
    - Connection string
    - Over the wire

- Windows
  - Credentials are not visible
  - Uses system security services
  - Uses caller’s identity for authentication
SQL Server Communication: Identities

- **Run with least privilege**
  - Create logins for your application
  - Only grant access to necessary databases and roles

- **SQL Server Authentication**
  - Don’t use sa!
  - Embed credentials in connection strings

- **Windows Authentication**
  - ASPNET user or Serviced Component identity
  - Use when impersonating the client
SQL Server Communication: Connection Strings

- **SQL Server Authentication**
  - “User ID=Customer; Password=P@ssw0rd”
  - Consider securing connection string

- **Windows Authentication**
  - “Integrated Security=true”
  - “Trusted_Connection=true”
Handling User Data: Overview

- Why worry?
- Types of attacks
- Validation
- Error reporting
Handling User Data: Why Worry?

- **User data can be used to attack a Web site to:**
  - Reveal implementation details
  - Create malicious data, execute malicious script
  - Access restricted resources

- **URLs, cookies, HTTP headers, form data, file uploads**

- **To avoid user data attacks:**
  - Do not accept user input without validating it
  - Validate on the server
  - Define valid input range and only accept data within that range
Handling User Data: Types of Attacks

- Form data
  - Form field modification
  - SQL injection
  - Cross-site scripting
- URL format
  - Canonicalization attacks
- Cookies
- HTTP headers
Handling User Data: Validation

- **ASP.NET validation controls**
  - RequiredFieldValidator
  - CompareValidator
  - RangeValidator
  - RegularExpressionValidator
  - CustomValidator

- **Client validation is controlled by the EnableClientScript property**

```xml
<asp:TextBox id="pwd" runat="server"/>
<asp:RequiredFieldValidator
    ControlToValidate="pwd"
    ErrorMessage="Password required."
    EnableClientScript="true"
    id="pwdRequired"
    runat="server"
/>
```
Handling User Data: Validation (continued)

- SQL script injection
  - Use parameterized queries or stored procedures
  - Use ADO.NET Parameters collection

```csharp
string sql = "SELECT COUNT(EmailName) FROM Users WHERE " +
             "EmailName=@Username AND Password=@Password";

SqlCommand cmd = new SqlCommand(sql, connection);
cmd.Parameters.Add("@Username", txtUsername);
cmd.Parameters.Add("@Password", txtPassword);

connection.Open();
int count = (int)command.ExecuteScalar();
connection.Close();
```
Handling User Data: Error Reporting

- Conceal information
  - Logon credentials
  - Implementation details
  - Don’t deploy source files to production servers

- Don’t run in debug mode
  - <compilation debug="false" />

- Don’t return error details to users
  - <customErrors mode="RemoteOnly" />
Encryption & Privacy

- SSL
- IPSec

Encrypting Data
- System.Security.Cryptography

Hashing
- Store form passwords as hashes
Resources

- “Building Secure ASP.NET Applications”

- “.NET Framework Security”
  - by LaMacchia, Lange, Lyons, Martin, and Price

- “Developing Secure Web Applications”
  - Microsoft Course #2300
  - Available at OGI March 31, 2003