OFBiz Security Permissions

Security, as discussed in this article, is about permissions being granted to access OFBiz artifacts such as services, screens and data.

How permissions are organised (the 2 levels of security)

The larger domain of security is split into 2 categories in OFBiz:

1. application/functionality level security
2. data level security

Category 1 (user permission, eg ORDERMGR_CREATE**)** is **UserLogin dependent** and doesn't know about anything except the UserLogin, the permissions checked for different screens, services, etc, and the SecurityGroup structure that maps between them.
Category 2 (party permission, eg ORDERMGR_ROLE_CREATE) is **Party dependent** and can be combined with Category 1, usually with special "**role limited** " permissions that require not just the permission, but some **relationship** between the Party and whatever artifact are being controlled. This supports quite a wide range of access control.

It should be possible (in theory, not tested) to use Category 1 security without the party component. However, Category 2 is very dependent on the Party data object and the data object related to it.

This is the general design. What exists OOTB in OFBiz has various examples of both, but no attempt has been made to create a comprehensive or at least "generically complete" set of security settings in either style #1 or #2. Though some effort has been done to add more granular permissions in Category 1 (like ORDERMGR_SALES_CREATE, see above and below for details), this is an area of customization that tends to be pretty complex and there is possibly as much variety in it as there are combinations of relationships of entities in OFBiz.

If you need to know more details look at Security Administration

Wrong use of "role limited" permissions

Note that a "role limited" permissions to work **requires** an enforcing relationship to be used. There are currently a number of cases in OFBiz where "role limited" permissions are used without enforcing relationships. For instance often the <alt-permission tag is used in simple methods as if the "role limited" permission would complement the primary user permission when it's actually only redundant since the <check-permission tag is not supposed to infer by itself what to do with this alternative permission. This is explained and discussed in this dev ML thread. In this other user ML thread a solution is proposed to prevent this to spread all over the code. Two other propositions has been discussed some years ago, notably this OFBiz Security Redesign.

There was also this discussion about NIST and RBAC (Role Based Access Control) started by Ruth. But has Adrian's remarked (many times) we have always this technical discrepancy between users (userLogin) and parties (Party). In OFBiz an user and a party have different meanings (a party may use several different userLogins).

Also note though that there are cases where the alternative permission could be used, but not with "role limited" permissions, only user permissions. For instance someone may not have the right to create all types of order but only sales orders. Then with the primary permission defined ORDERMGR_CREATE this person can't create a sales order but with the alternative permission defined as ORDERMGR_SALES_CREATE it's possible. As you see access permissions can be a very complex topic!

The following is a technical description explaining security and permissions in OFBiz.

How to use permissions

Category 1 is based on **SecurityPermissions** records. The names of permissions consist out of two parts separated with a '_' character. Most of the time, the first part is the application name and the second part is the action or operation. The application name is normally the same as the component name (or webapp name in case of sub-component webapp) written in uppercase characters. The action can be anything. However VIEW, CREATE, UPDATE, DELETE (related to CRUD functions) and ADMIN are the most commonly used. ADMIN is a special action which allows all operations.

For example, the security permission for creating an order, is ORDERMGR_CREATE. The application is the Order Manager (ORDERMGR) and the action is 'CREATE' which indicates that this permission controls the creation of an new order. OOTB, this permission is granted to the 'ORDERENTRY' **SecurityGroup** entity, so every user ID (userLoginId) which is part of the ORDERENTRY group can create orders.

We currently have several systems/levels where this is implemented:

At login level

The **base-permission** defined in the ofbiz-component.xml file as an attribute of the <webapp element controls the access to component. A "base-permission" definition might contains several permissions constituents.

To login an user must have at least the permission COMPONENT-NAME_VIEW or COMPONENT-NAME_ADMIN permission. Most of the components have the values **OFBTOOLS** coupled with generally a COMPONENT-NAME permission. This means that **both** OFBTOOLS and (comma is used as an **ANI** operator in the definition of base-permissions) COMPONENT-NAME_VIEW (or COMPONENT-NAME_ADMIN permission) are required to get access to the web app. The OFBTOOLS is needed to allows permission to every web application configuration. Note also that the NONE base-permission allows anyone without any specific permission to access to the component. Also, due to how permissions constituents are parsed, you can't have underscores in the COMPONENT-NAME.

The base-permission, which is still widely used in OFBiz OOTB, has been deprecated. A new **access-permission** attribute of <webapp element has been introduced after this exchange but has not yet replaced base-permission (there are none occurrences of access-permission in OOTB trunk code)

Note (to be reviewed/discussed)

After Adrian's 1s post in the thread above, I checked if OFBTOOLS is really strictly needed to gain access to a component. This is not the case for a number of components (take assemaint for instance) which can be at least accessed by the admin. But this needs to be clarified (only the admin?) and it seems we should no longer wait to replace base-permission by the clearer access-permission in trunk. I don't know why Adrian Crum did not continue, maybe only lack of time?

At the component menu level

It's pretty simple to use. The component top-level menu will only show the components to which a logged-on user has at least the WEBAPP-NAME_VIEW or COMPONENT-NAME_ADMIN permissions, i.e. the same as on the login level. This is done in appbar.ftl which defines which tabs are visible in the applications tabs bar.

At the request (controller.xml) level

Though not related with permissions, there are two important parameters here. Every request (<request-map) tag has a security (<security) tag with the 2 parameters: 'https' to define if SSL will be used for this request and the 'auth' parameter to define if a user needs to be logged on to execute this request. If the user is not logged on the login screen will be shown and after a successful login AND if the security checking on the other levels are successful, the request will be executed.

At screen level

In the '<section' part of a screen definition and '<condition' tag the test with the '<if-has-permission' tag with the 'permission' and 'action' test can be performed. Depending on the outcome the '<widgets>' or '<fail-widgets>' can be shown.
You may also use <if-service-permission.

At Freemarker template snippet level

The session variable *security* is always present in the screen context. You can use it to request security informations using Security class public methods: hasPermission, hasEntityPermission, hasRolePermission.

At service definition level

This functionnality allows the (re-)usage of services in other components with a different security schema. An example is the project manager component which uses many services of the workeffort component but has a different security schema.
In the service definition, (files in the servicedef directory) the '<permission-service' can be inserted pointing to a service which will check if the service is allowed to be executed. The input to this service is at least the 'main-action' parameter to specify the action (CREATE, UPDATE etc) but also any other input parameter such as a productId, or workEffortId as defined is the security service.
You can extend the base permission service using ECA rules on the permission service used by the service you want to reuse. Just have it run your permission service after the main one IFF the main one results in an error (failed permission check), and make sure your ECA rule has it put its results in the context if your security scenario succeeds.
A good basic implementation example is exampleGenericPermission in the Example component

At service implementation level.

- Minilanguage.
  Using the *check-permission* and *if-has-permission* tags to check the access.

- Java and Groovy.
  Using the org.ofbiz.security.Security API methods such as *hasEntityPermission.*

At record level, by using Role limited permissions or related means

The purpose of role-limited permissions is to tie a SecurityPermission to record level security using the RoleType/PartyRole and related entities. In OFBiz this is how record level permissions are done, i.e. somehow the user (through her Party record) is associated with another record in the database and that specific relationship must exist in order for the role-limited permission to take effect.

A good example is in the secondary hasPermission method in OrderServices class or how ProductStoreRole, ContentAndRole, PartyRole, entities are used in Java code (and at large *ENTITY-NAME*Role entities).

Note how the main OrderService.hasPermission() method uses the OrderRole entity without explicit role-limited permission. See also checkStoreCustomerRole in ProductEvents class for the same with the ProductStoreRole entity.

A view entity can also be used to define a path from the Party in question to the target/desired entity through relationships. This is usually do-able and easy to do with a single view entity, and if a query on that entity with the proper constraints returns any results then you know the user/party has the permission. See for instance the ProductCategoryMemberAndRole view entity in the checkProductRelatedPermission (in the ProductServices.xml file) and how are the catalog role limited permissions defined and used for an example.

You can imagine your own ways in function of your requirements...

Protected views

This is another mechanism which is used to protect from data leakage (data stolen from a login/password couple compromised). It works like the well known grey list anti-spam feature (aka tarpitting). It's pretty simple to use.
You define the views you want to protect under a Security Group using the Protected Views menu.
There you define the

- View name using uri attribute of request-map element (controller.xml file)
- Maximum number of visits (per period)
- Duration during which the visits are considered (in seconds)
- Duration during which the view will not be accessible (in seconds). This parameter defines the tarpitting period.

You don't have to define anything else for the protected views mechanism to work. If you don't define a "protect" response, in case of blocked screen, it will be rendered blank by default.

- But you may define a generic (per instance) "protect" response view which will be rendered by default using the default.error.response.view parameter in security.properties.
- Instead of only per request or per instance (setting in security.properties), you may also define a "protect" view by application using a <protect view="name_of_view"/> element in controller.xml. If the "protect" response is not found, first it will check for an application default before falling back to per instance.

The code use a simple strategy as we don't need something very sophisticated to trap malignants. It's all in ProtectViewWorker.java.

Data model

All these levels rely on several database tables:

1. **SecurityPermission**: the lowest level where all basic permissions are defined. Typically used to set up the standard permissions (_ADMIN, _CREATE and _DELETE). If there is a need, the same set of permissions may be created with a "_ROLE" in there. Examples are: CATALOG_ADMIN, CATALOG_CREATE, CATALOG_DELETE, CATALOG_ROLE_ADMIN, CATALOG_ROLE_CREATE.
2. **SecurityGroup**: where the permission group is defined. For setting up groups that typically have blanket permissions. FULLADMIN and PARTYADMIN are examples.
3. **SecurityGroupPermission**: permissions are grouped together in this table. Associates SecurityPermission with SecurityGroup.
4. **UserLoginSecurityGroup**: Which userLogin id's have access to which permission groups. This is where parties can be associated with SecurityGroup. Note: Parties are associated with SecurityGroup by their userLoginId, but when adding a party to an entity role entity, the partyId is used.
5. **PartyRelationship**: security groups can be set for certain party relationships.
6. ***ENTITY-NAME*Role** (OrderRole, PartyRole, etc.):
   The org.ofbiz.core.security.OFBizSecurity class contains methods for applying/using these tables.
   Here is some stuff that David wrote in an email that shows 2 cases where group permissions are used:
   a. ~~has CONTENTMGR_VIEW (or CONTENTMGR_ADMIN, handled by hasEntityPermission method)~~
   b. ~~has CONTENTMGR_ROLE_VIEW (or CONTENTMGR_ROLE_ADMIN) and partyId of current userLogin is associated with the Content in the ContentRole table (in this case with any roleTypeId) (see Content management security)~~
   c. But finally In an exchange with Anil, David did not recommend to use these "old" Security class public methods (hasPermission, hasEntityPermission, hasRolePermission) but to rather use the "new" pattern used in the

- ProductServices.xml#checkProductRelatedPermission method:

checkProductRelatedPermission

```
    <!-- a method to centralize product security code, meant to be called in-line with
        call-simple-method, and the checkAction and callingMethodName attributes should be in the method
context -->
    <simple-method method-name="checkProductRelatedPermission" short-description="Check Product Related
Permission">
        <if-empty field="callingMethodName">
            <set value="this operation" field="callingMethodName"/>
        </if-empty>
        <if-empty field="checkAction">
            <set value="UPDATE" field="checkAction"/>
        </if-empty>
        <!-- find all role-categories that this product is a member of -->
        <if>
            <condition>
                <not><if-has-permission permission="CATALOG" action="_${checkAction}"/></not>
            </condition>
            <then>
                <set from-field="parameters.productId" field="lookupRoleCategoriesMap.productId"/>
                <set from-field="userLogin.partyId" field="lookupRoleCategoriesMap.partyId"/>
                <set value="LTD_ADMIN" field="lookupRoleCategoriesMap.roleTypeId"/>
                <find-by-and entity-name="ProductCategoryMemberAndRole" map="lookupRoleCategoriesMap"
list="roleCategories"/>
                <filter-list-by-date list="roleCategories"/>
                <filter-list-by-date list="roleCategories" from-field-name="roleFromDate" thru-field-
name="roleThruDate"/>
            </then>
        </if>
        <if>
            <condition>
                <not>
                    <or>
                        <if-has-permission permission="CATALOG" action="_${checkAction}"/>
                        <and>
                            <if-has-permission permission="CATALOG_ROLE" action="_${checkAction}"/>
                            <not><if-empty field="roleCategories"/></not>
                        </and>
                        <and>
                            <not><if-empty field="alternatePermissionRoot"/></not>
                            <if-has-permission permission="${alternatePermissionRoot}" action="
_${checkAction}"/>
                        </and>
                    </or>
                </not>
            </condition>
            <then>
                <add-error>
                    <fail-property resource="ProductUiLabels" property="
ProductCatalogCreatePermissionError"/>
                </add-error>
            </then>
        </if>
    </simple-method>
```

OFBTOOLS