Erstellen sicherer ASP.NET-Anwendungen

Authentifizierung, Autorisierung und sichere Kommunikation

Auf der Orientierungsseite finden Sie einen Ausgangspunkt und eine vollständige Übersicht zum *Erstellen sicherer ASP.NET-Anwendungen*.

Zusammenfassung

Nachstehend wird erläutert, wie ein benutzerdefiniertes Principalobjekt erstellt wird, das eine erweiterte rollenbasierte Funktionalität bereitstellt, die für die .NET-Autorisierung verwendet werden kann.

Vorgehensweise: Implementieren von IPrincipal

Das .NET Framework stellt die Klassen **WindowsPrincipal** und **GenericPrincipal** bereit, mit denen grundlegende Rollenüberprüfungsfunktionen für Windows- bzw. Nicht-Windows-Authentifizierungsmechanismen zur Verfügung stehen. Mit beiden Klassen wird die **IPrincipal**-Schnittstelle implementiert. Damit diese Objekte für die Autorisierung verwendet werden können, setzt ASP.NET voraus, dass die Objekte in **HttpContext.User** gespeichert sind. Bei Windows-basierten Anwendungen müssen sie in **Thread.CurrentPrincipal** gespeichert sein.

Die von diesen Klassen gebotenen Funktionen sind für die meisten Anwendungsszenarien ausreichend. Anwendungen können die **IPrincipal.IsInRole**-Methode explizit aufrufen, um programmatische Rollenüberprüfungen durchzuführen. Wird die **Demand**-Methode der **PrinicpalPermission**-Klasse für die Forderung verwendet, dass ein Aufrufer einer bestimmten Rolle angehören muss (entweder deklarativ oder obligatorisch), ergibt sich hieraus auch ein Aufruf von **IPrincipal.IsInRole**.

Unter bestimmten Umständen müssen Sie möglicherweise eigene Principalimplementierungen entwickeln, indem Sie eine Klasse erstellen, die die **IPrincipal**-Schnittstelle implementiert. Für die .NET-Autorisierung kann jede Klasse verwendet werden, die **IPrincipal** implementiert.

Zu den Gründen für die Implementierung einer eigenen **IPrincipal**-Klasse gehören die folgenden:

- Sie benötigen eine erweiterte Rollenüberprüfungsfunktionalität. Möglicherweise möchten Sie mit Methoden arbeiten, mit denen Sie prüfen können, ob ein bestimmter Benutzer mehreren Rollen angehört. Beispiel:

```
CustomPrincipal.IsInAllRoles( "Role1", "Role2", "Role3" )
CustomPrincipal.IsInAnyRole( "Role1", "Role2", "Role3" )
```

- Sie möchten eine zusätzliche Methode oder Eigenschaft implementieren, die eine Liste der Rollen in Form eines Arrays zurückgibt. Beispiel:

```
string[] roles = CustomPrincipal.Roles;
```

- Sie möchten, dass Ihre Anwendung eine Rollenhierarchielogik durchsetzt. Beispielsweise könnte ein "Senior Manager" als in der Hierarchie höher stehend als ein "Manager" betrachtet werden. Dies könnte mit Methoden wie der folgenden getestet werden:

```
CustomPrincipal.IsInHigherRole("Manager");
CustomPrincipal.IsInLowerRole("Manager");
```

- Sie möchten eine verzögerte Initialisierung der Rollenlisten implementieren. Beispielsweise könnten Sie die Rollenliste nur dann dynamisch laden, wenn eine Rollenüberprüfung angefordert wird.

In dieser Vorgehensweise wird erläutert, wie eine benutzerdefinierte **IPrincipal**-Klasse implementiert und für die rollenbasierte Autorisierung in einer ASP.NET-Anwendung verwendet wird, die mit der Formularauthentifizierung arbeitet.

Anforderungen

Im Folgenden finden Sie eine Liste der empfohlenen Hardware und Software und eine Beschreibung der Netzwerkinfrastruktur, der Fähigkeiten und Kenntnisse sowie der Service Packs, die Sie benötigen:

- Microsoft® Visual Studio® .NET als Entwicklungssystem

Die in dieser Vorgehensweise erläuterten Verfahren setzen zudem Kenntnisse der ASP.NET-Webentwicklung mit dem Entwicklungstool Microsoft Visual C#™ voraus.

Zusammenfassung

Diese Vorgehensweise enthält folgende Verfahren:

1. Erstellen einer einfachen Webanwendung
2. Konfigurieren der Webanwendung für die Formularauthentifizierung
3. Erzeugen eines Authentifizierungstickets für authentifizierte Benutzer
4. Erstellen einer Klasse, die **IPrincipal** implementiert und erweitert
5. Erstellen des **CustomPrincipal**-Objekts
6. Testen der Anwendung

1. Erstellen einer einfachen Webanwendung

Mit diesem Verfahren wird eine neue ASP.NET-Webanwendung erstellt. Die Anwendung enthält zwei Seiten, eine Standardseite, auf die nur authentifizierte Benutzer zugreifen können, und eine Anmeldeseite für die Eingabe der Anmeldeinformationen.

▶ **So erstellen Sie eine einfache Webanwendung**

1. Starten Sie Visual Studio .NET, und erstellen Sie mit C# eine neue ASP.NET-Webanwendung mit Namen **CustomPrincipalApp**.
2. Benennen Sie **WebForm1.aspx** in **Logon.aspx** um.
3. Fügen Sie **Logon.aspx** die in Tabelle 1 aufgeführten Steuerelemente hinzu, um ein Anmeldeformular zu erstellen.

Tabelle 1: *Die Steuerelemente für **Logon.aspx***

Typ des Steuerelements	Text	ID
Bezeichnungsfeld	User Name:	-
Bezeichnungsfeld	Password	-
Textfeld	-	txtUserName
Textfeld	-	txtPassword
Schaltfläche	Logon	btnLogon

4. Legen Sie die **TextMode**-Eigenschaft des Textfeldsteuerelements **txtPassword** auf **Password** fest.
5. Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf **CustomPrincipalApp**, zeigen Sie auf **Hinzufügen**, und klicken Sie dann auf **Web Form hinzufügen**.
6. Geben Sie **default.aspx** als neuen Namen des Formulars ein, und klicken Sie dann auf **Öffnen**.

2. Konfigurieren der Webanwendung für die Formular-authentifizierung

▶ **So bearbeiten Sie die Datei "Web.config" der Anwendung, um die Anwendung für die Formularauthentifizierung zu konfigurieren**

1. Öffnen Sie **Web.config** im Projektmappen-Explorer.
2. Suchen Sie das <**authentication**>-Element, und ändern Sie das **mode**-Attribut in **Forms**.
3. Fügen Sie das <**forms**>-Element als untergeordnetes Element des <**authentication**>-Elements hinzu, und legen Sie die Attribute **loginUrl**, **name**, **timeout** und **path** wie nachstehend gezeigt fest:

```
<authentication mode="Forms">
  <forms loginUrl="logon.aspx" name="AuthCookie" timeout="60" path="/">
  </forms>
</authentication>
```

4. Fügen Sie das folgende <**authorization**>-Element unterhalb des <**authentication**>-Elements hinzu. Dies sorgt dafür, dass nur authentifizierte Benutzer auf die Anwendung zugreifen können. Das im Vorfeld eingerichtete **loginUrl**-Attribut des <**authentication**>-Elements leitet nicht authentifizierte Anforderungen auf die Seite **Logon.aspx** um.

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

3. Erzeugen eines Authentifizierungstickets für authentifizierte Benutzer

In diesem Verfahren wird der Code zum Erzeugen des Authentifizierungstickets für authentifizierte Benutzer geschrieben. Das Authentifizierungsticket ist eine Art Cookie, das vom **FormsAuthenticationModule** von ASP.NET verwendet wird.

Der Authentifizierungscode umfasst in der Regel das Nachschlagen des übergebenen Benutzernamens und des Kennworts, und zwar entweder in einer benutzerdefinierten Datenbank oder im Verzeichnisdienst Microsoft Active Directory®.

Weitere Informationen darüber, wie dieses Nachschlagen erfolgt, finden Sie in den folgenden Vorgehensweisen im Abschnitt "Referenz" dieses Handbuchs:

- "Vorgehensweise: Verwenden der Formularauthentifizierung mit Active Directory"
- "Vorgehensweise: Verwenden der Formularauthentifizierung mit SQL Server 2000"

▶ So erzeugen Sie ein Authentifizierungsticket für authentifizierte Benutzer

1. Öffnen Sie die Datei **Logon.aspx.cs**, und fügen Sie am Anfang der Datei unterhalb der vorhandenen **using**-Anweisungen die folgenden **using**-Anweisung hinzu.

```
using System.Web.Security;
```

2. Fügen Sie der **WebForm1**-Klasse die folgende private Hilfsmethode mit Namen **IsAuthenticated** hinzu, die verwendet wird, um die Benutzernamen und Kennwörter für die Authentifizierung der Benutzer zu überprüfen. In diesem Code wird davon ausgegangen, dass sämtliche Kombinationen aus Benutzername und Kennwort gültig sind.

```
private bool IsAuthenticated( string username, string password )
{
  // Lookup code omitted for clarity
  // This code would typically validate the user name and password
  // combination against a SQL database or Active Directory
  // Simulate an authenticated user
  return true;
}
```

3. Fügen Sie die nachstehende private Hilfsmethode mit Namen **GetRoles** hinzu, die verwendet wird, um die Rollen abzurufen, denen der Benutzer angehört.

```
private string GetRoles( string username, string password )
{
  // Lookup code omitted for clarity
  // This code would typically look up the role list from a database table.
  // If the user was being authenticated against Active Directory, the
  // Security groups and/or distribution lists that the user belongs to may be
  // used instead

  // This GetRoles method returns a pipe delimited string containing roles
  // rather than returning an array, because the string format is convenient
  // for storing in the authentication ticket / cookie, as user data
  return "Senior Manager|Manager|Employee";
}
```

4. Zeigen Sie das Formular **Logon.aspx** im Entwurfsmodus an, und doppelklicken Sie dann auf die Schaltfläche **Logon**, um einen Ereignishandler für das Klickereignis zu erstellen.
5. Fügen Sie einen Aufruf der **IsAuthenticated**-Methode hinzu, und übergeben Sie den Benutzernamen und das Kennwort, der bzw. das im Anmeldeformular eingegeben wurde. Weisen Sie den Rückgabewert einer Variablen vom Typ **bool** zu, die angibt, ob der Benutzer authentifiziert werden konnte oder nicht.

```
bool isAuthenticated = IsAuthenticated( txtUserName.Text,
                                        txtPassword.Text );
```

6. Fügen Sie einen Aufruf der **GetRoles**-Methode hinzu, um für den Fall, dass der Benutzer authentifiziert wird, die Rollenliste des Benutzers abzurufen.

```
if (isAuthenticated == true )
{
  string roles = GetRoles( txtUserName.Text, txtPassword.Text );
```

7. Erstellen Sie ein neues Formularauthentifizierungsticket, das den Benutzernamen, eine Ablaufzeit und die Liste der Rollen enthält, denen der Benutzer angehört. Beachten Sie, dass die **UserData**-Eigenschaft des Authentifizierungstickets verwendet wird, um die Rollenliste des Benutzers zu speichern. Beachten Sie zudem, dass mit dem folgenden Code ein nicht persistentes Ticket erstellt wird, obwohl es vom Anwendungsszenario abhängig ist, ob das Ticket/Cookie persistent ist oder nicht.

```
  // Create the authentication ticket
  FormsAuthenticationTicket authTicket = new
       FormsAuthenticationTicket(1,                          // version
                                 txtUserName.Text,           // user name
                                 DateTime.Now,               // creation
                                 DateTime.Now.AddMinutes(60),// Expiration
                                 false,                      // Persistent
                                 roles );                    // User data
```

8. Fügen Sie Code zum Erstellen einer verschlüsselten Zeichenfolgendarstellung des Tickets hinzu, und speichern Sie diese als Datenwert in einem **HttpCookie**-Objekt.

```
  // Now encrypt the ticket.
  string encryptedTicket = FormsAuthentication.Encrypt(authTicket);
  // Create a cookie and add the encrypted ticket to the
  // cookie as data.
  HttpCookie authCookie =
               new HttpCookie(FormsAuthentication.FormsCookieName,
                              encryptedTicket);
```

9. Fügen Sie das Cookie der Cookieauflistung hinzu, die an den Browser des Benutzers zurückgegeben wird.

```
  // Add the cookie to the outgoing cookies collection.
  Response.Cookies.Add(authCookie);
```

10. Leiten Sie den Benutzer auf die ursprünglich angeforderte Seite um.

```
  // Redirect the user to the originally requested page
  Response.Redirect( FormsAuthentication.GetRedirectUrl(
                                              txtUserName.Text,
                                              false ));
}
```

4. Erstellen einer Klasse, die IPrincipal implementiert und erweitert

Mit diesem Verfahren wird eine Klasse erstellt, die die **IPrincipal**-Schnittstelle implementiert. Darüber hinaus werden der Klasse weitere Methoden und Eigenschaften hinzugefügt, um eine zusätzliche rollenbasierte Autorisierungsfunktionalität bereitzustellen.

▶ **So erstellen Sie eine Klasse, die IPrincipal implementiert und erweitert**

1. Fügen Sie dem aktuellen Projekt eine neue Klasse mit Namen **CustomPrincipal** hinzu.
2. Fügen Sie am Anfang von **CustomPrincipal.cs** die folgende **using**-Anweisung hinzu.

```
using System.Security.Principal;
```

3. Leiten Sie die **CustomPrincipal**-Klasse von der **IPrincipal**-Schnittstelle ab.

```
public class CustomPrincipal : IPrincipal
```

4. Fügen Sie der Klasse die folgenden Membervariablen vom Typ **private** hinzu, um das **IIdentity**-Objekt zu verwalten, das mit dem aktuellen Principal und der Rollenliste des Principals verbunden ist.

```
private IIdentity _identity;
private string [] _roles;
```

5. Fügen Sie dem Standardkonstruktor der Klasse ein **IIdentity**-Objekt und ein Array mit Rollen hinzu. Verwenden Sie die übergebenen Werte, um die privaten Membervariablen wie nachstehend gezeigt zu initialisieren.

```
public CustomPrincipal(IIdentity identity, string [] roles)
{
  _identity = identity;
  _roles = new string[roles.Length];
  roles.CopyTo(_roles, 0);
  Array.Sort(_roles);
}
```

6. Implementieren Sie die **IsInRole**-Methode und die **Identity**-Eigenschaft wie von der **IPrincipal**-Schnittstelle definiert (siehe unten).

```
// IPrincipal Implementation
public bool IsInRole(string role)
{
  return Array.BinarySearch( _roles, role ) > 0 ? true : false;
}
public IIdentity Identity
{
  get
  {
    return _identity;
  }
}
```

7. Fügen Sie die beiden folgenden Methoden vom Typ **public** hinzu, mit denen eine erweiterte rollenbasierte Prüfungsfunktion bereitgestellt wird.

```
// Checks whether a principal is in all of the specified set of roles
public bool IsInAllRoles( params string [] roles )
{
  foreach (string searchrole in roles )
  {
    if (Array.BinarySearch(_roles, searchrole) < 0 )
      return false;
  }
  return true;
}
// Checks whether a principal is in any of the specified set of roles
public bool IsInAnyRoles( params string [] roles )
{
  foreach (string searchrole in roles )
  {
    if (Array.BinarySearch(_roles, searchrole ) > 0 )
      return true;
  }
  return false;
}
```

5. Erstellen des CustomPrincipal-Objekts

Mit diesem Verfahren wird ein Ereignishandler für die Anwendungsauthentifizierung implementiert, und basierend auf den Informationen im Authentifizierungsticket wird das **CustomPrincipal**-Objekt erstellt, das den authentifizierten Benutzer repräsentiert.

▶ **So erstellen Sie das CustomPrincipal-Objekt**

1. Öffnen Sie **gobal.asax** im Projektmappen-Explorer.
2. Wechseln Sie in die Codeansicht, und fügen Sie am Anfang der Datei die folgenden **using**-Anweisungen hinzu.

```
using System.Web.Security;
using System.Security.Principal;
```

3. Suchen Sie den Ereignishandler **Application_AuthenticateRequest**, und fügen Sie den folgenden Code hinzu, um aus der mit der Anforderung übergebenen Cookieauflistung das Formularauthentifizierungscookie abzurufen.

```
// Extract the forms authentication cookie
string cookieName = FormsAuthentication.FormsCookieName;
HttpCookie authCookie = Context.Request.Cookies[cookieName];

if(null == authCookie)
{
  // There is no authentication cookie.
  return;
}
```

4. Fügen Sie den folgenden Code hinzu, um das Authentifizierungsticket aus dem Formularauthentifizierungscookie zu extrahieren und zu entschlüsseln.

```
FormsAuthenticationTicket authTicket = null;
try
{
  authTicket = FormsAuthentication.Decrypt(authCookie.Value);
}
catch(Exception ex)
{
  // Log exception details (omitted for simplicity)
  return;
}

if (null == authTicket)
{
  // Cookie failed to decrypt.
  return;
}
```

5. Fügen Sie den folgenden Code hinzu, um die per Pipe getrennte Liste der Rollennamen auszulesen, die bei der ursprünglichen Authentifizierung des Benutzers an das Ticket angehängt wurde.

```
// When the ticket was created, the UserData property was assigned a
// pipe delimited string of role names.
string[] roles = authTicket.UserData.Split('|');
```

6. Fügen Sie den folgenden Code hinzu, um das **FormsIdentity**-Objekt mit dem Benutzernamen zu erstellen, der dem Ticketnamen entnommen wurde, und um ein **CustomPrincipal**-Objekt zu erstellen, das diese Identität zusammen mit der Rollenliste des Benutzers enthält.

```
// Create an Identity object
FormsIdentity id = new FormsIdentity( authTicket );

// This principal will flow throughout the request.
CustomPrincipal principal = new CustomPrincipal(id, roles);
// Attach the new principal object to the current HttpContext object
Context.User = principal;
```

5. Testen der Anwendung

Mit diesem Verfahren wird der Seite **default.aspx** Code hinzugefügt, um Informationen aus dem **CustomPrincipal**-Objekt anzuzeigen, das an das aktuelle **HttpContext**-Objekt angefügt wurde. So wird sichergestellt, dass das Objekt ordnungsgemäß erstellt und der aktuellen Webanforderung zugewiesen wurde. Hiermit wird zudem die rollenbasierte Funktionalität geprüft, die von der neuen Klasse unterstützt wird.

▶ **So testen Sie die Anwendung**

1. Doppelklicken Sie im Projektmappen-Explorer auf **default.aspx**.
2. Doppelklicken Sie auf das Webformular **default.aspx**, um den Ereignishandler zum Laden der Seite anzuzeigen.
3. Führen Sie einen Bildlauf zum Anfang der Datei durch, und fügen Sie unterhalb der vorhandenen **using**-Anweisungen die folgende **using**-Anweisung hinzu.

```
using System.Security.Principal;
```

4. Kehren Sie zum Ereignishandler für das Laden der Seite zurück, und fügen Sie den folgenden Code hinzu, um den Identitätsnamen anzuzeigen, der dem **CustomPrincipal**-Objekt zugewiesen wurde, das mit der aktuellen Webanforderung verbunden ist.

```
CustomPrincipal cp = HttpContext.Current.User as CustomPrincipal;
Response.Write( "Authenticated Identity is: " +
                cp.Identity.Name );
Response.Write( "<p>" );
```

5. Fügen Sie den folgenden Code hinzu, um die Rollenmitgliedschaft der aktuell authentifizierten Identität zu prüfen, und zwar unter Verwendung der standardmäßigen **IsInRole**-Methode und der zusätzlich von der **CustomPrincipal**-Klasse unterstützten Methoden **IsInAnyRole** und **IsInAllRoles**.

```
if ( cp.IsInRole("Senior Manager") )
{
  Response.Write( cp.Identity.Name + " is in the " + "Senior Manager Role" );
  Response.Write( "<p>" );
}

  if ( cp.IsInAnyRoles("Senior Manager", "Manager", "Employee", "Sales") )
  {
    Response.Write( cp.Identity.Name + " is in one of the specified roles");
    Response.Write( "<p>" );
  }
  if ( cp.IsInAllRoles("Senior Manager", "Manager", "Employee", "Sales") )
  {
    Response.Write( cp.Identity.Name + " is in ALL of the specified roles" );
    Response.Write( "<p>" );
  }
  else
  {
    Response.Write( cp.Identity.Name +
                    " is not in ALL of the specified roles" );
    Response.Write("<p>");
  }

  if ( cp.IsInRole("Sales") )
    Response.Write( "User is in Sales role<p>" );
  else
    Response.Write( "User is not in Sales role<p>" );
```

6. Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf **default.aspx**, und klicken Sie dann auf **Als Startseite festlegen**.
7. Klicken Sie im Menü **Erstellen** auf **Projektmappe erstellen**.
8. Drücken Sie **STRG+F5**, um die Anwendung auszuführen. Da **default.aspx** als Startseite konfiguriert ist, ist dies auch die zuerst angeforderte Seite.
9. Wenn Sie auf die Anmeldeseite umgeleitet werden (da Sie anfänglich ja nicht über ein Authentifizierungsticket verfügen), geben Sie irgendeinen Benutzernamen und ein beliebiges Kennwort ein, und klicken Sie dann auf **Logon**.
10. Vergewissern Sie sich, dass Sie auf die Seite **default.aspx** umgeleitet werden und dass die Benutzeridentität und die korrekten Rollendetails angezeigt werden. Der Benutzer ist ein Mitglied der Rollen **Senior Manager**, **Manager** und **Employee**, jedoch kein Mitglied der Rolle **Sales**.

Weitere Ressourcen

Weitere Informationen über die formularbasierte Authentifizierung finden Sie in den folgenden Vorgehensweisen im Abschnitt "Referenz" dieses Handbuchs:

- "Vorgehensweise: Verwenden der Formularauthentifizierung mit GenericPrincipal-Objekten"
- "Vorgehensweise: Verwenden der Formularauthentifizierung mit Active Directory"
- "Vorgehensweise: Verwenden der Formularauthentifizierung mit SQL Server 2000"