ASP.NET Security

Introduction

- Web applications are available to any user who can connect to the web server.
- Web applications are subject to several types of attacks.
- ASP.NET simplifies programming secure applications by providing a built-in infrastructure that supplies application-level protection against unauthorized access to Web pages.

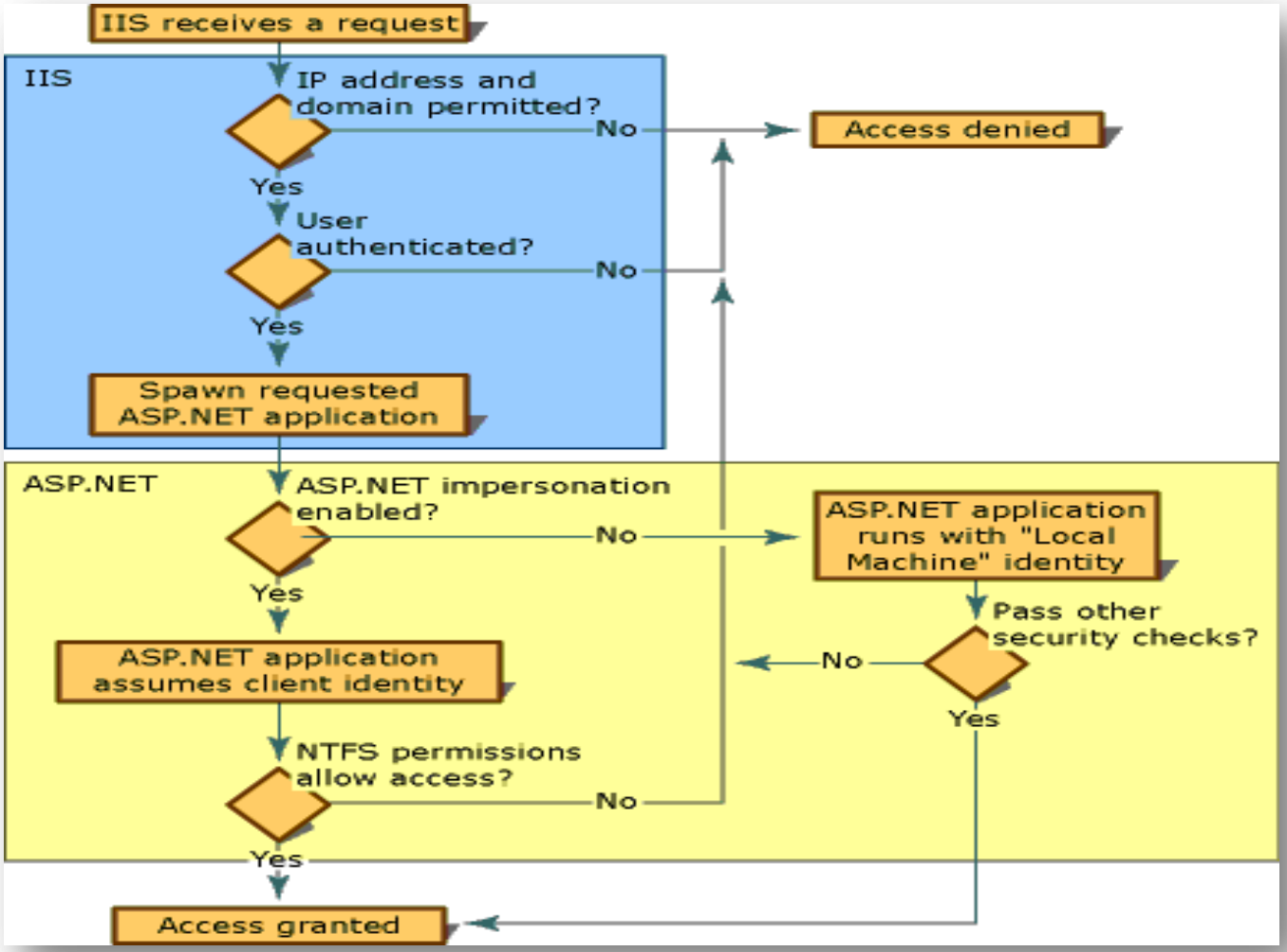
Why Security?

- Unauthorized Access to Private Data
- Packet Sniffing (Eavesdropping)
- Site Availability

Common Web Attacks

- Cross-Site Scripting
- Hidden-field Tampering
- Eavesdropping
- Session Hijacking
- Denial of Service (DOS)

ASP.NET Security Flow for Request



Areas of Security

- Authentication
- Authorization
- Impersonation
- Encryption
- Data Validation

Authentication

- To ascertain the caller's identity.
- Usually involves entering credentials into some sort of login page or window.
- The credentials are then authenticated against the Windows user accounts on the computer, a list of users in a file, or a back-end database.
- Three types: Forms, Windows & Passport.

Authorization

- The process of determining whether that user has sufficient permissions to perform a given action.
- Windows as well as code impose its own security checks.

Impersonation

- In ASP.NET, code runs with under ASPNET account.
- Impersonation allows a portion of the code to run under a different identity, with a different set of Windows permissions.

Restricted File Types

- ASP.NET automatically provides a basic level of security by blocking requests for certain file types.

.asax, .ascx, .cs, .vb, .config, .csproj, .vbproj, .resx, .resources .

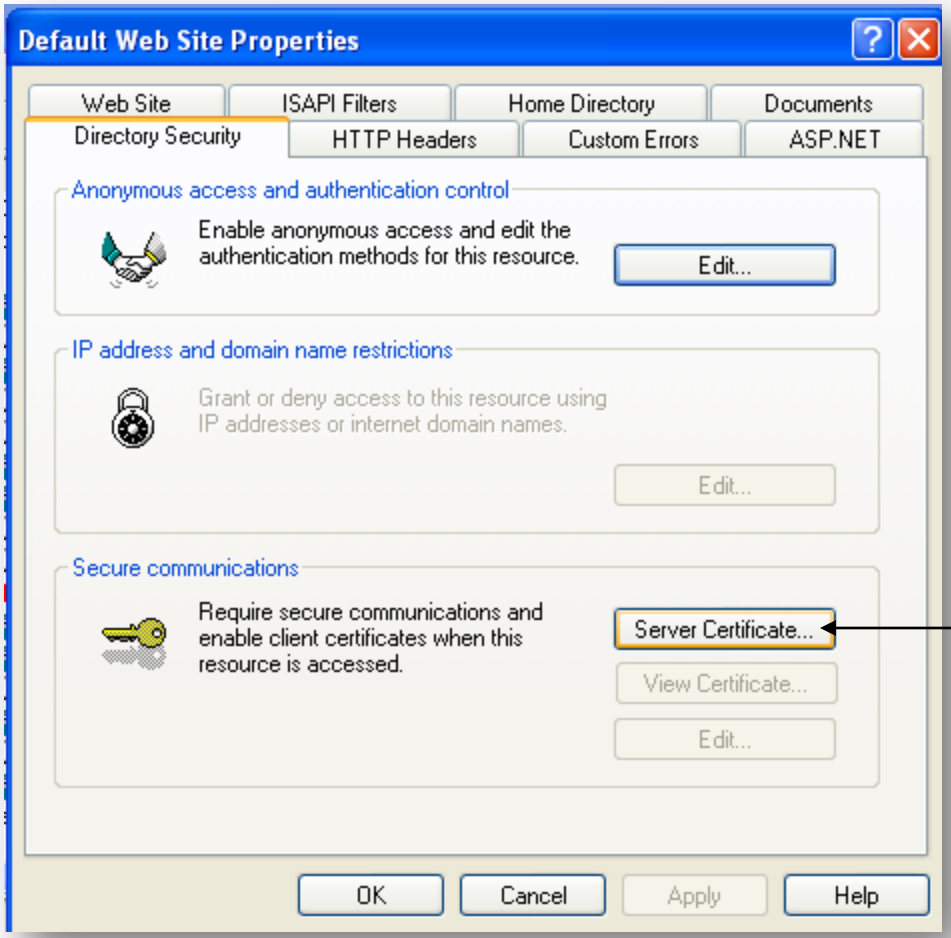
Certificates

- Certificate ensures that the site & organization information is registered & verified with the certificate authority.
- A certificate is required to use SSL which encrypts all information sent between client & server.
- One need to purchase a certificate from certificate authority.

Certificate Authorities

- Verisign (http://www.verisign.com/)
- GeoTrust (http://www.geotrust.com/)
- GlobalSign (http://www.globalsign.com/)
- E-mail a certificate request to certificate authority.
- IIS has a wizard which asks for information & generates the request file to be e-mailed.

Request File Generation



Click here to start the wizard.

Secure Sockets Layer (SSL)

- SSL encrypts communication between a client and a website.
- Certificates ensure server identity to client.
- IIS Directory settings can specify that individual folders require an SSL connection.
- To access the page over SSL, the client simply types the URL with a preceding *https* instead of *http*.
- In ASP.NET code, one can check whether connection is secure or not, by using *IsSecureConnection* of *Request* object.

Forms Authentication

- Authenticates a user by asking the user to type credentials (e.g., user name/password) into a web form.
- Put login form in application directly & configure it in web.config file.

```
<system.web>
        <authentication mode="Forms">
        <forms name="iConnect" loginUrl="Login.aspx" path="/" />
        </authentication>
…
</system.web>
```

Forms Authentication

- When a user accesses for the first^{t} time, ASP.Net redirects the user to the login page.
- If the login is successful, ASP.Net issues a ticket in the form of a cookie and redirects the user to the page originally requested.
- Uses cookie to authenticate next time.

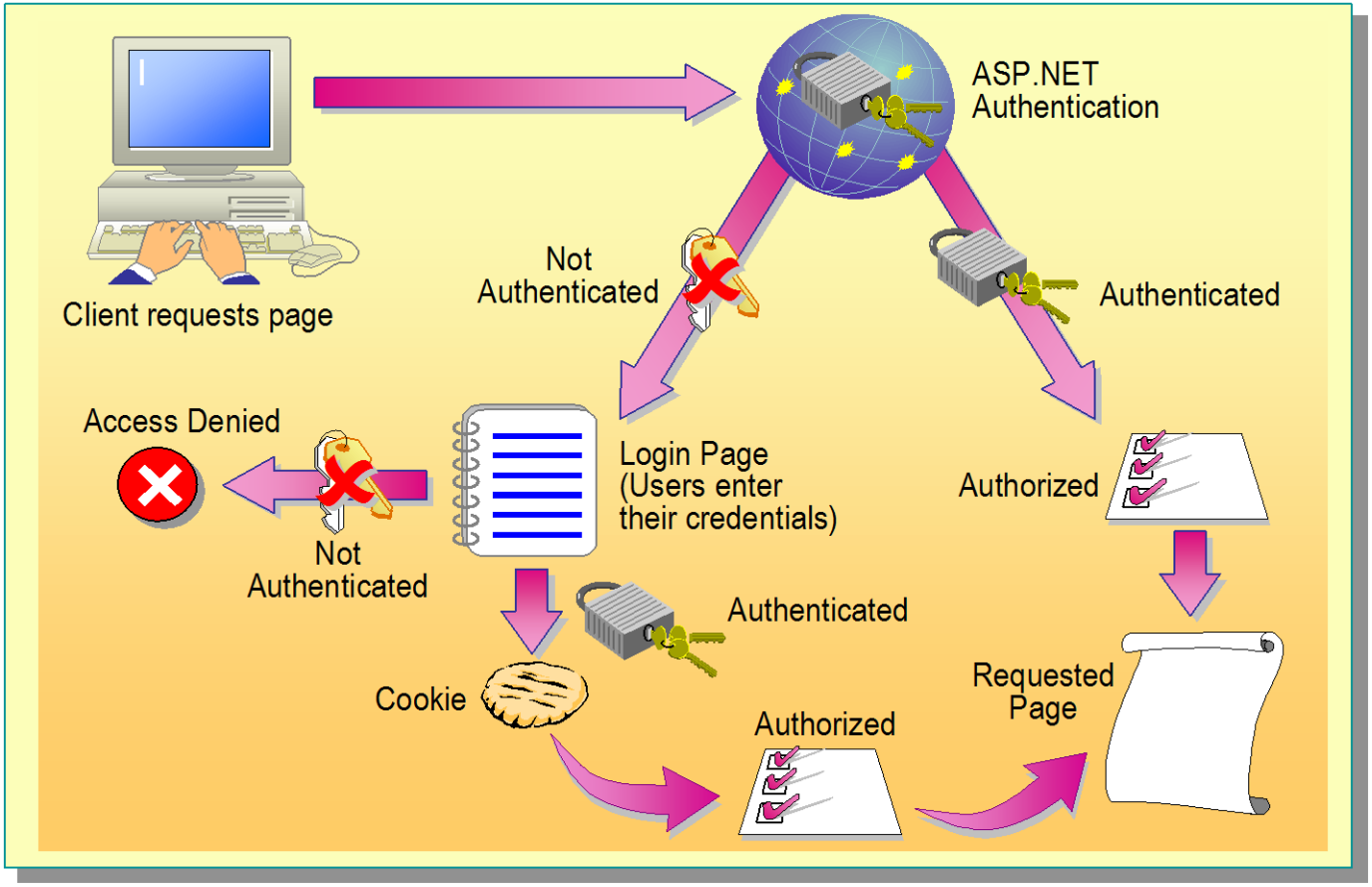
Forms Authentication Settings

- Name
  - The name of the HTTP cookie to use for authentication (defaults to .ASPXAUTH).
  - If multiple application on the web server, use unique names.
- loginUrl
  - Custom login page where user is redirected if no valid authentication cookie is found.
- Protection
  - The type of encryption and validation used for the security cookie.
  - Possible values *None, Validation, Encryption* and *All.*
  - Validation ensures the cookie isn't changed during transit

Forms Authentication Settings

- Timeout
  - The number of minutes before the cookie expires. (default is 30).
  - ASP.NET will refresh the cookie when it receives a request.
- Path
  - The path for cookies issued by the application.
  - Default is /.
- requireSSL
  - Specifies whether a Secure Sockets Layer (SSL) connection is required when transmitting authentication information.

Forms Authentication Flow



Authorization

- Control web page access.
- <authorization> section of web.config has access control rules.

```
<system.web>
        <authentication mode="Forms">
        ...
        </authentication>
        <authorization>
                <deny users="*" />
        </authorization>
...
</system.web>
```

* for all users (autheticated & non-autheticated)

<allow> & <deny> Element Attributes

- Users
  - Specifies users by their names in a comma separated list.
  - “?” matches all anonymous users.
- Roles
  - Specifies access groups that are allowed or denied access. Comma-separated list of roles.
- Verbs
  - Specifies the HTTP transmission method (e.g GET) that is allowed or denied access.

Authorization

```
<system.web>
        <authentication mode=”Forms”>
        <forms name=”iConnect” loginUrl=”Login.aspx” path=”/” />
        </authentication>
        <authorization>
                <allow users=“ram,paddy” />
                <deny users=”?” />
        </authorization>
</system.web>
```

Controlling Access to Specific Directories

- A common application design is to place files that require authentication in a separate directory.
- Add a web.config file that specifies stricter settings in the secured directory.
- Subdirectory config file override the parent directory config settings.

Controlling Access to Specific Page

- Use <location> tag in web.config.
- It is directly nested in <configuration>
- One can have multiple <location> tags.

```
<location path="~/SecuredPage.aspx" >
        <system.web>
                <authorization>
                        <deny users="?" />
                </authorization>
        </system.web>
</location>
```

Forms Authentication Configuration

- Enable anonymous access in IIS
- Configure <authentication> section
  - Set mode to “Forms”
  - Add the <forms> section
- Configure <authorization> section
  - Deny access to anonymous user
- Create logon page
  - Validate the user
  - Provide authentication cookie
  - Redirect the user to the requested page

Login Page

- ASP.NET provides a special *FormsAuthentication* class, which provides static methods that help manage the authentication.

```
protected void Button1_Click(object sender, EventArgs e)
{
if (TextBox1.Text == “iconnect” && TextBox2.Text == “iconnect”) {
FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true);
}
else {
Response.Write(“Invalid credentials”);
}
}
```

FormsAuthentication Members

- Authenticate()
  - Checks a user name and password against a list of accounts that can be entered in the web.config file.
  - Does not require manual test for user & password.
- RedirectFromLoginPage()
  - Logs the user into an ASP.NET application by creating the cookie & redirecting the user to the requested secure page.
  - Second parameter if true, creates persistent cookie.

FormsAuthentication Members

- SignOut()
  - Logs the user out of the ASP.NET application by removing the current cookie.
- SetAuthCookie()
  - Logs the user into an ASP.NET application by creating and attaching the forms authentication cookie. Do not redirect.
- HashPasswordForStoringInConfigFile()
  - Encrypts a string of text using the specified algorithm (SHA1 or MD5).

<credentials> Element in web.config

- Hard-coding user name & password is not a good idea.
- One can store user names & password in web.config in credentials section. Use *Autheticate* method.
- Limitation: File Size. Use database.

```
<forms name="iConnect" loginUrl="Login.aspx" path="/">
        <credentials passwordFormat="Clear">
        <user name="iconnect" password="iconnect" />
</credentials>
</forms>
```

Clear: stores password in clear text.

MD5 : Hashes password using MD5 hash algo.

SHA1: Hashes password using SHA1 hash algo

Preventing Page Review after Logout

- Local caching by browsers and proxy servers may allow a user to review information even after they have logged out.

- HTTP Headers to avoid this scenario is :
  - **Expires** : the expire header is set to an absolute date & time
  - **Pragma No-Cache** : Signals proxy servers to expire responses
  - **Cache-Control** : Values for it are
    - A) Public – Content stored in public shared caches
    - B) Private - Content stored in private shared caches
    - C) No-Cache – Content not cached
    - D) No-Store - Content may be cached for the length of the session, but not archived.

Prevening Page Review

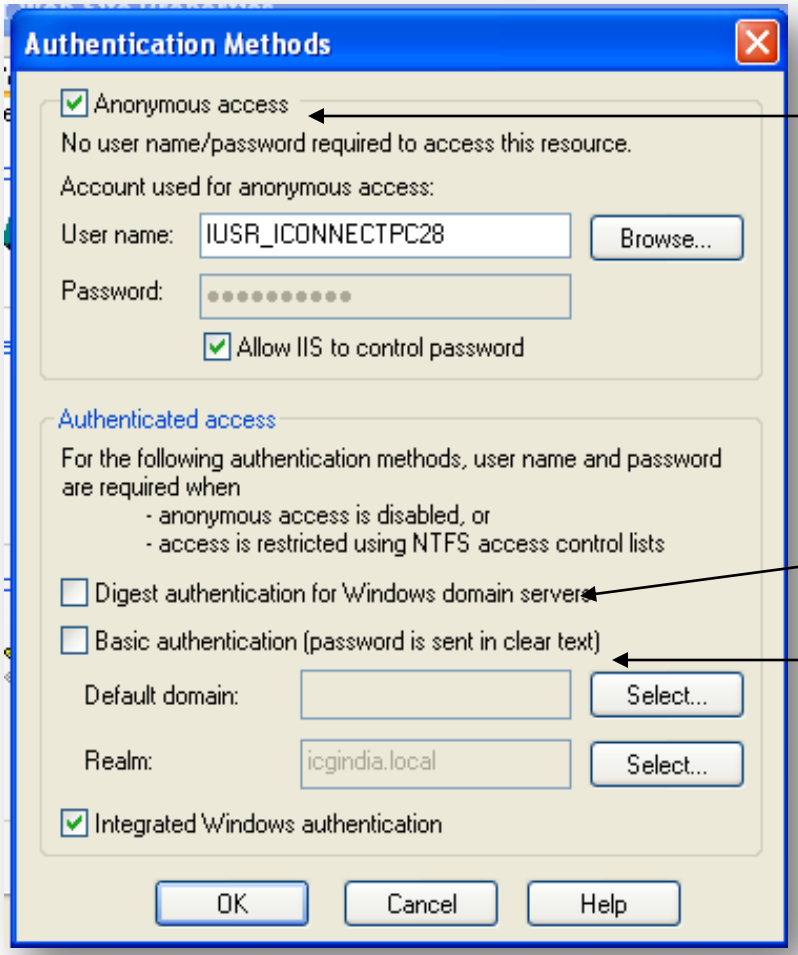
File Edit Format View Help

```
protected void Page_Load(object sender, EventArgs e)
    {
        Response.Cache.SetExpires(DateTime.Now);
        Response.Cache.SetCacheability(HttpCacheability.NoCache);

                lbl.Text = "U r Login ";
    }
    protected void Button1_Click(object sender, EventArgs e)
    {

        FormsAuthentication.SignOut();
        FormsAuthentication.RedirectToLoginPage("Login.aspx");
    }
```

Windows Authentication

- User is authenticated by IIS
- Easiest of all
- Can be used with in combination of Basic, Digest authentication.
- Request flow
  - Client makes request
  - IIS authenticates request, forwards security token to ASP.NET.
  - ASP.NET limits access to resources using security token.

Windows Authentication Methods



Anonymous Access

Digest Authetication

Basic Authentication

Windows Authentication

- Basic authentication: transmits a user name and password in each request; IIS maps them to an account on the web server and generates a token.
  - Suppose a web page is placed in the virtual directory
  - Suppose IIS is configured to disallow anonymous access to that directory and to require basic authentication
  - When a user attempts to access it for the first time (via HTTP request, a status code 401 is returned indicating that it requires basic authentication
  - The user's browser then prompts the user for windows user name/password
  - Problem: User name/password sent in plain text between the browser and the web server with each request; user needs a windows account

Windows Authentication

- Digest authentication: User name/password are sent as an encrypted token with each request
  - Supported by only IE 5.0 or later.
- Windows Integrated Authetication : When using Integrated authentication, Internet Explorer can send the required information automatically using the currently logged-in Windows account on the client, provided it's on a trusted domain.
  - Integrated authentication is supported only on Internet Explorer 2.0 and later and won't work across proxy servers.

Windows Authentication Configuration

- Set mode to “Windows”
- Configure <authorization> section

```
<authentication mode=" Windows" />
<authorization>
  <deny users="?" />
  <allow users= "*" />
</authorization>
```

Passport Authentication

- Single sign-in across member sites
- Includes user profiles services
- Integrated into ASP.NET authentication
- Scenarios
  - Don’t want to maintain a database of users
  - Provide personalized content
  - Need to provide single-sign in capabilities
- Visit http://www.passport.com/

Passport Authentication Configuration

- What you need:
  - Install Passport SDK
  - Register with Microsoft Passport
- Set mode to “Passport”
- Configure <passport> section

```
<authentication mode="Passport">
        <passport redirectUrl="internal|url" />
</authentication>
```

Impersonation

- Code impersonate identity of the user. Code access to resources beyond user privileges not allowed.

  e.g. If user not allowed access to file, code cannot access the file.

- Used in Windows network.

- For Impersonation to work, the authenticated user account must correspond to a Windows account.

Impersonation Configuration in web.config

- <identity impersonate = “false” />
- <identity impersonate = “true” />
- <identity impersonate = “true” userName = “username” password = “password” />

Programmatic Authorization

- *Page* object has *User* property which provides an instance of *IPrincipal*.

- *User* has one property & one method:

  - *Identity*: property provides an instance of the *System.Security.Principal.IIdentity* object.

  - *IsInRole()*:Takes a single parameter, a string representation of the system role. Returns true if the user is member of the role specified.

User.Identity

- Name
  - Provides the username of the user.
- IsAuthenticated()
  - Returns a Boolean value specifying whether the user has been authenticated.
- AuthenticationType
  - Provides the authentication type of the current user. e.g. Basic, Forms, Passport etc.

Windows Users (Check Roles)

```
if(User.IsInRole("BUILTIN\Administrators"))
    Response.Write("You are an Admin");
else if(User.IsInRole("BUILTIN\Users"))
    Response.Write("You are a User");
else
    Response.Write("Invalid user");
```

WindowsIdentity Class

- Namespace: *Syatem.Security.Principal.*
- *WindowsIdentity* much richer than *Identity*.

```
protected void Page_Load(object sender, EventArgs e)
{
WindowsIdentity AuthUser = WindowsIdentity.GetCurrent();
Response.Write(AuthUser.AuthenticationType.ToString() + “<br>” +
AuthUser.ImpersonationLevel.ToString() + “<br>” +
AuthUser.IsAnonymous.ToString() + “<br>” +
AuthUser.IsAuthenticated.ToString() + “<br>” +
AuthUser.IsGuest.ToString() + “<br>” +
AuthUser.IsSystem.ToString() + “<br>” +
AuthUser.Name.ToString());
}
```

Programmatic Impersonation

```
if(User.GetType()==typeof(WindowsPrincipal))
{
   WindowsIdentity id=(WindowsIdentity)User.Identity;
   WindowsImpersonationContext impersonateContext=id.Impersonate();
   // now perform tasks under the impersonated ID.
   …
   // Revert to the original ID
   impersonateContext.Undo();
}
```