Implementing Security

This chapter presents some typical security implementations in Servoy applications.

In This Chapter

Basic Servoy Authentication

Practical where the security will be managed by the developer of the application.

Steps

- Using the **User/Group Editor**, decide what authorization permission groups are required for the application. Create these groups before development begins.
- During development of tables and forms, use the settings on the **Security** tab of each editor to configure what permissions each group has for the table/form. During form development, keep in mind that elements must be named in order to assign permissions.
- Check the `mustAuthenticate` flag of the solution.
- Create users with either the **User/Group Editor** or with the **Users** page in the Application Server administration utility. Assign users to groups in either editor.
- Deploy the application. When launching the application, the default login window will appear to authenticate the user.

Custom Authentication

To allow tenant administrators or 'super users' to administrate users, or when using an external authentication source, custom authentication is required.

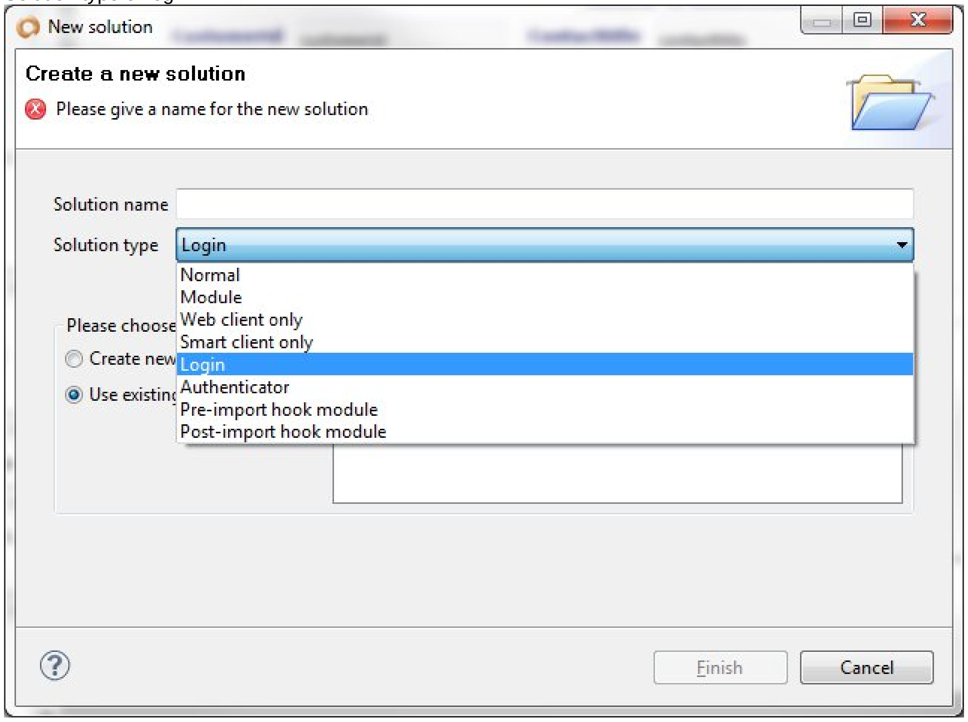
Custom authentication allows the developer to use a users table in the database for authentication or access an external authentication directory.

To use external authentication, most likely a plugin is required.

Steps

- If using Servoy for storing user data, create a user table for the application or set of applications. Make sure to at least include a username and password column in the table.
- Create a login solution. A login solution should have the following attributes:

- Solution type of login



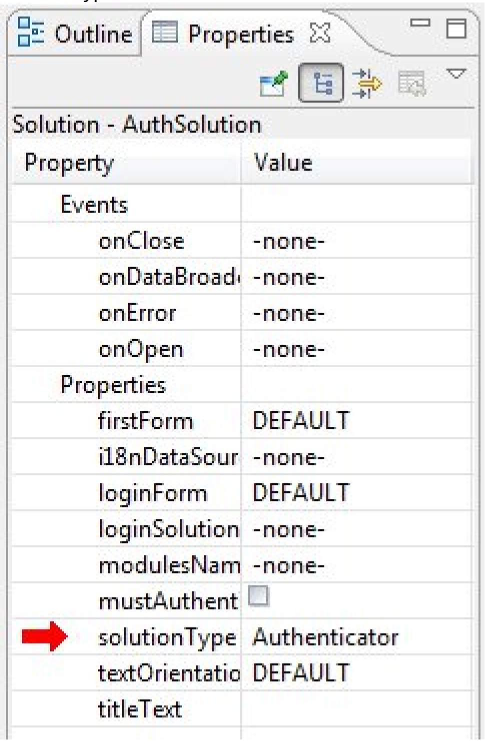
Solution name:

Solution type: Login

Normal
Module
Web client only
Smart client only
Login
Authenticator
Pre-import hook module
Post-import hook module

- A login form set as the first form
- The login form should have fields (normally form variables) for entering a username and password
- A button or method that will validate the fields are not null and finally call the authentication method of the authenticator solution. See Login Method Example.
- The login solution has no database access, so if there is any validation done during the login process (find tenant id, verify user exists, etc.), the login solution must call an authenticator solution for any database information.

- Create an authenticator solution (during the solution creation process, same as login solution). The authenticator has the following attributes:
  - Solution type of authenticator



Solution - AuthSolution

Property	Value
Events	
onClose	-none-
onDataBroad	-none-
onError	-none-
onOpen	-none-
Properties	
firstForm	DEFAULT
i18nDataSour	-none-
loginForm	DEFAULT
loginSolution	-none-
modulesNam	-none-
mustAuthent	☐
solutionType	Authenticator
textOrientatio	DEFAULT
titleText	

  - Includes a global method to authenticate users. This method will check the users table or call external authentication to authenticate. This method will also log the user into Servoy. See Authentication Method Example.
  - Can also include other methods for checking a user or getting a tenant id.
  - The authenticator solution is not a module of the solution, but must be in the repository of the Application Server or in the developer's workspace. It can be set to a module of the solution to make it easier to export and import to the Application Server, but it will never run on the client. It will always stay on the Server.
- Set the login solution property to the login solution.
- Deploy the application, making sure to include the authenticator solution an import is done on the Application Server. The custom login form will appear when launching the solution.

Enhanced Security

When first introduced in Servoy, this method of using Login and Authenticator solutions was referred to as Enhanced Security. When looking for any references to this method in other resources (such as the Servoy Forum), try searching for 'Enhanced Security'.

Solution onOpen Method

For almost every implementation of security, the solution should have an onOpen method assigned. This event is triggered right after the authentication process is complete. Some of the functions of this method in regards to security include:

- Setting up of tenancy for a multi-tenant solution. Table filters are enacted at this time to filter the data appropriately for the tenant.
- Applying custom security. If the security model is custom, these permissions should be set at this time. This would normally involve reading metadata from the database and applying permissions/restrictions based on the user's group.
- Setting security variables. Normally, in order to better facilitate using information from the users and tenants tables, global variables should be set for the current user id and the current tenant id. If these variables were not set during the login process, then they should be set here.
- Database switching. If the application is set up as a multi-tenant with a database for each tenant, this is the time to switch to the tenant's database.

Login Method Example

The code below is an example of a typical custom login method. In this scenario, the login page contains the following form variables:

- userName - the name entered at login
- password - the password entered at login
- errorMessage - any error messages returned to the login solution.

The tenant is verified during login before actual authentication occurs.

```
function login(){

        errorMessage = null;

        if(!userName){
                errorMessage = 'Please specify a user name';
                return false;
        }
        if(!password){
                errorMessage = 'Please specify a password';
                return false;
        }

        var tenantID = security.authenticate("myAuthenticator","getTenant",[userName]);
        if(tenantID){
                if(security.authenticate("myAuthenticator","loginUser",[userName,password])){
                        return true;
                } else {
                        errorMessage = "No tenant found. Please check your password";
                }
        }
        errorMessage = 'Login Failed';
}
```

Authentication Method Example

The code below is an example of a typical authentication method.

```
function loginUser(user, password) {
        if (!(user && password)) {
                application.output('Unexpected credentials received', LOGGINGLEVEL.DEBUG);
                return false;
        }
        var authenticated = ... //either query database or use LDAP

        if (authenticated) {
                var ok = security.login(user, user, ['group']) // Assume a group for each department
                application.output('User ' + user + ' authenticated: ' + ok, LOGGINGLEVEL.DEBUG);
                return ok;
        }
        application.output('User ' + user + ' could not be authenticated', LOGGINGLEVEL.DEBUG);
        // Sleep for 3 seconds to discourage brute force attacks
        application.sleep(3000);
        return false;
}
```

When authentication fails, adding a pauze can be useful to discourage brute force password attacks.

Note that there is the option to query the database or get an external authentication. A user groups table might be read to create the array of groups the user has privileges with. Also note that the only thing that is returned is a true or false and that reporting errors to the user does not occur at the authentication level.

Fully Custom Security

In a fully custom security implementation, both authentication and authorization information is handled outside of Servoy built in security paradigm.

Steps

- Create a custom authentication implementation as indicated above.
- Create table(s) in the database to store information for permissions and restrictions in the application.
- In the onOpen method of the solution, create code to read the metadata and apply permissions/restrictions for the current user.

The developer writes code to assign form and table security to each part of the solution. This style of solution is normally managed by exclusion: instead of assigning permission to every part, just assign permissions to the part you want controlled. An example of fully custom security can be found on the ServoyForge site in the Security framework.