Security Checklist for Implementation of Resource Controllers

Overview

This page outlines how to implement secure backend resources. It is recommended to go through the following checklist for every application domain. The resource controller for classes, CL_OO_ADT_RC_CLASS, is used here as an example.

Checklist

1. **Check which URI patterns are registered in the application class**
   A request from the client is delegated to the responsible resource controller class via the so-called *resource application* class. (In other words: a resource application is a domain-specific router for requests.) For a given static URI path, you can find the respective resource application class by considering the implementations (and their filter values) of BAdI BADI_ADT_REST_RFC_APPLICATION in Enhancement Spot SADT_REST_RFC_APPLICATION.
   In method REGISTER_RESOURCES of the resource application class, the individual resource controller ('handler') classes are registered for different URI patterns ('templates').
   Example:

```
registry->register_resource( template = '/oo/classes/{classname}'
handler_class = 'CL_OO_ADT_RC_CLASS' ).
registry->register_resource( template = '/oo/classes/{classname}/source/{includename}'
handler_class = 'CL_OO_ADT_RC_CLASS' ).
```

   - Check that the registered URI patterns match the filter values of the BAdI implementation
2. **In any resource controller class: Provide exception handling for methods which are not implemented**
   - Ensure that there is no unimplemented method in the resource controller classes. If a method is not productively used, raise an exception of type CX_ADT_RES_METH_NOT_SUPPORTED. While this exception is integratively thrown in the resource controller basis classes (usually CL_REST_RESOURCE or subclasses of CL_REST_RESOURCE; sometimes the outdated CL_ADT_RESOURCE_CONTROLLER), you have to care for the integrity of redefined methods in derived resource controller classes.
3. **For all implemented methods in the resource controllers, check whether all URI patterns (see 1) are handled correctly**
   Some methods (e.g. CL_OO_ADT_RC_CLASS->IF_ADT_RESOURCE_CONTROLLER~CREATE) may be implemented only for one of the registered URI patterns. (Example: the create method for the class resource controller can only deal with meta data (URI /oo/classes/*classname)* but it cannot deal with source code.)
   - Make sure that the method implementation checks the URI pattern correctly and raises an exception CX_ADT_RES_METH_NOT_SUPPORTED if the method is not allowed for the specific case
     (Example: source code URI /oo/classes/*classname*/source/*includename* processed in the create method.)
4. **Check the validation of query parameters**
   - There must be *no undocumented query parameters*, especially no back-doors ('security by obscurity') to process special requests without a specific authorization check.
   - All query parameters shall be validated.
     - Non-existing query-parameters can be ignored.
     - Check for obligatory parameters. If a parameter is obligatory but missing, raise an exception CX_ADT_REST_PARAMETER_NOT_FND. This happens automatically if you use CX_ADT_REST_PARAMETER_NOT_FND->GET_URI_QUERY_PARAMETER.
     - Check the uniqueness of parameter values.
     - Check whether parameter values have the right technical type (e.g. integer or string, greater than zero, and so on). If a parameter value is not consistent with the expected format, raise an exception type CX_ADT_REST_PARAM_VALUE_INVLD. Basic checks for the technical type are also implemented in CX_ADT_REST_PARAMETER_NOT_FND->GET_URI_QUERY_PARAMETER.
   - Do not allow any kind of third path processing if a parameter value is incorrect. Example: If the parameter must be set to TRUE or FALSE, any kind of other value should either lead to an exception CX_ADT_REST_PARAM_VALUE_INVLD (preferable) or at least be interpreted either as TRUE or FALSE.
5. **Implement authorization checks for all methods**
   - Every single activity that can be executed (i.e., every method for every valid URI pattern) needs to be authorized.
   - As for ABAP Workbench activities, function module RS_ACCESS_PERMISSION shall be used to check access permission in every single method. As this is the central function module which also checks for system changeability option, only use this function module for authority check of development objects such as functions, programs, classes, tables, etc.
6. **Implement automated unit tests for authority checks**
   - Create a test package <*domain*>*_ADT_TEST_HOME* below package SADT_HOME, e.g. SEO_ADT_TEST_HOME
   - Create a test program <*domain*>*_ADT_TEST_AUTH_CHECK_*<resource> for every single resource controller, e.g. SEO_ADT_TEST_AUTH_CHECK_CLAS, in the new test package.

- Implement test classes in the program as described in How to Implement Security Tests for Resource Controllers.

7. **Maintain authorization proposals for authority object S_ADT_RES**
   As implemented in the ADT REST framework, the remote access to the backend resources of ABAP Development Tools is fundamentally controlled by authority object S_ADT_RES. A resource is only accessible when the authority object is maintained accordingly in the user profile (cf. internal SAP note 1657744). To support the maintenance of the user profiles resp. user roles, ensure that any relevant URI pattern is given as a default authorization value:
   - Start transaction SU22.
   - On the selection screen, choose "RFC Function Module" in the drop-down box "Type of Application". Enter the name "SADT_REST_RFC_ENDPOINT" in the field "Function Module". Press F8.
   - Check the authorization proposals in general. The 'Proposal Status' for authorization object S_ADT_RES must be set to 'YS' (yes), while the Proposal Status for all other objects typically should be set to 'NO'.
   - The authority object S_ADT_RES represents a white list of all allowed static URI prefixes. Verify whether the authorization proposals ('Default Authorization Values') for object S_ADT_RES for the field URI and the resource controller are maintained. Use F4 help to get the values for the URI patterns. The URI patterns should be equal to all filter values registered in the registration BAdI BADI_ADT_REST_RFC_APPLICATION). (Hint: In function group SADT_REST there is a unit test which checks the completeness and consistency of the default authorization values.)