Upgrade existing implementations to Servoy 5.2's Enhanced Security

About this tutorial

Servoy 5.2 onwards provides an enhanced security mechanism, separating the logic surrounding the login process from the actual authentication logic, to provide more robust authentication.

This tutorial provides the information how to adopt existing solutions to the Enhanced Security that Servoy 5.2 (and onwards) offers.

About Enhanced Security

Enhanced Security is a Application Server wide setting that changes the way that the Servoy Smart Client authenticates with the Servoy Application Server.

By default Enhanced Security is enabled and existing solutions that fall in either one of the two scenario's below are affected by it:

1. Solutions running in the Servoy Smart Client that **do not require authentication**
2. Solutions running in the Servoy Smart Client that **utilize the custom login form** functionality

In these two scenario's modification of existing solutions is required to adopt the enhanced security mechanism in Servoy 5.2 and onwards. All other deployment scenario's are not affected by Enhanced Security, but when a solution is modified to adopt Enhanced Security, they do work with it.

It is possible to disable Enhanced Security, but this is strongly discouraged.

When Enhanced Security is enabled, the following changes:

- Solutions do not have a loginForm, but a loginSolution, the loginForm property is obsolete.
- All Smart Client must authenticate, overruling the mustAuthenticate property on solutions

Enabling/disabling Enhanced Security

Although it's highly recommended to utilize the Enhanced Security from Servoy 5.2 onwards, it is possible to disable it, both in Servoy Developer, as well as on the Servoy Application Server. When disabled, the login & authentication behavior is unchanged from Servoy versions before 5.2.

Servoy Developer: A new preference has been added under Window > Preferences > Servoy > Developer, called "Run Servoy Application Server with Enhanced Security"

Servoy Application Server: A new setting has been added on the Servoy Admin page, under "Servoy Server Home" > "Admin settings", called "servoy.application_server.enhancedSecurity"

Scenario 1: Solutions in the Servoy Smart Client that do not require authentication

With Enhanced Security enabled, all solutions that run in the Servoy Smart Client require authentication. The mustAuthenticate property on solution level is ignored.

Possible options are:

- Running the solution in the Servoy Web Client only
- Add login functionality to the Solution
- Disabling Enhanced Security (strongly discouraged)

Scenario 2: Solutions in the Servoy Smart Client that utilize the custom login form functionality

With Enhanced Security enabled, solutions running in the Servoy Smart Client that utilize a custom login form (the loginForm property on solution level is set) will require modification to continue to operate the way the user is accustomed to.

The custom login form needs to be split off into a separate solution of type "Login", while the actual authentication logic needs to be split off into a separate solution of type "Authenticator".

The Login solution will retrieve the user credentials and call a global method in the Authenticator solution, which in turn will call the function security.login(). When the call to security.login() was successful, the main solution will load in the client.

Authenticator solution

Servoy 5.2 introduces the type "Authenticator" for solutions. Solutions of type "Authenticator" are used by solutions of type "Login" to retrieve data and to authenticate the Client with the Servoy Application Server.

An Authenticator solution is called from the a Login solution using the function:

```
security.authenticate(authenticatorSolutionName, globalMethodName, argumentsArray);
```

The Authenticator solution **exposes it's functionality** through one or more global methods to the Login solution. Any global method that is defined in the Authenticator solution can be called with the security.authenticate method from within the Login solution. The return value of the global method that is called in the Authenticator solution is also returned by the security.authenticate call in the Login solution. The types that are supported for the return value are the standard JavaScript types (String, Number, boolean, Array etc.) and JSDataSet.

Authenticator solutions are **stateless**, meaning that it will not remember the value of variables in between subsequent calls.

As the Authenticator solution is only **used by Login solution to authenticate the Client** and **optionally to retrieve data** for the login process, both by calling global methods, the Authenticator solution does not require any forms.

The Authenticator solution does not have to be part of the main solution, it just needs to be available on the Application Server. It can be included into the main solution as a module for easier deployment though.

Login solution

Servoy 5.2 introduces the type "Login" for solutions. Solutions of type "Login" can be used in solutions of type "Normal", "Smart Client only" and "Web Client only" to provide the login logic at solution start.

Login solutions can contain Forms and business logic, but have a limited scripting API:

- Direct access to data from any Database Server is restricted, which means for example that Forms cannot be based on datasources.
- The function databaseManager.getDataSetByQuery() does not work.
- The RawSQL plugin will not work.
- The entire security node does not work, with the exception of security.authenticate().

The Login solution can however request data required for the login process through an Authenticator solution using the function:

```
security.authenticate(authenticatorSolutionName, globalMethodName, argumentsArray);
```

This function allows the Login solution to call a global method with optional arguments in the specified Authenticator solution. The return value of the global method is also returned by the security.authenticate call in the Login solution. The types that are supported for the return value are the standard JavaScript types (String, Number, boolean, Array etc.) and JSDataSet.

As direct access to data from the datasources is not allowed, the forms in the Login solution cannot be based on a datasource (set datasource to -none-)

The Login solution is to be included as a module into the solution that requires authentication. This can be done by just adding the Login solution as a module, or by setting the loginSolutionName property on solution level of the solution that requires authentication. The Login solution that is used for a solution is the first module (alphabetically) of type Login" that resides directly under the main solution.

When the Login solution has made a call to the Authenticator solution and in the method executed in the Authenticator solution the Client was authenticated (security.login(...) was called successfully), upon returning of the call to the authenticator solution, the main solution will be loaded. Any values set in (global) variables set during the login process will remain available once the main solution is loaded and the user will be redirected to the first form of the main solution.

New Setup

Instead of defining a login form for a solution, a login solution is set on the solution.

The login solution should contain only the user interface to present the user with a credentials form (typically username/password).

When the user enters its credentials, the login solution will submit these to a authenticator solution (which is running inside the Servoy Application Server).

The authenticator solution will validate the credentials and, if successful, mark the user as authenticated with the server, by calling the function security. login(...).

When the client is successfully authenticated, the regular solution will be loaded in the client and activated.

Converting a login form scenario to Enhanced Security

- First, create 2 new solutions, one of solutionType Login and one of solutionType Authenticator
- Include the Login solution as a module into the main solution that is to be converted to Enhanced Security. This will automatically set the loginSolutionName property on solution level of the main solution.
- Move the custom login form from the main solution into the Login solution (right-click the Form in the Solution Explorer to access the Move form functionality). When the login logic consists of multiple forms, move all the forms required for the login process to the Login solution. In this scenario, set the firstForm property in the Login solution to the first form the user needs to see in the login process.
- Move (or duplicate) any other solution object (media's, valuelists, global methods etc.) required in the login process to the Login solution.

- Set the loginForm property of the main solution to "DEFAULT" to cleanup this property, as it has no use anymore (rightclick the propertyname in the property editor for the "Restore Default Value" option)
- Split off the actual authentication logic into the Authentication module as a global method
- Make sure the login process does not depend on datasources or utilizes scripting API functions that are not available in the Login solution. When data is required in the login process, setup global methods in the Authenticator solution to provide the data to the Login solution

Example

Existing code

This example shows how to convert a solution that uses its own logic to validate user credentials (for instance checking with an LDAP server). The login form prompts the user for a username, password and department. The departments are retrieved from a value list that contains all records departments table and are shown in a dropdown.

```
var userName = ''; //Will be specified by user

var departmentName = ''; //Will be specified by user

function login(event) {
  var authenticated = ... //  contact LDAP server
  if (authenticated) {
    var ok =  security.login(userName, userName, [departmentName]) // Assume a group for each department
    application.output('User ' + userName + ' authenticated: ' + ok, LOGGINGLEVEL.DEBUG);
  } else {
    application.output('User 'userName' could not be authenticated', LOGGINGLEVEL.DEBUG);
  }
}
```

New solution

A new solution 'Login' (solutionType Login) is created, the login form from the original solution is moved to the Login solution, the form has no datasource attached (datasource is set to -none-). A custom value list is created in the new solution called departments (no entries, will be filled in the login form)
The onShow event of the form retrieves required data via the authenticator.

```
function onShow(firstShow, event) {
  if (firstShow) {
    var departments = security.authenticate('Authenticator','getDepartments');
    application.setValueListItems('departments', departments)
  }
}
```

The login method attached to the login button in the login solution is altered to also call the authenticator:

```
function login(event) {
  security.authenticate('Authenticator', 'login', [userName, passWord, departmentName]);
}
```

Another solution 'Authenticator' (solutionType Authenticator) is created. The departments value list from the original solution is copied to the Authenticator solution.
The solution will not have any forms, only global methods for fetching of the required data and the actual authentication logic.

```
function getDepartments() {
  return application.getValueListItems('departments')
}

function login(user, password, department) {
  if (!(user && password && department)) {
    application.output('Unexpected credentials received', LOGGINGLEVEL.DEBUG);
    return false;
  }
  var authenticated = ... // contact LDAP server
  if (authenticated) {
    var ok = security.login(object.user, object.user, object.dep) // 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);
  return false;
}
```

Authentication using Servoy's built-in Security

When Servoy's built-in Security mechanism (users & groups) is used, there is a short-cut, that can be taken, that eliminates the need for a Authenticator solution.

When the authentication code is something along the following lines:

```
function login(userName, password) {
   var userUID = security.getUserUID(userName);
   if (security.checkPassword(userUID, password)) {
      return security.login(userName, userUID, security.getUserGroups(userUID))
   }
   return false;
}
```

The shortcut that can be taken in the use of null Authenticator in the Login solution:

```
security.authenticate(null, null, [userName, password]);
```

With this code, Servoy will check the combination of userName and password against the built-in Security of Servoy and if the user is known and the password is correct, the authentication of the Client will be successful and the user will be logged in with the Groups he/she belongs to.

In this scenario there is no need for a Authenticator solution.

Using the Authenticator solution to offload the process of retrieval of information

The Authenticator solution runs on the Servoy Application Server and can provide data to the Login solution by exposing global methods. Any (global) variable set in the Login solution remains available when the main solution gets loaded after successful authentication.

These features can be utilized by solutions that run in the Smart Client to retrieve data from datasources, where many individual SQL statements are required or a lot of data needs to be processed by some business logic to get the required information, for example with solution metadata.

By offloading the process of retrieval of the information to the server, fewer round-trips are required between the client and the server, resulting in better performance. An example of this could be the retrieval of solution metadata, for example a navigation structure or security settings.

The security.authenticate(...) function can be utilized as long as there is no logged in user.

Examples

From the Servoy Example SVN a simple demo solution demonstrating the Enhanced Security mechanism can be checked out:

- servoyEnhancedSecurityDemo: main solution
- servoyEnhancedSecurityDemoLogin: login solution
- servoyEnhancedSecurityDemoAuthenticator: Authenticator solution
- servoyEnhancedSecurityDemoResources: The resources project for this set of demo solutions