



SAP Customer Experience

## Security

SAP Commerce Cloud Developer Training



# The Context



After installing SAP Commerce, it is a best practice to look at **all aspects of security**.

# Basics

## Basics

Type-Based Access Rights  
Restrictions  
Spring Security  
Custom Access Rights  
Additional Security Features

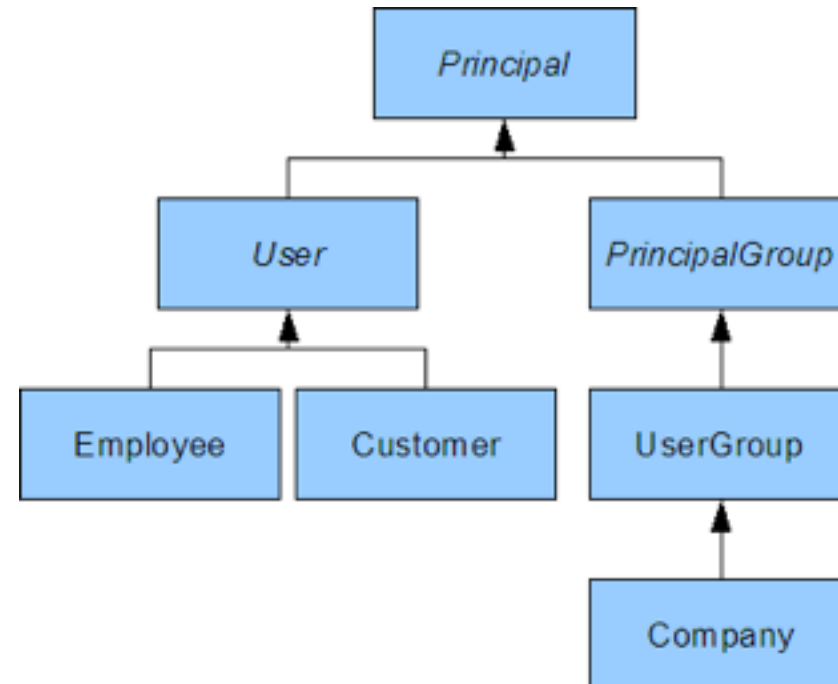


# Security areas to consider

- Web access control
  - IP range
  - Spring security per web app
- Administration rights
  - HAC access
- Data permissions
  - Role-based
  - Type and Item
- Database security
  - Transparent symmetric encryption
  - Field encryption
  - Limiting DB user rights

# Basics

- User accounts in the SAP Commerce Suite can be individual people or roles:
  - who is allowed or is not allowed to authenticate against a part of the application
  - who is allowed or is not allowed to perform specific tasks
- `PrincipalGroup` defines `UserGroup`, `Company`
  - `Company`: Unlike `UserGroups`, `Companies` can hold addresses
- `User` defines: `Employee`, `Customer`
- Default user accounts and groups that cannot be removed:
  - `anonymous` (`Customer`)
  - `admin` (`Employee`)
  - `admingroup`



# Where Do User Accounts Affect the SAP Commerce Suite?

- JaloSession
  - At any given time, a user must be assigned to the JaloSession
- Backoffice
  - The Backoffice displays (or hides) elements depending on the user and the user groups the user belongs to
- Cockpits
  - In addition to user-specific configuration, Cockpits also have workflow integration and can let users manage workflow steps
- Web Services
  - The Web Services system allows user-specific access rights
- Order Process
  - A customer must be associated with shopping cart, either by logging in, or as a guest.
- Addresses
  - The reference to a user account is mandatory
- CronJobs
  - CronJobs use a JaloSession and therefore also require that a user be set

# Type-Based Access Rights

Basics  
**Type-Based Access Rights**  
Restrictions  
Spring Security  
Custom Access Rights  
Additional Security Features



# Type-Based Access Rights – Overview

- Access rights for SAP Commerce types and their attributes
- Access is granted to individual users and/or user groups
- Affect the entire type, not individual items
- Also can affect individual type attributes
- Effective in the Backoffice, Backoffice Cockpits and web services
- The entire type or attribute will be hidden from display to the user account

Advantages	Disadvantages
Attribute-based	Affect the entire type, not just individual instances
Can be imported from and exported to Excel easily	Not effective everywhere (for example, on the ServiceLayer)

Permission Management - User Group ( [employeegroup] )

Filter Context

SEARCH

+

12 items

Context	Read	Change	Create	Remove
Audit Report Data	✗	✗	✗	✗
C2L Item	✓	✗	✗	✗
Catalog	✓	✗	✗	✗
Catalog version	✓	✗	✗	✗
Cx Mapper Script	✗	✗	✗	✗
Employee	✓	✓	✗	✗
Enumeration Value	✓	✗	✗	✗
Item	✗	✗	✗	✗

✗

✓

 Inherited 

✗

 Denied 

✓

 Granted

Filter Attributes

SEARCH

1

/ 2

36 items

Attribute	Read	Change
Active catalog version	✓	✗
Agreements	✓	✗
Assigned Cockpit Item Templates	✓	✗
Base Stores	✓	✗
Buyer	✓	✗
Catalog Version	✓	✗
Catalog Versions	✓	✗
Comments	✓	✗



# Importing

- Type access configuration can be imported through ImpEx

```
$START_USERRIGHTS
Type;UID;MemberOfGroups;Password;Target;read;change;create;remove;change_perm
UserGroup;productManagerGroup;cockpitGroup;;;;;;;;;
# Access Rights for Products & Catalog;;;;;;;;;
;;;Product;+++++;
;;;Product.ean;+---;
;;;Catalog;+++;
;;;Media;+++++;
$END_USERRIGHTS
```

- Full syntax:
  - <https://help.hybris.com/1811/hcd/e472718cafe840c39fbb5ceb00002e52.html>

## API CRUD example

- Generic service for checking permission assignments:

```
permissionCheckingService.checkTypePermission(typeCode,  
  
PermissionsConstants.REMOVE).isDenied();
```

- For typical CRUD permission checking use:

`PermissionCRUDService` – a wrapper over `PermissionCheckingService`

```
permissionCRUDService.canReadType( typeCode );  
permissionCRUDService.canChangeAttribute(typeCode,  
                                         attributeQualifier );
```

# Restrictions

Basics  
Type-Based Access Rights  
**Restrictions**  
Spring Security  
Custom Access Rights  
Additional Security Features



# Restrictions – Overview

## ■ Restrictions / Personalization Rules



- Restrictions define a filter which is added to FlexibleSearch statements at execution time
  - for the specified type
  - for a user or a user group.
- System-wide effect
- The restriction is added to the WHERE clause of the SQL statement produced by FlexibleSearch.

Active ☒ True ☐ False

Identifier

---

**PROPERTIES**

Name <input type="text"/>	 Filter  <input type="text" value="{approvalStatus} = 8796100821083"/>	Restricted Type <input type="text" value="Product [Product]"/>	Apply on <input type="text" value="[customergroup]"/>
---------------------------	--	--	---

Advantages	Disadvantages
Automatically affect every FlexibleSearch	Requires knowledge of FlexibleSearch syntax
Can block access to individual type instances	May require extended SAP Commerce data model knowledge

# Factory-Predefined Access Rights

- A special set of factory-predefined restrictions and type-based access rights:
  - Category
    - Visibility for certain user groups (such as customergroup) to display the categories in the web frontend
  - Catalog
    - Made readable and writeable for certain user groups (such as catalogmanagergroup)
  - Language
    - Made readable and writeable for certain user groups that can have read and write access to several languages

Advantages	Disadvantages
Allows easy rights management for many common use cases	Not very generic approach
Can be used as a starting point for further customization	

# ImpEx example

Restrictions can be imported as any other item through Impex:

```
INSERT_UPDATE SearchRestriction;code;principal(UID);restrictedType(code);active;query  
;FrontRestriction;customergroup;Product;true;{catalogVersion} IN (?  
session.catalogversions)
```

# Spring Security

Basics  
Type-Based Access Rights  
Restrictions  
**Spring Security**  
Custom Access Rights  
Additional Security Features



# Spring security

- Spring security framework takes care of:
  - Restricting access
  - Delegating authentication and authorization
  - Remember me services, login pages etc.
- Spring security framework is used in:
  - Cockpits
  - Accelerators
- Each web application has separate security configuration



## Spring security (2)

- **To enable**

- Use spring delegating filter `springSecurityFilterChain` in your application web.xml

- **To configure**

- Use 'security' xml namespace

```
<security:intercept-url pattern="/my-account*"
                        access="hasRole( 'ROLE_CUSTOMERGROUP' )"
                        requires-channel="https" />
```

- For consistent authentication across all extensions, use Commerce-provided `CoreAuthenticationProvider` (using the Spring Security bean `coreAuthenticationProvider`)
- Customize authentication: extend `CoreAuthenticationProvider` and wire into Spring Security

```
@Override
public Authentication authenticate(...)
{
    User user = getUserByLogin( userDetails.getUserName() );
    Object credential = authentication.getCredentials();
    ... //verify - compare
}
```

# Custom Access Rights

Basics  
Type-Based Access Rights  
Restrictions  
Spring Security  
**Custom Access Rights**  
Additional Security Features



## Custom Access Rights – Overview

- Allows defining very fine-grained access control
- You may define your own permission types
- You can grant or deny permissions to the item instances
- However... don't overuse
  - managing and checking permissions at an item level can considerably degrade performance, since a huge number of items can be involved
  - If possible, use type-based permissions and with restrictions
- Exposed API allows checking of both Item and Type permissions

# Limitations

- Permissions are not automatically enforced.
  - Your code must invoke a method or two from `PermissionCheckingService`
  - By default, only the Media management `SecureMediaFilter` uses item-based rights
- Permission assignments are calculated, hence it's not possible to search for the effective permission assignments using a Flexible Search Query of the form:  
`return all items that have a permission XYZ granted for principal P.`
- Permission assignments are currently not represented by models, so you cannot directly import them using ImpEx scripts.
  - Use the `PermissionManagementService` in ImpEx using bean-shell scripting



So, should we use Custom Access Rights?



Given its many drawbacks, use it only as a last resort

# API example

- Create item permission

```
permissionManagementService.createPermission(permissionName);
```

- Check item permission:

```
permissionCheckingService.checkItemPermission(item, principal, permissionName)  
    .isGranted();
```

- Add item permission:

```
new PermissionAssignment(PermissionsConstants.READ, principal);  
permissionManagementService.addItemPermission(mediaItem, permissionAssignment);
```

- More info:

<https://help.hybris.com/latest/hcd/8c0ae05c8669101481d3cb6fb6bdec16.html>

# Additional Security Features

Basics  
Type-Based Access Rights  
Restrictions  
Spring Security  
Custom Access Rights  
**Additional Security Features**



# Password Security Policies

- The SAP Commerce Platform allows fine control of password handling using:
- Password security policies:
  - Requirements that must be met when setting or changing a password
  - OOTB **regex** and **blacklist** security policies are included
  - You can implement your own policies and enforce them on the storefront
- PasswordPolicyService:
  - Validates user password against defined policy requirements
  - Returns a list of PasswordPolicyViolation objects if validation fails
  - Service is used every time password is set or changed
  - Internally, UserService also delegates its checks to PasswordPolicyService

# Password Change Auditing

- Register all the changes made to your password
- **UserPasswordChangeAudit** is an item type and therefore traceable in the Backoffice
- **UserPasswordChangeAuditPrepareInterceptor** provides logic for recording password changes

The screenshot displays the SAP Administration Cockpit interface. On the left is a navigation menu with options like Home, Inbox, System, OAuth, CORS Filter, Advanced Configuration, Tools, Output Documents, Workflow Administration, Validation, Scripting, Business Processes, Background Processes, Types, Saved Queries, and Backoffice Saved Queries. The main area shows a table of audit records for 'UserPasswordChangeAudit'. Below the table, there is a configuration section for a specific audit item, including fields for Encoded Password, IP Address, Password Encoding, and a checkbox for 'Is blocked for processing'. The 'COPIES' section at the bottom shows target and source synchronization timestamps.

uid	Password Encoding	Encoded Password	PK #	userPK	changingApplication	change
✓ cmsmanager-powertools	pbkdf2	1000:5f784b90af69cd21ef91a3309771625...	8796093972486	8796098134020		admin
✓ cmsmanager-electronics	pbkdf2	1000:389fc639dec6b00b0b3c247f4298e82e...	8796093939718	8796098101252		admin

0 ITEMS SELECTED

UserPasswordChangeAudit[8796093120518]

Encoded Password: 1000:0d2cc75ead8e49c25a4bfa340e18ef  
IP Address:   
Password Encoding: pbkdf2  
Is blocked for processing: ☐ True ☒ False ☐ N/A  
uid: anonymous  
userPK: 8796093087748

COPIES

Target synchronization timestamps:   
Source synchronization timestamps:



## oauth2 Extension

- Replaces the `webservicescommons/oauthauthorizationserver` extension
  - Doesn't introduce significant new functionality
- Exposes the HTTP endpoint as authorization server with 2 endpoints:
  - `/authorizationserver/authorize`
  - `/authorizationserver/token`
- No need to enable authorization server by adding the `oauth2` extension to `localextensions.xml` it's part of the platform extensions
- Configure the `oauth2` extension in `project.properties`
- Manage OAuth clients and access tokens using the `system/OAuth` tab in the Backoffice

# Securing HAC Using Roles

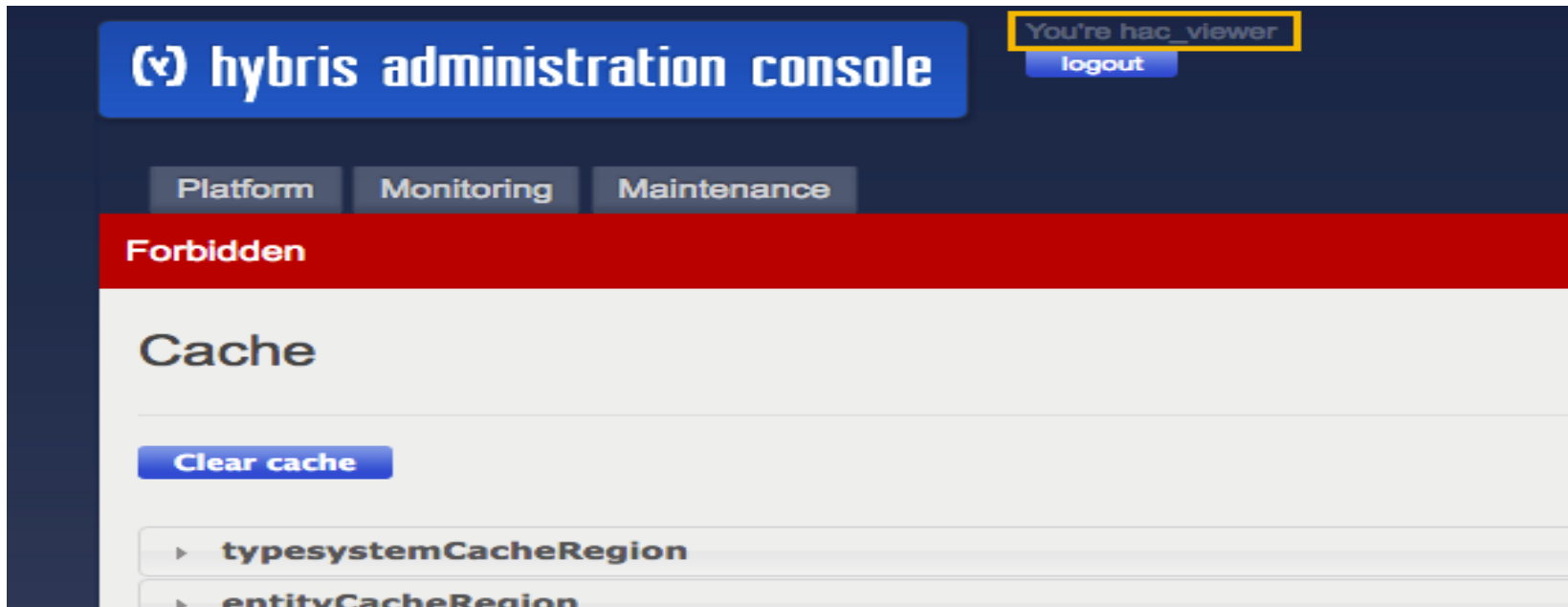
- Configure particular user access to different areas in the Hybris Administration Console:
  - Based on assigned roles, users have access to specific HAC tabs or actions
  - The HAC provides multiple predefined roles OOTB
    - You can configure your own roles using Spring Security
  - Roles are represented by userGroup entries in the database
  - Roles are imported as essential data during system initialization or update
  - Each role, and the urls it grants access to, are configured in the `spring-security-config.xml` file (using the tag `intercept-urls`)

# Securing HAC Using Roles - Example

- Only users with the ROLE\_HAC\_MONITORING\_CACHE role can clear the cache
  - configured in spring-security-config.xml:

```
<intercept-url pattern="/monitoring/cache/**/clear"  
    access="ROLE_ADMINGROUP, ROLE_HAC_MONITORING_CACHE"/>
```

```
<intercept-url pattern="/monitoring/cache/**"  
    access="ROLE_ADMINGROUP, ROLE_HAC_MONITORING_CACHE,  
ROLE_HAC_MONITORING_CACHE_LIMITED"/>
```





SAP Commerce uses **type-based access rights** to grant access rights at the **type** or **attribute** level for different users or userGroups

- Type-based access rights normally only affect Backoffice, Backoffice Cockpits and web services; if you want to apply them in your code, use the `PermissionCheckingService`

A **restriction** will add a search condition to a flexible search statement when the current user/usergroup and type match the user/usergroup and type specified in the restriction.

- Defined in Backoffice or ImpEx
- Restrictions work at the instance level and have a **system-wide** effect!

**Spring security** restricts web access, and is configured individually for each web application

**Custom access rights** work at the instance level; defined by `PermissionManagementService` and `PermissionCheckingService`

- Do **not** overuse to avoid impacting performance

# Exercise

## Security



# Thank you.