

---

## CHAPTER 16

---

# SECURING COMPONENTS

### **Difficulty**

Sprinter

### **Skills**

- Be Knowledgeable about topics covered in the previous chapters of the “Content Components” section.
- Be familiar with interfaces, ZCML and some of the security concepts, such as permissions, roles and principals.

### **Problem/Task**

While we had to make basic security assertions in order to get our message board to work, it does not really represent a secure system at this point. Some end-user views require the `zope.ManageContent` permission and there are no granular roles defined.

### **Solution**

Zope 3 comes with a flexible security mechanism. The two fundamental concepts are permissions and principals. Permissions are like keys to doors that open to a particular functionality. For example, we might need the permission `zope.View` to look at a message’s detail screen. Principals, on the other hand, are agents of the system that execute actions. The most common example of a principal is a user of the system. The goal is now to grant permissions to principals, which is the duty of another sub-system known as the `securitypolicy`.

Zope 3 does not enforce any particular security policy. In contrary, it encourages site administrators to carefully choose the security policy and use one that fits their needs best. The default Zope 3 distribution comes with a default security policy (`zope.app.securitypolicy`) that supports the concept of roles. Roles are like hats people wear, as Jim Fulton would say, and can be seen as a collection of permissions. A single user can have several hats, but only wear one at a time. Prominent examples of roles include “editor” and “administrator”. Therefore, the default security policy supports mappings from permissions to principals, permissions to roles, and roles to principals. This chapter will use the default security policy to setup the security, but will clearly mark the sections that are security policy specific.

The first task will be to define a sensible set of permissions and change the existing directives to use these new permissions. This is a bit tedious, but it is important that you do this carefully, since the quality of your security depends on this task. While doing this, you usually discover that you missed a permission and even a role, so do not hesitate to add some. That is everything the programmer should ever do. The site administrator, who uses the default security policy, will then define roles and grant permissions to them. Finally the roles are granted to some users for testing.

Securing an object does not require any modification to the existing Python code as you will see going through the chapter, since everything is configured via ZCML. Therefore security can be completely configured using ZCML, leaving the Python code untouched, which is another advantage of using Zope 3 (in comparison to Zope 2, for example).

## 16.1 Step I: Delcarations of Permissions

Other than in Zope 2, permissions have to be explicitly defined. For our message board it will suffice to define the following four basic permissions:

- **View** – Allow users to access the data for message boards and messages. Every regular message board `User` is going to have this permission.
- **Add** – Allows someone to create (i.e. post) a message and add it to the message board or another message. Note that every regular `User` is allowed to do this, since posting and replying must be possible.
- **Edit** – Editing content (after it is created) is only a permission the message board `Editor` possesses (for moderation), since we would not want a regular user to be able to manipulate posts after creation.
- **Delete** – The `Editor` must be able to get rid of messages, of course. Therefore the `Delete` permission is assigned to her. Note that this permission does not allow the editor to delete `MessageBoard` objects from folders or other containers.

## 16.2. USING THE PERMISSIONS

Let's define the permissions now. Note that they must appear at the very beginning of the configuration file, so that they will be defined by the time the other directives (that will use the permissions) are executed. Here are the four directives you should add to your main `configure.zcml` file:

```
1 <permission
2   id="book.messageboard.View"
3   title="View Message Board and Messages"
4   description="View the Message Board and all its content."
5 />
6 <permission
7   id="book.messageboard.Add"
8   title="Add Message"
9   description="Add Message."
10 />
11 <permission
12   id="book.messageboard.Edit"
13   title="Edit Messages"
14   description="Edit Messages."
15 />
16 <permission
17   id="book.messageboard.Delete"
18   title="Delete Message"
19   description="Delete Message."
20 />
```

The `zope:permission` directive defines and creates a new permission in the global permission registry. The `id` should be a unique name for the permission, so it is a good idea to give the name a dotted prefix, like `book.messageboard`. in this case. Note that the `id` must be a valid URI or a dotted name – if there is no dot in the dotted version, a `ValidationError` will be raised. The `id` is used as identifier in the following configuration steps. The `title` of the permissions is a short description that will be used in GUIs to identify the permission, while the `description` is a longer explanation that serves more or less as documentation. Both the `id` and `title` are required attributes.

## 16.2 Step II: Using the Permissions

Now that we have defined these permissions, we also have to use them; let's start with the main message board configuration file (`messageboard/configure.zcml`). In the following walk-through we are only going to use the last part of the permission name to refer to the permission, leaving off `book.messageboard`. However, the full `id` has to be specified for the configuration to execute.

- Change the first `require` statement of in the `MessageBoard` content directive to use the `View` permission (line 42). This makes the `description` and the items accessible to all board users. Similarly, change line 64 for the `Message`.

- Change the permission of line 46 to `Edit`, since only the message board administrator should be able to change any of the properties of the `MessageBoard` object.
- All the container functionality will only require the view permission, so change the permission on line 68 to `View`. This is unsecure, since this includes read and write methods, but it will suffice for this demonstration.
- For the `Message` we need to be able to set the attributes with the `Add` permission, so change line 72 to specify this permission.

Now let's go to the `browser` configuration file (`messageboard/browser/configure.zcml`) and fix the permissions there.

- The permissions for the message board's add form (line 11), add menu item (line 18), and its edit form (line 27) stay unchanged, since only an administrator should be able manage the board.
- Since we want every user to see the messages in a messageboard, the permission on line 33 should become `View`. Since the `contents` view is meant for management, only principals with the `Edit` permission should be able to see it (line 34). Finally, you need the `Add` permission to actually add new messages to the message board (line 35). The same is true for the message's container views permissions (line 84–86).
- Since all user should be able to see the message thread and the message details, the permissions on line 43, 94, and 106 should become `View`.
- On line 61 you should change the permission to `Add`, because you only allow messages to be added to the message board, if the user has this permission. The same is true for the message's add menu item on line 68.
- On line 78 make sure that a user can only access the edit screen if he has the `Edit` permission.

That's it. If you would restart Zope 3 at this point, you could not even access the `MessageBoard` and/or `Message` instances. Therefore we need to create some roles next and assign permissions to them.

### 16.3 Step III: Declaration of Roles

The declaration of roles is specific to Zope 3's default security policy. Another security policy might not even have the concept of roles at all. Therefore, the role

## 16.3. DECLARATION OF ROLES

declaration and grants to the permissions should not even be part of your package. For simplicity and keeping it all at one place, we are going to store the policy-specific security configuration in `security.zcml`. For our message board package we really only need two roles, “User” and “Editor”, which are declared as follows:

```
1 <role
2   id="book.messageboard.User"
3   title="Message Board User"
4   description="Users that actually use the Message Board."/>
5
6 <role
7   id="book.messageboard.Editor"
8   title="Message Board Editor"
9   description="The Editor can edit and delete Messages."/>
```

Equivalently to the `zope:permission` directive, the `zope:role` directive creates and registers a new role with the global role registry. Again, the `id` must be a unique identifier that is used throughout the configuration process to identify the role. Both, the `id` and the `title` are required.

Next we grant the new permissions to the new roles, i.e. create a permission-role map. The user should be only to add and view messages, while the editor is allowed to execute all permission.

```
1 <grant
2   permission="book.messageboard.View"
3   role="book.messageboard.User"
4   />
5 <grant
6   permission="book.messageboard.Add"
7   role="book.messageboard.User"
8   />
9 <grant
10  permission="book.messageboard.Edit"
11  role="book.messageboard.Editor"
12  />
13 <grant
14  permission="book.messageboard.Delete"
15  role="book.messageboard.Editor"
16  />
```

The `zope:grant` directive is fairly complex, since it permits all three different types of security mappings. It allows you to assign a permission to a principal, a role to a principal, and a permission to a role. Therefore the directive has three optional arguments: `permission`, `role`, and `principal`. Exactly two of the three arguments have to be specified to make it a valid directive. All three security objects are specified by their `id`.

Finally, you have to include the `security.zcml` file into your other configuration. This is simply done by adding the following inclusion directive in the `ZOPE3/principals.zcml` file:

```
1 <include package="book.messageboard" file="security.zcml" />
```

The reason we put it here is to make it obvious that this file depends on the security policy. Also, when assigning permissions to roles we want all possible permissions the system can have to be defined. Since the `principals.zcml` file is the last ZCML to be evaluated, this is the best place to put the declarations.

## 16.4 Step IV: Assigning Roles to Principals

To make our package work again, we now have to connect the roles to some principals. We are going to create two new principals called `boarduser` and `boardeditor`. To do that, go to the Zope 3 root directory and add the following lines to `principals.zcml`:

```
1 <principal
2     id="book.messageboard.boarduser"
3     title="Message Board User"
4     login="boarduser" password="book"
5 />
6 <grant
7     role="book.messageboard.User"
8     principal="book.messageboard.boarduser"
9 />
10
11 <principal
12     id="book.messageboard.boardeditor"
13     title="Message Board Editor"
14     login="boardeditor" password="book"
15 />
16 <grant
17     role="book.messageboard.User"
18     principal="book.messageboard.boardeditor"
19 />
20 <grant
21     role="book.messageboard.Editor"
22     principal="book.messageboard.boardeditor"
23 />
```

The `zope:principal` directive creates and registers a new principal/user in the system. Like for all security object directives, the `id` and `title` attributes are required. We could also specify a `description` as well. In addition to these three attributes, the developer *must* specify a login and password (plain text) for the user, which is used for authentication of course.

Note that you might want to grant the `book.messageboard.User` role to the `zope.anybody` principal, so that everyone can view and add messages.

The `zope.anybody` principal is an unauthenticated principal, which is defined using the `zope:unauthenticatedPrincipal` directive, which has the same three basic attributes the `zope:principal` directive had, but does not accept the `login` and `password` attribute.

Now your system should be secure and usable. If you restart Zope 3 now, you will see that only the message board's `Editor` can freely manipulate objects. (Of course you have to log in as one.)

### **Exercises**

1. In retrospect it was a bad idea to give the `book.messageboard.User` role the `book.messageboard.View` and `book.messageboard.Add` permission, since you cannot differentiate between an anonymous user reading the board and a user being able to add messages anymore. Add yet another role called `book.messageboard.Viewer` and make the `details.html` and `thread.html` view available to this role. Then grant this role to the unauthenticated principal (`anybody`) and verify that this principal can indeed access these views.
2. (Referring to the previous problem) On the other hand, instead of creating another role, we could just grant the `View` permission to the anonymous principal directly. Do that and ensure that the unauthenticated principal can see these views.