Exporting Permissions to Hana: Declarative Approach

Authors: Jakub Sendor  
Dr Slim Trabelsi  
Stuart Short  
Company: SAP Research, SAP Labs, France  
Created on: 19 April 2013

Author Bio

Jakub Sendor: is a research associate working in the Security & Trust research practice at SAP Labs France in Sophia-Antipolis. He is involved in the prototypes development initiatives around declarative authorization policies as well as working in A4Cloud (Accountability for Cloud) EU publicly funded project. His main interest comprise of cloud computing, dynamic programming languages and enterprise Java systems.

Slim Trabelsi PhD is a Senior researcher in the Security & Trust practice at SAP Labs France Sophia-Antipolis. He is currently managing the security WP of the EU PPP Fi-Ware project and the EIT – trust in the Cloud EU Project. His main research fields are: privacy, trust, Authorizations, cloud, social networks, anonymization.

Stuart Short is a Senior researcher in the Security & Trust practice at SAP Labs France Sophia Antipolis. He is currently managing, for SAP, OPTET an EU publicly funded project based on trustworthiness in the Cloud. Main research interests include privacy, trust, cloud computing and business process management.
TABLE OF CONTENTS

AUTHOR BIO ......................................................................................................................... 1
ABSTRACT ............................................................................................................................ 3
MOTIVATION .......................................................................................................................... 4
APPROACH ............................................................................................................................ 4
TECHNOLOGY OVERVIEW ............................................................................................... 5
SAP ABAP Authorizations ................................................................................................. 5
Introducing XACML ............................................................................................................ 6
TRANSLATING ABAP AUTHORIZATIONS INTO XACML RULES. ........................................ 7
ENCODING TO HANA .......................................................................................................... 8
Storing the Authorization Policy Together with the Data ....................................................... 9
Dedicated Table Acting as Policy Repository .................................................................... 10
CONCLUSION ..................................................................................................................... 11
REFERENCES ..................................................................................................................... 11
ABSTRACT

Migrating authorizations and permissions from traditional SAP systems to HANA is not yet an automated task. In order to avoid migration overhead, errors, and inconsistencies we propose a translation mechanism that transforms any authorization model to a flexible XML based authorization standard called XACML. From this XML representation we generate HANA authorization filters that reflect the original authorization rules and execute them on HANA without affecting the performance.
MOTIVATION
Currently at SAP we are in a migration phase to SAP HANA. Most of the SAP core applications and systems are currently replicating their databases to HANA DB through the accelerators and the other replication enablers. Some of them are partially migrated (Hana DB is used only for analytics, but traditional transactions are still executed on the traditional DB), and others are fully migrated (all the transactions are executed on HANA).

When the data is transported from the traditional DB to HANA, the permissions and authorizations related to it are not automatically exported. Every DB administrator or developer has to re-write the permissions manually on HANA. Such a manual approach is not optimal in terms of time/resource overhead, maintenance, and correctness. There is always a risk that the new permissions defined for the HANA dataset do not reflect the original ones, especially when the permission schema is quite complex.

Some other approaches exist, like keeping the authorization check on the application level in order to filter the result set received from HANA, to remove from it the rows that user do not have access to. Such an approach is more consistent than the manual one, but it introduces a huge overhead during the authorization checks, especially when analytic requests are sent to HANA. With such an approach, any performance benefit introduced by HANA is likely to be lost.

In our research team, we propose a declarative approach that can be used to automatically generate and maintain the new Hana permissions. As an example, we decided to use ABAP authorization objects as input that should be migrated to HANA. The original permissions are imported, transformed into a standard authorization declarative language (XACML) and then exported as HANA permissions or filters.

APPROACH
The three-step approach depicted below explains how the authorizations can migrate from an ABAP application system to HANA through an XACML translation. We take the example of a migration between an ERP system, where authorizations are defined in ABAP, and a HANA framework where the same authorizations must be enforced without affecting the performance.

1. In a traditional SAP ERP system, the authorizations and permissions are declared and defined in the code of the application as ABAP authorization objects where restriction rules are applied. We propose to import these authorization objects, parse the restriction rules and map them to an XACML authorization model.
2. This mapping will generate an authorization policy expressed according to the XACML standard and represented in an XML file. This file will contain the equivalent of the ABAP authorization object and the restrictions rules defined in the original system. This policy can then be easily maintained, modified, updated and managed.
3. The content of the XACML authorization policy is then transformed into a HANA filter. This HANA filter is in practice an SQL query defining the restriction imposed by the policy. The advantage of using this kind of filters in the HANA platform is the preservation of the high performance querying system. The authorization check is then executed at the HANA level as a pre-query to the query coming from the system.

All the three steps depicted here are executed automatically to avoid any inconsistency or mistake introduced by a manual human intervention.
TECHNOLOGY OVERVIEW

SAP ABAP Authorizations
The SAP ABAP authorization system is targeted at protecting SAP system resources such as:

• Transactions
• Programs
• Services
• Program Components
• Business Objects
• ...

All the above are protected by authorization objects which are defined based on the activity and responsibilities of the user. Authorization fields are data elements stored in the ABAP dictionary and their corresponding assignments. The authorization fields specify the category of the resource data or transaction. The values of the fields are defined by a system administrator.
While performing the authorization check, the system compares the authorization field values specified by the programmer in the ABAP CHECK statement with the values retrieved from the authorization object specified for the user.

Example:
We will take here a very simply example from the ABAP Authorization Concept documentation [1]

A Travel Request transaction with code ‘S_TRQST’ is protected by an authorization object named ‘S_TRVL_BKS’.

S_TRVL_BKS’ is defined as follows:

```
// Authorization Field (Activity represented as ACTVT from ABAP Dictionary)
ACTVT
// Authorization Field (Activity represented as Customer Type from ABAP Dictionary)
CUSTTYPE
```

The programmer then creates an AUTHORITY CHECK to instantiate the authorization object and fills the value of the fields:

```
AUTHORITY-CHECK OBJECT ‘S_TRVL_BKS’
    ID ‘ACTVT’ FIELD ’02’
    ID ‘CUSTTYPE’ FIELD ’B’
IF SY_SUBRC < > 0.
    MESSAGE E
END IF
```

With the above authority check, the application can determine if the user has the relevant authorizations and whether these authorizations are appropriately defined.

Authorizations are instances of generic authorization objects which define their structure (list of authorization fields). An authorization profile is a bundle of authorizations which can be assigned to a user role. To access the programs, services, components, business objects or to execute SAP transactions, a user requires all the corresponding authorizations.

**Introducing XACML**

XACML [2] stands for eXtensible Access Control Markup Language. The standard defines a declarative access control policy language represented in XML and specified by the OASIS standards organization. The XACML language model is based on the three main elements that create a structured way to formulate an access control restriction: policy set, policy and rule. Using these three elements it is possible to create very complex policies, describing sophisticated authorization requirements and imposing additional conditions on an access requestor, as well as very simple policies, e.g. denying access to the particular user.

The basic element of the policy is a `<Rule>` element that defines: 1) an (optional) target to which the rule is applied by describing it as a set of attributes and 2) the effect of applying the rule. It also provides the possibility to express additional conditions that could further restrict applicability of the rule (mathematical and logical functions).

The target element consists of four elements: 1) subject, which specifies the attributes of the requestor 2) resource, which describes the set of attributes characterizing the asset protected by the access control policy 3) action, which defines the set of allowed operations that could be performed on the resource 4) environment that can provide additional data that is required to take authorization decision, independent of subject, resource and action (e.g. date and time).

The effect of the rule is one of the two possible values: "Permit" or "Deny". It states that if the rule is applicable for the incoming request than the access could be granted or not, respectively. There is, however,
a possibility that the attributes stated in the rule’s target element do not match those in access request. In this case, the rule would be interpreted as “Not Applicable”.

**TRANSLATING ABAP AUTHORIZATIONS INTO XACML RULES**

As we have already seen, authorization check in ABAP is a function which checks if the values of the authorization fields in a certain authorization object for a given user are equal to the values specified by the programmer in the invocation of the check. As such, this ABAP construct corresponds to XACML policy which in its basic element – policy or rule target – checks whether the attributes in XACML request matches the values recorded in the policy.

We will take here the simply example from the section above. Just as reminder, here is the coding for the authorization check in this program:

```abap
AUTHORITY-CHECK OBJECT 'S_TRVL_BKS'
  ID 'ACTVT'    FIELD '02'
  ID 'CUSTTYPE' FIELD 'B'.
IF SY-SUBRC <> 0.
  RAISE bad_authorization.
ENDIF.
```

As we will show, the translation of this syntactical construct into an XACML rule is in most cases a step that can be performed automatically, so we can simply ignore the actual meaning of the authorization fields. Nevertheless, some of them (which are connected to the ABAP Dictionary) appear very often in the authorization objects and we can easily classify them as one of the XACML rule target element attributes (subject, resource or action attributes). Here it is the case for ‘ACTVT’ field which stands for ‘activity’. This field corresponds directly to the Action element in the rule target. It takes two values: 02 for change and 03 for display. In the XACML representation of this authorization check, ‘activity’ will be the Action element attribute.

Thus, the corresponding XACML rule will look as follows. Multiple such rules can subsequently be combined into an XACML policy:

```xml
<Rule Effect="Permit" RuleId="permit-request-creation">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">change</AttributeValue>
        </ActionMatch>
        <ActionAttributeDesignator AttributeId="activity" MustBePresent="true" />
      </Action>
    </Actions>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">B</AttributeValue>
        </SubjectMatch>
        <SubjectAttributeDesignator AttributeId="customer-type" MustBePresent="true" />
      </Subject>
    </Subjects>
  </Target>
</Rule>
```
In the target of this XACML rule, we specify the following elements: one attribute value for the action attribute and another attribute value for the subject attribute. If, during policy evaluation, the values of these attributes (i.e., change for activity and B for customer-type) are matched (by applying the string-equal function which is nothing more than just string equality) the result of this rule will evaluate to “Permit” as it is specified in the “Effect” attribute of the rule.

In this very basic example, we see that the translation step can be performed automatically by analyzing the source code of ABAP programs and retrieving the information stored in authorization check statements. To provide more meaningful policies, it is also necessary to incorporate information recorded in the ABAP Dictionary (in order to, e.g., make use of full attribute names instead of ABAP mnemonics and to distinguish which of the authorization fields should be represented as subject attributes and which as action attributes).

**ENCODING TO HANA**

Transforming XACML authorization policies into HANA filters requires the implementation of an policy engine to be able to:

- Match the corresponding policy with the data access request coming from the application
- Generate the query filter related to the applicable policy
- Execute the query filter and send back the response to the application

We have adjusted the XACML engine architecture defined in the standard version 2.0 (2) to fulfill these requirements. The design diagram is presented in Figure 3. The main difference between this solution and the standard architecture is the involvement of the Policy Administration Point (PAP) before the incoming access control request enters the engine. The classical approach triggers the invocation of the PAP from the Policy Decision Point (PDP), during the request evaluation.

In the XACML for HANA solution, the PAP takes care of invoking the process of encoding policies into database tables, either by storing them next to the actual data (by extending data tables structure) or in a separate policy table. The encoding is described in more details in next paragraphs.

![Figure 3 Proposed XACML Engine Architecture for HANA](image-url)
We propose two approaches to implement the encoding:

1. Storing the authorization policy together with the data; a list of subjects who can access the data, actions that can be performed on the data and extra conditions are recorded as additional columns in the database tables, for each data table.
2. Define a dedicated table which acts as translated policy repository; the original data model remains unchanged, except that an additional table is introduced to capture information retrieved from XACML policy (subjects, resources, actions).

**Figure 4: Storing Authorization Together with Data - Simple Policy Encoding**

**Storing the Authorization Policy Together with the Data**

In the first approach, XACML policies are analyzed upon being fed into Policy Administration Point (PAP). For each resource enlisted in the policy, the set of subjects and actions is retrieved and is used to populate the respective columns in the data tables stored in HANA. A couple of different techniques for associating the subject and action with each row could be envisioned.

The simplest method is where a set of subjects or actions are stored as a concatenated list of strings. This is depicted in the Figure 4, where at the bottom an example of the modified data table is shown. Two additional columns (Subjects and Actions) record the translated information from the policy. Further columns, according to different attributes, might be necessary.
In another method (Figure 5) the data rows are duplicated according to the possible combinations of subject and action attributes, so that in each row a single subject and an associated action is reflected. This approach simplifies the filtering function but does so at the expense of substantially increasing the data volume. Recall that HANA implements compression techniques (e.g., run length encoding) in which repeated values in columns are stored very efficiently. On the other hand, other databases may require significantly more disk space to represent this data volume. Another concern arises with respect to the modification or the deletion of duplicated data rows, as from the application perspective they should be perceived as a single logical entity. Thus row modification and deletion operations would also need to be handled specifically (as would be read operations), to update or remove multiple all physical rows that belong to logical entity. This might be seen as big performance overhead, however HANA is an insert-only-database [3] so updating or deleting entries is only seen as logical operation which technically is optimized to execute much faster than in traditional disk-based databases.

Both methods introduce an initial overhead when it comes creating data tables. Also, each creation and update of the row is influenced in the sense that the additional columns storing the policy information have to be populated with the extra data coming from the Policy Administration Point.

**Dedicated Table Acting as Policy Repository**

The second approach to store the authorization policy in HANA does not impose any modifications on the original data structures. All information retrieved from the policy is represented as a single table which in the simplest form comprises of three columns: Subject, Resource and Action – with possible additional columns for specific conditions. The Resource column is necessary in this case, because in contrast to the first approach, the authorization information is not inserted directly into the data row that it controls access to. The structure of this approach is presented in Figure 6.
CONCLUSION

In this document we presented our solution to address the crucial problem of transferring authorizations and permissions from SAP ERP systems to SAP HANA. Our approach is based on the translation of the different source authorizations into an authorization standard XACML. This declarative approach based on the XACML standard offers a flexible solution to manage and maintain the authorizations and permissions in a decoupled way (without touching the application’s source code). From these XACML authorization policies, we automatically generate HANA SQL filters that assume the role of the ABAP authority checks. These SQL HANA filters offer the advantage of preserving the high performance analytics functionalities by integrating the permissions rules into the data stored in HANA.

The main advantages offered by our authorization exporting tool are:

- Automate the import and translation of the authorizations and permissions from the source systems
- Use a declarative authorization policy standard to easily manage and maintain the authorization rules
- Avoid human errors during the authorization export to HANA
- Avoid computation overhead due to performing authorization checks on the application side
- Benefit from the high performance capabilities of HANA to boost the authorization checks
- Propose a decoupled tool that can be adapted to any system independently of the configuration.

REFERENCES


