Internal access control (permissions) - CustomerID

Permissions

Ubisecure CustomerID uses permissions for internal authorization and for customizing the user interfaces. For example, the self-service interface may be customized for different kind of user accounts.The self-service interface can be configured to permit certain user groups to manage their own authentication method credentials and whether methods are enabled for them, while precluding this from other user groups. The administration interface scope can also be heavily customized based on the user's permissions.

The permissions are not directly assigned to users, but through internal roles. Each internal role may contain multiple permissions and a user may have multiple internal roles for multiple different organizations. For example, the user may have two internal roles in organization A and one internal role in organization B. The total permissions in organization A is a sum of the two roles the user has in organization A and permissions for organization B are the ones assigned to the role the user has in that organization. The permissions can also be recursive, that is, the permissions are valid on the organization where the role(s) reside(s) and in all suborganizations it has.

The mapping between permissions and roles is made in the permissions.properties property file. In the default Windows installation, it is located at:

```
C:\Program Files\Ubisecure\customerid\application\custom\permissions.properties
```

The following example provides information about the permission to role mapping:

```
# permissions.properties

user.list = rel:OrganizationUser, inh:OrganizationMainUser
user.read.personal = rel:OrganizationUser, inh:OrganizationMainUser
user.read.roles = rel:OrganizationUser, inh:OrganizationMainUser
user.read.mandates = rel:OrganizationUser, inh:OrganizationMainUser
user.edit = inh:OrganizationMainUser
user.create = inh:OrganizationMainUser
user.delete = inh:OrganizationMainUser
user.approval.read = rel:OrganizationUser, inh:OrganizationMainUser
user.approval.edit = rel:OrganizationUser, inh:OrganizationMainUser
user.approval.approve = rel:OrganizationUser, inh:OrganizationMainUser
```

In the example configuration, the `OrganizationMainUser` role includes all user related permissions in the organization where the role resides and in all of its suborganizations (recursive). The `OrganizationUser` role includes permissions to read and review user information and to approve pending user applications in the organization where the role resides and nowhere else.

The keywords for the `permissions.properties` file are described in the following table.

KEYWORD	DESCRIPTION
`abs:`	The absolute path to the target role in the `eIDM Users` branch. For example: `abs:eIDM/eIDMMainUser`
`rel:`	Permissions are only valid in the organization where the role resides.
`inh:`	The permissions are valid in the organization where the role resides and in all of its suborganizations.
`dirinh:`	The permissions are valid in the organization where the role resides and in all of its suborganizations, but the role must have been directly given to the user. If the user has received the role via another role then this permission is not valid.
`par:`	The permission is granted if the user has the specified role in the parent of the organization in question. If the organization in question is on the top level then the role can be from that organization as it has no parent.
`grp:`	The permissions are for a certain group in the `eIDM Groups` branch. For example, `grp:eIDMUser`
`any:`	The permissions are for users with the selected role in any organization. For example, `any:CorporateUser`
`:unless:`	This suffix is used with other keywords, the first role and another additional role. It is for permissions that are valid only if there are not users with direct role in the current organization. Note that super user permissions can not be overridden with ":unless:" definitions. For example, `abs:eIDM/eIDMMainUser:unless:OrganizationMainUser` In this example the *eIDM/eIDMMainUser* has the permission unless there is a user with direct role *OrganizationMainUser* in the current organization.

Table 1. Keywords for the *permission.properties* file

Configuration

The following table describes all supported permissions available in `permissions.properties` file.

PERMISSION	DESCRIPTION
`superuser`	This permission defines the users who have all possible permissions in the system. The users defined here have all the other permissions in any permissions category.
`user.list`	This permission defines the users who are allowed to list organizations' users in the admin service.

`user.read.personal`	This permission defines the users who are allowed to read the users' personal information in the admin service.
	
	You may also define field specific read permissions by adding the field name after `user.read.personal.`
	
	Example:
	`user.read.personal.ssn = inh:OrganizationMainUser`
	
	This example means that only someone in the `OrganizationMainUser` role can see the user's social security number in the admin service.
	
	You can define also permissions for authentication methods with:
	
	`user.read.personal.method`
	`user.read.personal.method.<authentication method technical name>`
	
	Example:
	
	`user.read.personal.method.ubikey.otp.1 = inh:OrganizationMainUser`
	
	This example means that only someone in the `OrganizationMainUser` role can see the state of the user's `ubikey.otp.1` authentication method.
	
	Field specific permissions override the general permission.

`user.read.roles`	This permission defines the users who are allowed to read the users' role information in the admin service.
`user.read.mandates`	This permission defines the users who are allowed to read the users' mandate information in the admin service.
`user.mandates.remove`	This permission defines those users who are allowed to remove mandates from organization users in the admin service.

`user.edit`	This permission defines the users who are allowed to edit the users' information in the admin service.
	
	You may also define field specific edit permissions by adding the field name after `user.edit.`
	
	Example:
	`user.edit.ssn =`
	
	This example means that nobody can edit the user's social security number in the admin service.
	
	A special case is `user.edit.accountstatus`. This permission definition gives the possibility to enable or disable a user.
	
	Also authentication methods are handled similarly to user information fields. These permissions define the users who are allowed to manage other users authentication data in the admin service. You can define permissions for authentication methods with:
	
	`user.edit.method`
	`user.edit.method.<authentication method technical name>`
	
	Example:
	
	`user.edit.method.ubikey.sms.1 = inh:OrganizationMainUser`
	
	This example means that only someone in the `OrganizationMainUser` role can alter the state of the user's `ubikey.sms.1` authentication method.
	
	Field specific permissions override the general permission.

`user.create`	This permission defines the users who are allowed to create new users in the admin service.
`user.delete`	This permission defines the users who are allowed to delete other organization users in the admin service.

`user.move`	This permission defines the users who are allowed to move other organization users in the admin service
`user.approval.read`	This permission defines the users who are allowed to read user approvals in the admin service. You may also define field specific read permissions by adding the field name after `user.approval.read.` Example: `user.approval.read.ssn = inh:OrganizationMainUser` This example means that only someone in the `OrganizationMainUser` role can see the user's social security number in the approval view. Field specific permissions override the general permission.
`user.approval.edit`	This permission defines the users who are allowed to edit user approvals in the admin service. You may also define field specific edit permissions by adding the field name after `user.approval.edit.` Example: `user.approval.edit.ssn =` This example means that nobody can edit the user's social security number in the approval view. Field specific permissions override the general permission.
`user.approval.approve`	This permission defines the users who are allowed to approve other organization users in the admin service.
`organization.read`	This permission defines the users who are allowed to read organization information in the admin service. You may also define field specific read permissions by adding the field name after `organization.read.` Example: `organization.read.technicalname = inh:OrganizationMainUser` This example means that only someone in the `OrganizationMainUser` role can see the organization's technical name in the admin service. Field specific permissions override the general permission.
`organization.edit`	This permission defines the users who are allowed to edit organization information in the admin service. You may also define field specific edit permissions by adding the field name after `organization.edit.` Example: `organization.edit.class =` This example means that nobody can edit the organization's type (was previously called "class") in the admin service. Field specific permissions override the general permission.
`organization.create`	This permission defines the users who are allowed to create organizations in the admin service.
`organization.delete`	This permission defines the users who are allowed to delete organizations in the admin service.
`role.read`	This permission defines the users who are allowed to read role assignment information in the admin service.
`role.assign` `role.assign.`*`OrganizationUser`* `role.assign.`*`OrganizationUser`*`.class.`*`test`*	This permission defines the users who are allowed to assign roles to users in the admin service. Role-specific permissions are allowed in `role.assign`. For example, this property defines users that are allowed to assign users to OrganizationUser-role. (Use role's entity name) Organization type (was previously called "class") specific permissions can be used with `role.assign`. For example, this property defines users that are allowed to assign users to `OrganizationUser` role in organizations with type test. Role and organization type are separated with **.class.** keyword. For other organization types `role.assign.OrganizationUser` permission is used. If a role-specific permission is not defined then `role.assign` permission is used.

`role.invite` `role.invite.`*`OrganizationUser`* `role.invite.`*`OrganizationUser`*`.class.`*`test`*	This permission defines those users who are allowed to invite users to roles in the admin service. Role-specific permissions are allowed in `role.invite`. For example, this property defines users that are allowed to invite users to `OrganizationUser` role. (Use role's entity name) Organization type (was previously called "class") specific permissions can be used with `role.invite`. For example, this property defines users that are allowed to invite users to `OrganizationUser` role in organizations with type `test`. Role and organization type are separated with **.class.** keyword. For other organization types `role.invite.OrganizationUser` permission is used. If a role-specific permission is not defined then `role.invite` permission is used.
`role.deassign` `role.deassign.`*`OrganizationUser`* `role.deassign.`*`OrganizationUser`*`.class.`*`test`*	This permission defines the users who are allowed to remove roles from users in the admin service. Role-specific permissions are allowed in `role.deassign`. For example, this property defines users that are allowed to remove `OrganizationUser` role from users. (Use role's entity name) Organization type (was previously called "class") specific permissions can be used with `role.deassign`. For example, this property defines users that are allowed to remove the `OrganizationUser` role from users in organizations with type `test`. Role and organization type are separated with **.class.** keyword. For other organization types `role.deassign.OrganizationUser` permission is used. If a role-specific permission is not defined then `role.deassign` permission is used.
`role.list_approvals`	This permission defines the users who are allowed to list and view role assignment requests in the admin service.
`role.approve`	This permission defines the users who are allowed to approve role assignments in the admin service.
`role.delete`	This permission defines those users who are allowed to delete roles in the admin service.
`role.create`	This permission defines those users who are allowed to create roles in the admin service.
`role.edit`	This permission defines those users who are allowed to edit roles in the admin service.
`role.listusers`	This permission defines those users who are allowed to list users in selected role.
`mandate.read`	This permission defines the users who are allowed to read mandate information concerning received mandates in the admin service.
`mandate.approve`	This permission defines the users who are allowed to approve received mandates in the admin service.
`mandate.remove`	This permission defines the users who are allowed to remove either mandate actuators or the received mandate in the admin service.
`mandate.create`	This permission defines the users who are allowed to create new mandates in the admin service.

`self.read`	This permission defines the users who are allowed to read their own personal information in the self-service interface.
	
	You may also define field specific read permissions by adding the field name after `self.read.`
	
	Example:
	`self.read.custom1 = any:OrganizationMainUser`
	
	This example means that only someone who has the `OrganizationMainUser` role (in any organization) can see the `custom1` field in the self-service interface.
	
	Also authentication methods are handled similarly to user information fields. These permissions define the users who are allowed to read their authentication data in the self-service interface. You can define permission for authentication methods with:
	
	`self.read.method`
	`self.read.method.<authentication method technical name>`
	
	Example:
	`self.read.method.ubikey.sms.1 =`
	
	This example means that nobody is allowed to see the state of the `ubikey.sms.1` authentication method in the self-service interface.
	
	Field specific permissions override the general permission.

`self.edit`	This permission defines the users who are allowed to edit their own personal information in the self-service interface.
	
	You may also define field specific edit permissions by adding the field name after `self.edit.`
	
	Example:
	`self.edit.ssn =`
	
	This example means that nobody can edit their own social security number in the self-service interface.
	
	Also authentication methods are handled similarly to user information fields. These permissions define the users who are allowed to manage their authentication data in the self-service interface. You can define permission for authentication methods with:
	
	`self.edit.method`
	`self.edit.method.<authentication method technical name>`
	
	Example:
	`self.edit.method.ubikey.sms.1 =`
	
	This example means that nobody is allowed to alter the state of the `ubikey.sms.1` authentication method in the self-service interface.
	
	Field specific permissions override the general permission.

`self.role`	This permission defines the users who are allowed to read their role information in the self-service interface.
`self.mobilenoconfirm`	This permission defines the users who are allowed to change their mobile number without the need to confirm the new number.
`self.emailnoconfirm`	This permission defines the users who are allowed to change their email address without the need to confirm the new address.
`self.organization`	This permission defines the users who are allowed to manage organization information in the self-service interface.
`self.requestroles`	This permission defines the users who are allowed to request roles in the self-service interface. These users should also have the `self.role` permission.
`self.request.predefined.roles`	This permission defines those users who are allowed to request predefined roles in the self-service interface. These users should also have the `self.role` permission.
`self.mandate.read`	This permission defines those users who are allowed to read mandate information in the self-service interface.
`self.mandate.approve`	This permission defines those users who are allowed to approve mandates in the self-service interface.
`self.mandate.remove`	This permission defines those users who are allowed to remove mandates in the self-service interface.
`self.mandate.create`	This permission defines those users who are allowed to create new mandates in the self-service interface.
`access.admin`	This permission defines the users who are allowed to access the admin service interface.

`access.admin.organizations`	This permission defines the users who are allowed to access the organization list tab in the admin service interface front page.
`access.admin.users`	This permission defines the users who are allowed to access the user search / list tab in the admin service interface front page.
`access.admin.approvals`	This permission defines the users who are allowed to access the approval tab in the admin service interface front page.
`access.selfservice.personal`	This permission defines the users who are allowed to access the personal tab in the self-service interface.
`access.selfservice.roles`	This permission defines the users who are allowed to access the roles tab in the self-service interface.
`access.selfservice.mandates`	This permission defines the users who are allowed to access the mandates tab in the self-service interface.

Denying operations

CustomerID permissions will fall back to default values in case a given permission is omitted from permissions.properties. In case there is a need to disable a functionality for all users, it is therefore necessary to define the permission entry key, but leave the actual definition empty.

In the example below, the self.read duplicates the default behavior, but self.edit removes all permissions for self.edit, effectively user will see all personal information in self-service, but will not be able to edit it.

```
# permissions.properties

self.read = grp:eIDMUser
self.edit =
```

Equivalent example, note that self.read is commented out, which falls back to default values, effectively user will see all personal information in self-service, but will not be able to edit it.

```
# permissions.properties

#self.read = grp:eIDMUser
self.edit =
```

Opposing example, note that self.read and self.edit are commented out, which make both settings fall back to default values, effectively user will see all personal information in self-service and will be able to edit it.

```
# permissions.properties

#self.read = grp:eIDMUser
#self.edit = grp:eIDMUser
```