4) Безопасность и авторизация в ASP.NET // Конфигурация приложений ASP.NET

Безопасность и авторизация в ASP.NET

Рассмотрим основные понятия, связанные с процессом обеспечения безопасности. *Аутентификацией* (*authentication*) называется процесс идентификации пользователей приложений. *Авторизация* (*authorization*) – это процесс предоставления доступа пользователям на основе их идентификационных данных. Аутентификация наряду с авторизацией представляют собой средства защиты веб-приложений от несанкционированного доступа.

ASP.NET совместно с веб-сервером обеспечивает несколько возможных типов аутентификации: `Windows` (по умолчанию), `Forms`, `Passport` и `None`. Выбор типа определяет механизм хранения маркеров аутентифицированного пользователя. В случае типа `Windows` маркер помещается в контекст потока рабочего процесса ASP.NET. Если используется тип `Forms`, маркер передается от клиента к серверу и обратно в cookie-файле.

Задать требования аутентификации клиентов веб-приложения можно путем добавления соответствующих элементов в конфигурационный файл приложения `web.config`. Тип аутентификации клиентов задается с помощью элемента `<authentication>`, атрибут `mode` которого может принимать одно из четырех значений: `Windows`, `Forms`, `Passport` и `None`. Аутентификация выполняется для ограничения доступа клиентов ко всему приложению или к его частям с помощью элемента `<authorization>`. В этот элемент добавляются подэлементы `<allow>` и `<deny>`, задающие предоставление или отказ в доступе отдельным пользователям и ролям. Метасимвол * используется для представления всех пользователей, а ? – для представления анонимных пользователей. Следующий пример конфигурационного файла отказывает анонимным пользователям в праве доступа к сайту:

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

Элементы `<allow>` и `<deny>` поддерживают три атрибута: `users`, `roles` и `verbs`. Значениями этих атрибутов могут быть разделенные запятыми списки пользователей, ролей и команд. При определении прав доступа пользователя к ресурсу ASP.NET придерживается следующего алгоритма:

- Строится список всех <allow> и <deny> элементов, начиная с ближайшего к ресурсу web.config, а затем добавляя элементы из web.config более высокого уровня, и, наконец, элементы machine.config.
- Список просматривается сверху вниз до первого совпадения имени или роли в списке с именем или ролью текущего пользователя.
- Тот элемент, который будет обнаружен при совпадении, определяет, может ли пользователь получить доступ к ресурсу. То есть, если обнаружили при совпадении элемент <allow>, то может, если <deny> - то нет.

Отметим, что если клиент аутентифицируется, то информация о нем доступна посредством свойства User класса Page или класса HttpContext. Свойство User указывает на реализацию интерфейса IPrincipal. Этот интерфейс содержит одно свойство и один метод. Свойство Identity является указателем на реализацию интерфейса IIdentity, а метод IsInRole() проверяет членство клиента в указанной группе. Ниже приведено описание интерфейса IIdentity:

```
public interface IIdentity
{
    string AuthenticationType { get; }
    bool IsAuthenticated { get; }
    string Name { get; }
}
```

Свойство IsAuthenticated используется для различения аутентифицируемых и анонимных клиентов. Свойство Name можно применить для выяснения идентичности клиента. Если клиент аутентифицируется, то свойство AuthenticationType позволит выяснить тип аутентификации.

Вернемся к рассмотрению различных типов аутентификации и остановимся на режиме аутентификации Forms. Принцип действия аутентификации в этом режиме следующий:

1. Когда пользователь первый раз запрашивает ресурс, требующий аутентификации, сервер перенаправляет запрос в выделенную страницу регистрации.
2. Регистрационная страница принимает данные пользователя (как правило, имя и пароль), а затем приложение аутентифицирует пользователя (предположительно с помощью базы данных).
3. Если пользователь успешно зарегистрировался, сервер предоставляет ему аутентификационный файл cookie в зашифрованном виде, который действителен на протяжении сеанса (но может быть сохранен на клиентском компьютере для использования в последующих сеансах).
4. Пользователь перенаправляется к интересующему его ресурсу, однако теперь он предоставляет в запросе идентификационный cookie и получает доступ.

Если в файле web.config задать аутентификацию Forms, то дополнительные настройки режима можно выполнить во вложенном элементе <forms>, атрибуты которого содержит таблице.

Атрибуты элемента <forms>

Атрибут	Значения	Значение по умолчанию	Описание
name	Строка	.ASPXAUTH	Имя файла cookie
loginUrl	Адрес URL	login.aspx	Адрес URL страницы регистрации
protection	All, None, Encryption, Validation	All	Режим защиты файла cookie
timeout	Минуты	30	Скользящее время жизни файла cookie (сбрасывается при каждом запросе)
path	Маршрут	/	Маршрут файла cookie

Пример файла web.config, конфигурирующего аутентификацию в режиме Forms, показан в следующем листинге:

```
<configuration>
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx"/>
    </authentication>
  </system.web>
</configuration>
```

Теперь, чтобы аутентификация на основе cookie заработала, нужно реализовать страницу регистрации. Для реализации страницы регистрации платформа ASP.NET предоставляет класс FormsAuthentication. Чтобы предоставить аутентификационный файл клиенту, нужно вызвать метод SetAuthCookie(). Метод RedirectFromLoginPage() предоставляет cookie клиенту и перенаправляет клиента в запрошенную им страницу.

Приведем пример страницы регистрации.

```
<!-- Файл login.aspx -->
<%@ Page Language="C#" %>

<script runat="server">
    protected void OnClick_Login(object src, EventArgs e)
    {
        if ((m_usrname.Text == "Alex") && (m_pass.Text == "pass"))
        {
            FormsAuthentication.RedirectFromLoginPage(
                                m_usrname.Text,
m_save_pass.Checked);
        }
        else
        {
            msg.Text = "Неверно. Попытайтесь еще раз";
        }
    }
</script>

<html>
<body>
    <form id="Form1" runat="server">
    <h2>Страница регистрации</h2>
    Имя пользователя:
    <asp:TextBox ID="m_usrname" runat="server" /><br />
    Пароль:
    <asp:TextBox  ID="m_pass"  TextMode="Password"  runat="server"
/>
    <br />
    Запомнить пароль?
    <asp:CheckBox ID="m_save_pass" runat="server" /><br />
```

```
        <asp:Button ID="Button1" Text="Регистрация"
                    OnClick="OnClick_Login" runat="server" /><br />
        <asp:Label ID="msg" runat="server" />
        </form>
    </body>
    </html>
```

Файловая авторизация

Авторизация на основе URL - один из краеугольных камней авторизации ASP.NET. Однако в ASP.NET также используется другой тип авторизации, который часто пропускается или игнорируется многими разработчиками. Это авторизация на основе файлов, реализуемая модулем **FileAuthorizationModule**. Авторизация на основе файлов работает, только в случае применения Windows-аутентификации.

Чтобы понять суть файловой авторизации, необходимо разобраться, как операционная система Windows обеспечивает безопасность файловой системы. В случае файловой системы NTFS можно установить списки ***ACL (access control list - список контроля доступа)***, указывающие идентичность пользователей и ролей, которым открыт или запрещен доступ к индивидуальным файлам. Модуль FileAuthorizationModule просто проверяет права доступа к запрошенному файлу, определенные Windows.

Например, если запрашивается веб-страница, FileAuthorizationModule проверяет, имеет ли текущий аутентифицированный IIS пользователь права доступа к соответствующему файлу .aspx. Если не имеет, то код страницы не выполняется и пользователь получает сообщение о запрете доступа.

Чтобы понять необходимость в модуле FileAuthorizationModule, необходимо вспомнить, как ASP.NET выполняет код. Если не включено заимствование прав, ASP.NET выполняется от имени фиксированного пользовательской учетной записи, такой как ASPNET. Операционная система Windows будет проверять, имеет ли учетная запись ASPNET права, необходимые для доступа к файлу .aspx, но она не выполнит ту же проверку для пользователя, аутентифицированного IIS. Модуль FileAuthorizationModule заполняет этот пробел. Он осуществляет проверку авторизации с учетом контекста безопасности текущего пользователя. В результате системный администратор может устанавливать права доступа к файлам или папкам и контролировать доступ к частям приложения ASPNET. Обычно проще и удобнее использовать правила авторизации в файле web.config. Однако если необходимо воспользоваться преимуществами существующих привилегий Windows в локальной или корпоративной сети, то это можно сделать.

УПРАВЛЕНИЕ ЧЛЕНСТВОМ И РОЛЯМИ

Наиболее заметным изменением, внесенным в механизм аутентификации Forms, является введение дополнительного API, а именно *API управления членством и ролями*.

Используемые совместно с классом `FormsAuthentication` новые классы `Membership` и `Roles` составляют полный арсенал необходимых разработчикам ASP.NET

средств защиты. Класс Membership предоставляет методы для управления учетными записями пользователей, в частности для добавления учетных записей новых пользователей, а также для удаления и редактирования существующих записей. Класс Roles служит связующим звеном между пользователями и их ролями. Все свойства класса Membership являются статическими и доступными только для чтения.

Таблица 44

Свойства класса Membership

Свойство	Описание
ApplicationName	Строка, идентифицирующую приложение. По умолчанию содержит путь к его корневой папке
EnablePasswordReset	Возвращает значение true, если поставщик поддерживает сброс паролей
EnablePasswordRetrieval	Возвращает значение true, если поставщик поддерживает восстановление паролей
MaxInvalidPasswordAttempts	Максимальное количество попыток ввода пароля перед блокированием пользователя
MinRequired-NonAlphanumericCharacters	Минимальное число знаков препинания в пароле
MinRequiredPasswordLength	Минимальная длина пароля
PasswordAttemptWindow	Время в минутах, в течение которого пользователь может пытаться ввести пароль, прежде чем будет заблокирован
PasswordStrength-RegularExpression	Регулярное выражение, которому должен отвечать пароль
Provider	Экземпляр используемого поставщика
Providers	Коллекция зарегистрированных поставщиков
RequiresQuestionAndAnswer	Возвращает значение true, если при восстановлении или сбросе пароля поставщик требует ответа на определенный вопрос
UserIsOnlineTimeWindow	Возвращает время в минутах, истекшее после последнего действия пользователя, в течение которого пользователь все еще будет считаться подключенным

Рассмотрим примеры кода. Начнем с наиболее простой операции - аутентификации. Применяя аутентификационные функции API членства, можно написать следующую функцию входа:

```
private void LogonUser(object sender, EventArgs e)
{
    string user = userName.Text;
    string pswd = password.Text;
    if (Membership.ValidateUser(user, pswd))
    {
        FormsAuthentication.RedirectFromLoginPage(user, false);
    }
    else
    {
        errorMsg.Text = "Sorry, not a valid account.";
    }
}
```

Для программного создания новой учетной записи достаточно написать `Membership.CreateUser(userName, pswd)`. Удаление учетной записи пользователя выполняется с помощью метода `Membership.DeleteUser()`.

Для получения информации об определенном пользователе используется метод `GetUser()`. Он принимает имя пользователя и возвращает объект `MembershipUser`.

```
MembershipUser user = Membership.GetUser("TheUser");
```

Получив объект `MembershipUser`, вы обладаете всей необходимой информацией о пользователе и можете программно изменять его пароль и прочие сведения.

Методу `ChangePassword()` передается старый пароль. В некоторых случаях можно предоставить пользователю возможность просто сбросить пароль (то есть удалить старый и автоматически сгенерировать новый), вместо того чтобы его изменять. Для этого используется метод `ResetPassword()`:

```
MembershipUser user = Membership.GetUser("TheUser");
string newPswd = user.ResetPassword();
```

Для включения поддержки ролей необходимо добавить в файл `web.config` приложения следующую строку:

```
<roleManager enabled="true" />
```

Блок `<authorization>` определяет, что только члены роли `Admin` имеют доступ к страницам, на которые распространяется действие файла `web.config`:

```
<authorization>
```

```
    <allow roles="Admin" />
    <deny users="*" />
</authorization>
```

В ASP.NET применяется API управления ролями, представленный членами класса `Roles`. Когда управление ролями включено, ASP.NET создает экземпляр этого класса и добавляет его в контекст каждого запроса, то есть в объект `HttpContext`. Следующий код показывает, как программным способом создать роли `Admin` и `Guest` и заполнить их именами пользователей:

```
Roles.CreateRole("Admin");
Roles.AddUserToRole("TheUser", "Admin");
Roles.CreateRole("Guest");
var guests = new[] { "UserX", "Godzilla" };
Roles.AddUsersToRole(guests, "Guest");
```

Во время выполнения страницы информация о запросившем ее пользователе и его ролях доступна через объект `User` контекста HTTP. Следующий код показывает, как установить, принадлежит ли пользователь к определенной роли, и включить соответствующие функции:

```
if (User.IsInRole("Admin"))
// включаем функции, специфические для данной роли
```

Все свойства класса `Roles` являются статическими и доступны только для чтения. Значения этих свойств соответствуют установкам из конфигурационного раздела `<roleManager>`.

Свойства класса `Roles`

Свойства	Описание
`ApplicationName`	Возвращает имя приложения
`CacheRolesInCookie`	Возвращает значение `true`, если хранение данных ролей в cookie разрешено
`CookieName`	Определяет имя cookie, используемого для хранения информации о ролях
`CookiePath`	Определяет путь, ассоциированный с ролевым cookie
`CookieProtectionValue`	Содержит установку шифрования ролевого cookie: `All`, `Clear`, `Hashed` или `Encrypted`

CookieRequreSSL	Указывает, должны ли ролевые cookie передаваться через SSL-соединение
CookieSlidingExpiration	Указывает, является срок действия ролевого cookie фиксированным или скользящим
CookieTimeout	Возвращает срок действия ролевого cookie в минутах
CreatePersistentCookie	Создает ролевой cookie, сохраняющийся в течение сеанса
Domain	Определяет домен ролевого cookie
Enabled	Указывает, включена ли функция управления ролями
MaxCachedResults	Определяет максимальное количество ролей, которое может быть сохранено для одного пользователя в cookie-файле
Provider	Возвращает текущий поставщик ролей
Providers	Возвращает список поддерживаемых поставщиков ролей

Конфигурация приложений ASP.NET

Существует два типа конфигурации:

- Server configuration (Конфигурация сервера). В этой конфигурации определяются стандартные значения параметров, которые будут использоваться всеми ASP NET-приложениями.
- Application configuration (Конфигурация приложения) В этой конфигурации определяются значения параметров, которые будут использоваться конкретным ASP.NET-приложением.

Файлы конфигурации

Сведения о конфигурации сервера и приложения хранятся в файлах в формате XML. Содержимое файлов можно легко прочесть, и, в случае необходимости, изменить.

Файл конфигурации сервера

Файл конфигурации сервера имеет название machine.config. Этот файл находится в каталоге \WINNT\Microsoft..NET\Framework. Каждая версия платформы .NET имеет свой каталог, в котором хранится файл конфигурации сервера. За счет этого можно одновременно использовать различные версии ASP.NET. Иными словами, вы можете продолжать использовать Web-приложения, которые работают с предыдущей версией платформы .NET, и одновременно разрабатывать приложения, использующие более новую версию.

Файлы конфигурации приложения

Чтобы сохранить значения параметров, которые используются конкретным Web-приложением, в корне виртуального каталога нужно создать файл web. conf ig. Если этот файл отсутствует, будут использоваться значения параметров конфигурации

приложения, принятые по умолчанию. Они хранятся в файле machine. conf ig. Если же файл web.config существует, то будут использованы значения параметров, которые содержатся в нем.

Формат файлов конфигурации

Файл web.config, также как и файл machine, config, хранится в формате XML. Файл состоит из разделов. В каждом разделе объединены взаимосвязанные параметры. Чтобы ознакомиться со структурой файла конфигурации и параметрами, значение которых можно изменить, просмотрите файл web. conf ig, он был создан средой Visual Studio, когда мы создавали новый проект ASP.NET Web-приложения.

```
<?xml version="l.0" encoding="utf-8" ?>
<!— версия = "1.0" KOflnpOBKa="utf-8" —>
<configuration> <!— конфигурация -->
<system.web>
<!— CUSTOM ERROR MESSAGES
Set mode="on" or "remoteonly" to enable custom
error messages, "off" to disable. Add
<error> tags for each of the errors you want to
handle.
-->
<!-- ПОЛЬЗОВАТЕЛЬСКИЕ СООБЩЕНИЯ ОБ ОШИБКАХ
Установите режим набора = "on" или "remoteonly",
чтобы разрешить пользовательские
сообщения об ошибках, "off" чтобы отключить их. Добавьте
тэги <error> (<ошибка>) для каждой из ошибок,
которую вы хотите обрабатывать.
-->
<customErrors
mode="0ff"
/>
<!-- AUTHENTICATION
This section sets the authentication policies of the application.
Possible modes are "Windows", "Forms", "Passport" and "None"
-->
<!-- ОПОЗНАВАНИЕ
Этот раздел устанавливает политику опознавания
приложения. Возможные режимы - "Windows", *
"Forms" ("Формы"), "Passport" ("Паспорт")
и "None" ("Никакой")
—>
<authentication mode="None" />
<!-- режим опознавания = "Никакой" -->
</system.web>
</configuration> <!— конфигурация —>
```