Sitecore CMS 6

Low-level CMS Security and Custom Providers

An introduction to low-level security and custom security provider concepts

Table of Contents

Chapter 1

Introduction

Sitecore CMS 6 security model is based on the ASP.NET security provider model and inherits all the advantages of the latter. This document explains the way the security providers are configured by default and explains how to create custom providers and integrate them into Sitecore CMS security.

This document assumes that you are familiar with the basic concepts of the ASP.NET 2.0 provider model.

This document contains the following chapters:

- Default security providers configuration in Sitecore CMS.
- Multiple security storages configuration.
- Writing custom security providers for Sitecore.

Chapter 2

Default Security Configuration

The default Sitecore CMS 6 installation is configured to use the ready-made Microsoft SQL set of providers for each of the following services: membership, roles and profiles.

The default Sitecore CMS 6 installation may also include the SQLite version of the security and database engines. The explanations below apply to SQLite as well.

2.1 Membership Service

The membership service is responsible for creating/updating/deleting of users, validating the user credentials, searching the users by different patterns, resetting user passwords, etc. All these operations are defined in the MembershipProvider base class, which all the membership providers must inherit.

Open the `web.config` file and navigate to the `system.web/membership` element in the. This is how this section looks like in a clean Sitecore CMS installation:

```
<membership defaultProvider="sitecore">
   <providers>
       <clear />
        <add name="sitecore" type="Sitecore.Security.SitecoreMembershipProvider,
Sitecore.Kernel" realProviderName="sql" providerWildcard="%" raiseEvents="true" />
       <add name="sql" type="System.Web.Security.SqlMembershipProvider"
            connectionStringName="core" applicationName="sitecore"
            minRequiredPasswordLength="1" minRequiredNonalphanumericCharacters="0"
            requiresQuestionAndAnswer="false" requiresUniqueEmail="false" />
       <add name="switcher" type="Sitecore.Security.SwitchingMembershipProvider"
            applicationName="sitecore" mappings="switchingProviders/membership" />
   </providers>
</membership>
```

Each provider definition must be added as the `<add/>` element inside the `<providers>` group of the `<membership>` element. The `<clear/>` element in the beginning instructs the ASP.NET engine to clear the previous provider definitions defined on a higher level of the configuration files (for instance, in `machine.config` file).

As you can see, there are three providers defined in this element. The provider named `sitecore` is a CMS system provider used to handle security caching correctly. The provider named `switcher` is used to handle multiple providers and it will be covered later in this article. We will describe the `sql` provider first.

The table below describes the `sql` provider attributes defined by default.

Attribute	Description
name	The name of the provider. It must be unique within the entire set of providers defined. The default value of this element for the Microsoft provider is `sql`.
type	The full name of the class which contains the implementation of the provider contract. The Microsoft SQL provider used in Sitecore CMS by default is implemented by the class `System.Web.Security.SqlMembershipProvider`
connectionStringName	The name of the connection string to the database where the security information is stored. The Microsoft SQL set of providers defines a number of tables to be present in the database in order to use it for storing security. These tables are by default located in the `core` database of Sitecore CMS.
applicationName	The area of the data visibility for a provider. If you wish the custom security data (users/roles) to be visible for Sitecore security tools, you should specify the `sitecore` (default) value for this attribute. The topic of “independent configuration” is described later in this article.
minRequiredPasswordLength	The minimum number of characters each user

Attribute	Description
	password must contain. The default value for SQL provider is `1`. That is why the default `Admin` account has the “b” password instead of a blank password by default.
minRequiredNonalphanumericCharacters	The number of non-alphanumeric characters the user passwords must contain. Non-alphanumeric characters include a set of symbols which are not letters and not digits. The default value of this attribute is “0”.
requiresQuestionAndAnswer	This attribute defines if the question and answer should be provided in the “create user” functionality. Default value is `false`.
requiresUniqueEmail	This attribute defines if each user must have a unique email. The default value is `false`.

The ASP.NET provider model article contains more information about these and other attributes of the membership provider.

The point which requires attention here is the `defaultProvider` attribute of the `<membership>` element. This provider is the one to be used by ASP.NET login controls. There are a number of static classes (for instance, `System.Web.Security.Membership`), which also rely on the provider specified in the `defaultProvider` attribute.

This attribute should have the `sitecore` value for Sitecore CMS 6. This is necessary to handle the security caching correctly. But when writing the code for your custom services you should assume that the value specified in the `realProviderName` attribute of the `sitecore` provider is considered to be the value for the `defaultProvider` attribute of the service.

The default value of the `realProviderName` attribute is `sql`, as it is the only provider defined in a blank installation. The requirement for this attribute is that the provider with such name must be present among the providers in the `<providers>` section.

Sitecore high-level security API also works through the default provider.

The login page of the `shell` site, for example, contains the standard ASP.NET login control, which calls the `ValidateUser` method of the default membership provider in order to verify the user’s credentials provided.

Another example: when you wish to edit a certain user and press the appropriate button in the ribbon, the system performs a number of actions, one of which is getting all the roles this user is a member of. In order to accomplish this operation, Sitecore API addresses the `Roles` class defined in `System.Web.Security` namespace and calls the method `GetRolesForUser`. This call also goes through the default provider of the RoleManager.

2.2 RoleManager Service

The role provider is responsible for creating and deleting roles, adding the users to or removing the users from roles, serving the requests of getting the users for a role and getting the roles for a user, etc. Similar to membership service, all these operations are defined as methods in a single base *RoleProvider* class, which all the role providers must inherit from.

The definition element of the *RoleManager* service is located just after the membership element in the web.config. Open the web.config file, and browse for the <roleManager> element of the <system.web> section. The default definition of this element looks similar to this:

```
<roleManager defaultProvider="sitecore" enabled="true">
   <providers>
      <clear />
        <add name="sitecore" type="Sitecore.Security.SitecoreRoleProvider, Sitecore.Kernel"
realProviderName="sql" raiseEvents="true" />
      <add name="sql" type="System.Web.Security.SqlRoleProvider" connectionStringName="core"
           applicationName="sitecore" />
      <add name="switcher" type="Sitecore.Security.SwitchingRoleProvider, Sitecore.Kernel"
           applicationName="sitecore" mappings="switchingProviders/roleManager" />
   </providers>
</roleManager>
```

All the attributes here have the same meaning as described in the membership service section. The only different attribute is the *enabled* attribute of the *<roleManager>* element. It simply indicates if the service is enabled. The default value is *true*. This attribute should be set to *false* only if you want to disable all the functionality related to the role management in Sitecore CMS.

This service is addressed anytime the action related to the role management is required, for example, when you create a role in the *RoleManager* security tool, or when you edit the user membership, etc.

2.3 Profile Service

Each user in any security system contains a set of properties which store its preferences, settings and status. These properties compose a so-called user profile. ASP.NET 2.0 functionality allows storing the user and its profile separately.

The profile service is defined in the *<system.web>* section of web.config file. The profile definition should look like this by default:

```
<profile defaultProvider="sql" enabled="true" inherits="Sitecore.Security.UserProfile,
Sitecore.Kernel">
   <providers>
      <clear />
      <add name="sql" type="System.Web.Profile.SqlProfileProvider" connectionStringName="core"
           applicationName="sitecore" />
      <add name="switcher" type="Sitecore.Security.SwitchingProfileProvider"
           applicationName="sitecore" mappings="switchingProviders/profile" />
   </providers>
   <properties>
      <clear />
      <add type="System.String" name="SC_UserData" />
   </properties>
</profile>
```

Unlike the two other services mentioned earlier in this document, the profile definition contains two groups, *<providers>* and *<properties>*. The *<providers>* group contains the definitions of the profile providers, just similar to the membership and role sections. The *<properties>* section contains additional properties the user profile should hold.

The profile object for each user is generated automatically at runtime basing on the information provided in the *profile* section. The `inherits` attribute defines a base class for the user profile. Hence, each profile object contains the set of properties implemented in that class (core properties). Plus, any number of custom properties can be added to this core set. Such custom properties must be defined inside the *<properties>* section. The table below explains some of the possible attributes of the custom properties:

Attribute	Description
name	The name of the property. It must be unique within the whole set of the properties (core and custom), otherwise an unhandled exception is thrown.
type	The .NET type this property will obtain in the code. For instance, if the `System.String` is entered as a value, the property will be of the `string` type for the ASP.NET engine.
provider	The name of the profile provider the property is related to. If this attribute is specified, then other profile providers will not see this property at all. If it is not specified, the default profile provider name is used. This attribute plays an important role in the independent configuration approach described later in this article.
customProviderData	The value of this attribute is passed to the profile provider which handles this property. The format of this attribute is not strictly defined, and it is a developer who is responsible for parsing this value and extracting necessary information. One of the possible usages is defining a custom property native type, which may differ from the .NET implementation.
readOnly	It is an optional attribute, which defines whether the property can be modified by a user. The default value is `false`, thus the user can read and write to this property.

The *readOnly* attribute allows the safe profile configuration. For example, there are properties you would like to display for every user of your security domain, but you want to deny the option to modify these properties. This is done by setting the `readOnly` attribute value to `true` for such properties.

You code may not take care about handling these situations – everything is resolved on a configuration level.

Chapter 3

Multiple Provider Configuration

The default security configuration provides all necessary basic options to build the solution infrastructure. But there are plenty of cases when it is necessary to plug the custom security storage in. Imagine a situation when your company already has a tuned and comprehensive security model (it can be the Active Directory domain, custom database or a free schema, xml file, etc.) and you would like to reuse this storage as well as the code which works with it.

This section describes the options offered by Sitecore CMS security model to support these scenarios.

3.1 Switching Providers Description

The key concept in the multiple providers configuration is the mechanism called switching providers. The functionality of this feature is described below:

- One switching provider is defined for each service: membership, role and profile.
- Each switching provider is a common provider which inherits the same base class (for each service appropriately).
- The main purpose of each switching provider is to get all security requests from the higher level of the application, define which provider(s) should handle the request and redirect a request to that (those) provider(s).

Let’s illustrate this situation with some examples.

You would like to have all the roles (Sitecore CMS default roles and your custom ones) be visible and accessible from the RoleManager security tool in a site shell. To make this possible, the switching provider should receive the request, and its GetAllRoles() method should iterate the available providers and concatenate the retrieved collections of roles.

Or, for instance, you validate the user credentials and you would like to be sure that the username and the password are validated against correct membership provider. In order to make this possible, the switching provider must receive the request, resolve which provider should handle this request and redirect it to that provider.

And finally, you are editing the user’s profile and you would like to be sure that the base properties, stored in default SQL database, will get there, while your custom properties, stored in Active Directory domain, will be updated accordingly. The switching provider receives the initial request, iterates the properties passed with this request and redirects the properties to the providers they came from.

3.2 Configuring Switching Providers

Now, when the aim of the switching providers is clear, let’s study the way to activate this feature.

Note
The switching provider feature is OFF by default. The way to turn it ON is described in this section.

You may notice that the section of each service in web.config contains a definition of the provider named *switcher*. In order to activate the multiple providers environment, you should first set the switching providers ON. To do this, set the *realProviderName* attribute of the *sitecore* provider in each service (*membership* and *roleManager*) to *switcher*.

The definition of the switching provider contains the same set of attributes as any other provider of the appropriate type. The only extra attribute is `mappings`. It stores the relative path to the section in web.config, where the mappings between Sitecore CMS domains and ASP.NET providers are defined. The default value of this attribute for membership service is `switchingProviders/membership`, for role manager service – `switchingProviders/roleManager`, for profile service – `switchingProviders/profile`. This path should be written starting from the *sitecore* parent section.

Let’s now browse to the mappings definition to investigate how it works under the hood. Open the web.config file, and browse to the *switchingProviders* element of the *sitecore* section. You’ll find something like the following:

```
<!-- SWITCHING PROVIDERS -->
<switchingProviders>
   <membership>
      <provider providerName="sql" storeFullNames="true" wildcard="%" domains="*" />
   </membership>
   <roleManager>
      <provider providerName="sql" storeFullNames="true" wildcard="%" domains="*" />
   </roleManager>
   <profile>
      <provider providerName="sql" storeFullNames="true" wildcard="%" domains="*" />
   </profile>
</switchingProviders>
```

The element contains 3 groups of mappings, one per service: membership, roleManager and profile. Each group contains the `<provider/>` elements of the following format:

Attribute	**Description**
name	The name of the provider, defined in the *system.web* section. When the system parses this configuration section, it verifies that the provider definition exists. Otherwise it throws an exception. By default only one mapping is defined and it targets at the only provider available – `sql`.
storeFullNames	The attribute defines whether the storage represented by the provider holds full names (domain\name) or just short names (name). In the former case the switching provider is responsible for trimming the domain name when passing the request to the appropriate provider (so-called “localizing”) and prefixing the entity name with the domain prefix when returning the result (so-called “globalizing”). The `sql` provider stores the full names and thus this attribute is set to `true`. But any custom storages, which are not aware of the Sitecore CMS domains (like Active Directory), store just the short entity names and thus will require this attribute to be set to `false`.

wildcard	The symbol used as wildcard in the search requests as a part of a pattern. Each provider may have a different symbol used for this purpose. For instance, the `sql` provider expects the “`%`” character, while the `ad` provider accepts only the asterisk (“`*`”). That is why this setting has been moved to a separate attribute on a provider level. The default value for the `sql` provider mapping is “`%`”.
domains	The list of domains handled by a certain provider must be defined here. For instance, say, you have plugged in a custom security storage and created a domain in Sitecore CMS for this storage (called *ad* for example). As a next step you should specify this domain in the *domains* attribute value. If your storage handles the users/roles from more than one domain, you should specify all of them here, separating them with commas. Note, that you cannot specify the same domain for two or more providers. The `sql` provider by default handles all the entities, thus “`*`” is specified there. This means that the requests from all the Sitecore CMS domains must be directed to the *sql* provider.

Let’s sum up the things said above:

- If the *domains* attribute contains more than one domain listed or the asterisk (“*”), the *storeFullNames* attribute must have the *true* value.
- Each provider can handle more than one domain, but each domain can be served by only one provider. Thus, we have the relation “N domains to 1 provider” here, where N is more or equal to one and less than infinity.
- The *domains* attribute must always be present, so that the switching provider knows which domains are served by a certain provider.

Important
The option to have different sets of user profile properties in different storages is not supported by the default Sitecore CMS installation. You must install the Active Directory module and configure it accordingly in order to use this option (see the Active Directory module documentation).

Chapter 4

Configuring Custom Providers

This chapter describes how to configure custom providers.

4.1 Introduction

The switching provider mechanism described in the previous section will not make sense when only one provider is defined. To take the full advantage of the feature, let's plug in custom providers for each service and see how this stuff works together.

First of all, there are two types of the custom providers' configuration. The first one is used when the custom provider is independent. This means that the custom storage contains the entire Sitecore CMS security (Sitecore Client roles, predefined users, etc.) and thus is self-sufficient. Such kind of the custom solution is usually developed to replace the standard user profile totally. For instance, if you wish to move the Sitecore CMS security entirely to Oracle or another RDBMS.

Such kind of solution doesn't require the switching provider functionality at all. Everything you need to do is to replace the standard *sql* provider definitions in *web.config* with the custom ones. There's nothing special in this configuration and it is not really a topic of this article.

A different approach should be taken when you want not to replace, but to extend the existing model. The following steps define the procedure you should follow in this case:

- Develop a membership provider (and configure it in the *system.web* section of *web.config*). The provider should inherit a *MembershipProvider* class from *System.Web.Security* namespace and should have all the virtual methods/properties overridden. This provider is not aware of Sitecore CMS and should not be. It should work with the custom storage in a chosen way and should not take care about where it is used.
- Develop a role provider (and configure it in the *system.web* section of web.config). It should inherit the *RoleProvider* class from the *System.Web.Security* namespace and should have all the virtual methods/properties overridden. Again, do not take care about Sitecore CMS at this point.
- [Optional] Develop a profile provider (and configure it in the *system.web* section of *web.config*). It should inherit a ProfileProvider class and must follow some special rules of implementation and configuration (more about this later in this article). Note that you are able to use the partial profile feature only if the Active Directory module is installed.
- Define a Sitecore CMS domain which the custom users/roles will be related to. If your set of custom providers stores the short names only (storeFullNames="false"), you should create a Sitecore domain and choose some name for it.
  If the full user names are stored, the domain name must correspond to the prefix of the full user name in external storage. If there are users or roles with different prefixes, all the appropriate domains must be created in Sitecore CMS.
- Edit the domain-provider mappings: add the membership and role provider mappings in the `<membership>` and `<roleManager>` groups. The order of the mappings is not important, it only influences the default order of the users/roles in the appropriate Sitecore CMS security applications.
- [Optional] Edit the domain-provider mappings: add the profile provider mapping to the `<profile>` group. This line must go before the provider which can handle all requests (*). Otherwise, the custom properties will be saved to the common storage and the relation to the custom storage will be lost.

4.2 Profile Provider Implementation Recommendations

Profile providers have some peculiarities in their implementation compared to other services. In order to develop a profile provider which supports partial profiles, you follow the next recommendations:

- Configuration

  i) Each property, which is to be served by this provider, must fill the attribute `customProviderData` with the value understandable by the provider. For instance, if it is to be an Active Directory property, start the attribute value with `ad|`.

  ii) The provider mapping must go before the general profile provider, which can handle any defined property.

- Implementation
  Any provider receives the entire collection of the properties. Each profile provider must filter its own properties (for instance, those starting from ad|) and remove them from the general collection after it handles the request. This will guarantee that other providers will not receive odd properties and will not save them to another storage.

4.3 Independent Configurations

Imagine a case, when you want to plug in an external application to the Sitecore CMS solution. This may be a kind of a module, which already uses the ASP.NET security provider model in its own way. Besides, you don't want Sitecore CMS to be aware of this module at all. In such case, you may have a conflict, and in order to resolve it, you may have to modify that external component a bit.

IMPORTANT
If there is no access to the external component sources, it may appear impossible to use this component in the independent configuration scenario described here.

Here are some general recommendations:

- Make sure this external component addresses its providers directly. That is not via the well-known static classes (`Membership, Roles`), but calling the necessary provider explicitly (`Membership.Providers["providername"]`)
- In the provider definition, the `applicationName` attribute must not be equal to any of Sitecore CMS providers, that is not to be `sitecore` in general case.
- The custom properties of the user profile, defined by this external component, must have the attribute `provider` set to the corresponding provider so that other (Sitecore CMS) providers can't see them.
- And finally, do not add any mappings with this external provider into the Sitecore CMS domain-provider map.

When this is done, your Sitecore CMS solution and this external component will not interfere.