EXTENDING XACML ACCESS CONTROL ARCHITECTURE TO DECOUPLE AUTHORIZATION DECISIONS FROM ENTERPRISE APPLICATIONS

Authors
Abhinav Gupta
Consultant Architect

Sapient® Global Markets

Most enterprise applications implement access control and authorization in their own proprietary manner, which results in extremely tight coupling of authorization decisions within applications. In many cases, implicit authorization decisions are embedded throughout the application. Embedded authorization permits users to perform a certain action without going through an access-control check from a third-party system. This approach can lead to serious control issues — eroding trust in applications and potentially exposing highly sensitive data to end users. Moreover, with embedded authorization, it becomes virtually impossible to obtain a consolidated view of the policies protecting the various resources of enterprise applications. These security issues can be controlled by centralizing and shifting authorization decisions to a separate place and then accessing them whenever and wherever authorization decisions need to be made. The eXtensible Access Control Markup Language (XACML) technology presents a simple and powerful solution for doing so. The following sections describe a detailed study of XACML technology and a step-wise approach for incorporating this technology in various enterprise applications.

CONTENTS

Introduction 4

Implementation 7

Case Study 9

Conclusions & Recommendations 17

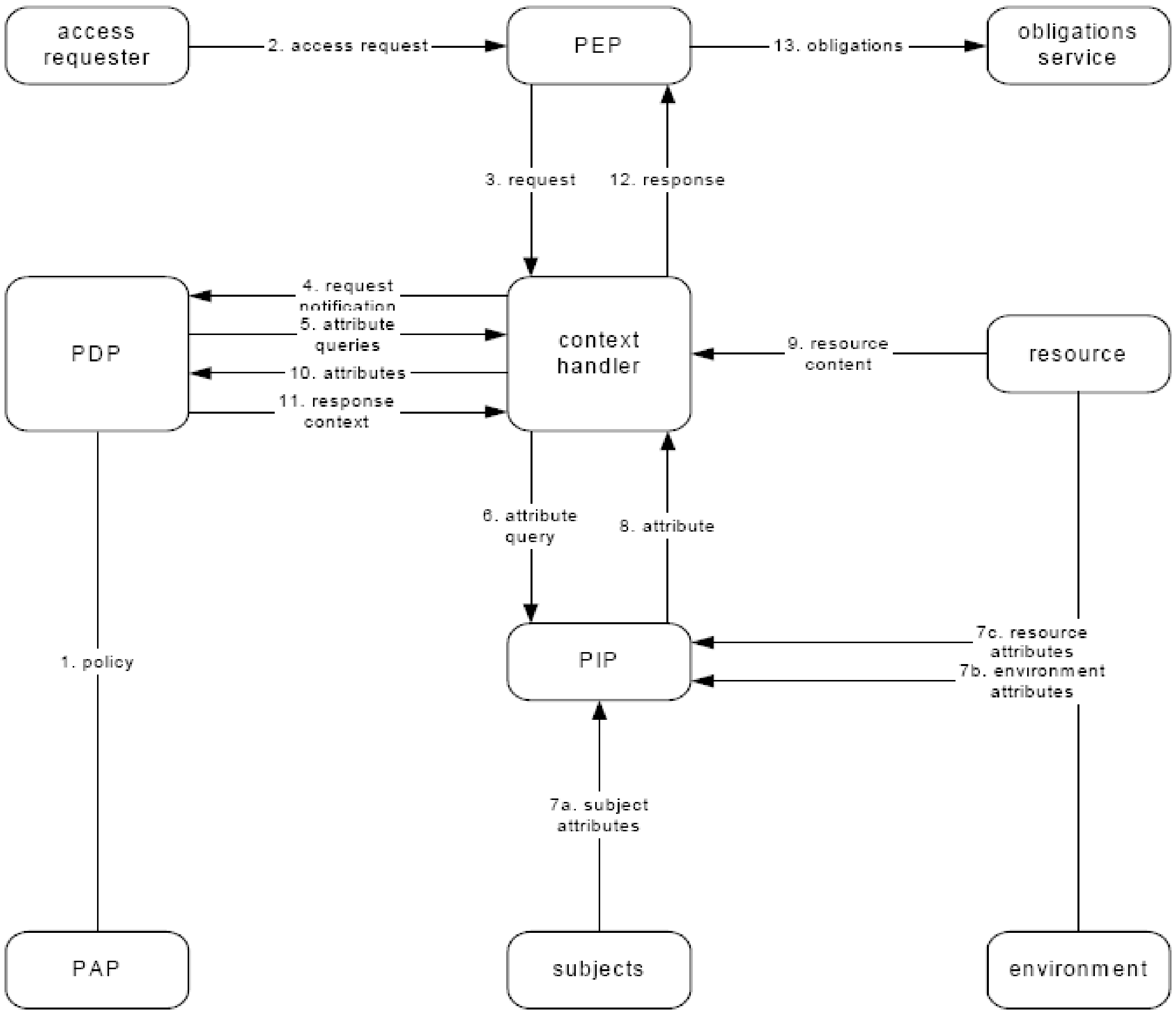
References 17

INTRODUCTION

The security policy of a large enterprise has many elements and points of enforcement. For most enterprises, the current practice is to manage the configuration of each point of enforcement independently in order to implement the security policy as accurately as possible. An unintended consequence of this approach is that modifying the security policies becomes expensive and often unreliable. It is also virtually impossible to obtain a consolidated view of the safeguards in effect throughout the enterprise to enforce the policy. At the same time, consumers, shareholders and regulators are increasingly pressuring corporate and government executives to demonstrate best practices in the protection of the information assets of the enterprise and its customers.

The eXtensible Access Control Markup Language (XACML) is technology developed to standardize the access control and authorization decisions in enterprise applications. It is based on XML language and therefore easy to understand and implement and easy to "plug in" across various tools and technologies.

The diagram below depicts the high-level flow of XACML—including various interfaces and actors involved in making authorization decisions.



The table below gives a brief definition of each of the components of the flow diagram:

#	Components	Description
1	Policy	Policies contain the rules that form the basis to arrive at an authorization decision. Policies are written in an XML document using XACML tags.
2	PAP	Policy Administration Point (PAP) acts as a repository of policies and makes them available to PDP.
3	PEP	Policy Enforcement Point (PEP) acts as the endpoint for authorization request and response.
4	PDP	Policy Decision Point (PDP) evaluates a request against available policies to make a decision.
5	PIP	Policy Information Point (PIP) acts as the source of various attribute values.
6	Context Handler	Context handler serves as "translator"— converting a request from its canonical form into XACML format and an XACML response into its canonical representation.
7	Obligations	The Obligations service is an operation specified in policy that must be executed by PEP before sending a response back to requester.

Because multiple policies can apply to a given request, there must be a mechanism to resolve conflicting decisions by the policies. XACML answers this concern through various combination algorithms as described below:

#	Algorithm	Description
1	Deny-overrides	If any of the policies or rules denies, the entire response says denied.
2	Permit-overrides	If any of the policies or rules permits, the entire response says permit.
3	First-applicable	Result is evaluated based on first rule or policy from the list of applicable rules.
4	Only-one-applicable	Result ensures that only one policy is applicable to the incoming request.

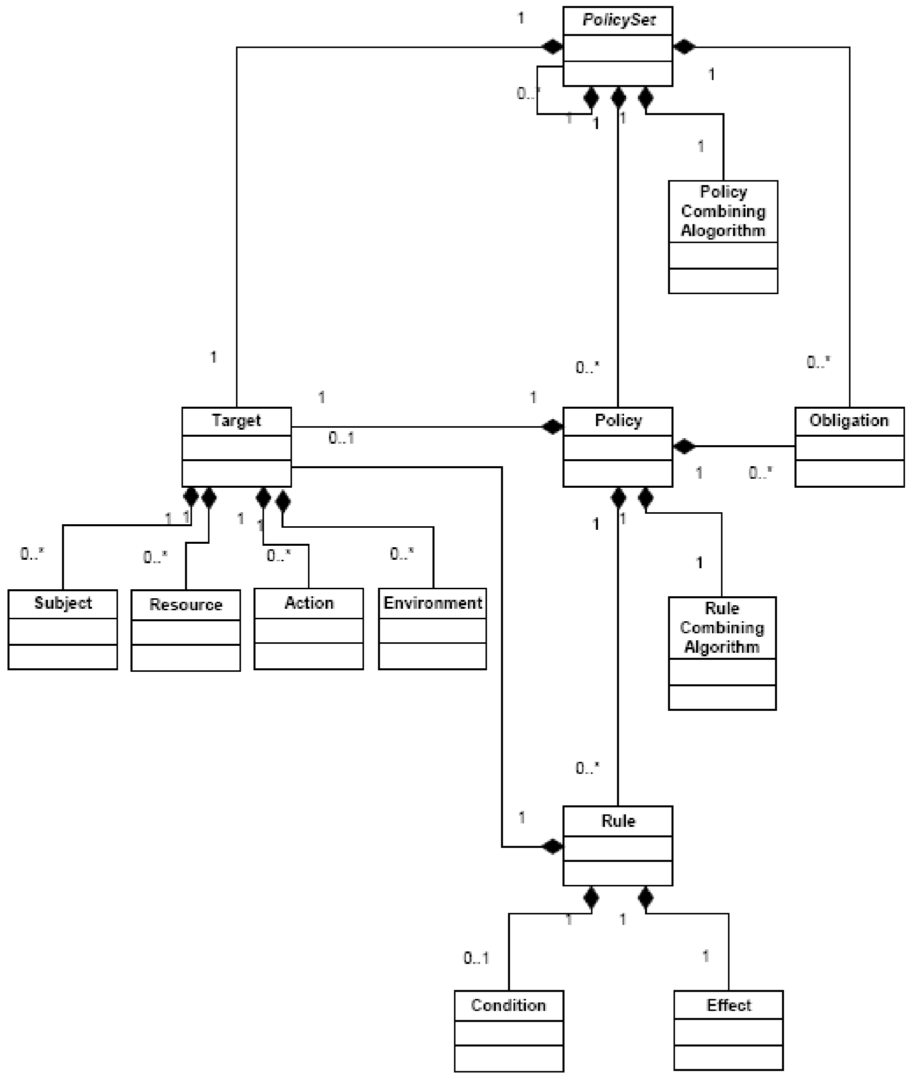
Introduction (continued)

The business functions and the rule applicable on those functions are defined in the policy document. Policy documents can be segregated based on the mutually exclusive business functions they represent. A policy set is created containing a reference to all the policy documents that act as a single point of entry to access the physical policy files.

At the core of every policy is a rule that stores the information to arrive at the answer to this fundamental question: *Can a subject perform a particular action on a particular resource in a particular environment?*

Various policies or rules are indexed by providing a target element on top of policies and rules. The XACML engine would proceed to evaluate a particular request only if the target is applicable to incoming request parameters. The effect of a rule can be "permit" or "deny" as defined along with the rule.

The diagram below depicts the policy model of the XACML language:



IMPLEMENTATION

The implementation of XACML starts with the identification of all implicit and explicit authorization requirements for the application. These requirements form the basis of the policy document that is used to validate a client's request to permit or deny. The security policy of a large enterprise has many elements and many points of enforcement.

Thus, any policy is a unique combination of subject, action, resource and environment. The table that follows illustrates a sample definition of policies:

#	Subject	Role	Resource	Action	Environment	Description	Obligations
1	Any active user	Trade-admin	Acumen	Navigate	GM	Any active user of Trade-admin role can navigate Acumen in GM	No Obligations
2	Any active user	Trade-admin	Acumen	Create	GM	Any active user of Trade-admin role can create data in Acumen in GM	Only trade life cycle data can be created by Trade-admin role
4	Any active user	Super-admin	Acumen	Navigate	GM	Any active user of Super-admin role can navigate Acumen in GM	No Obligations
5	Any active user	Super-admin	Acumen	Create	GM	Any active user of Super-admin role can create data in Acumen in GM	No Obligations

Implementation (continued)

In enterprise applications, authorization is typically divided into the following categories:

Coarse-grained Security

Coarse-grained security authorizes a user at a broader level—validating whether or not a user is allowed to access the application and, if allowed, which parts of application the user is allowed to see. Implementing coarse-grained security in an application involves multiple steps:

- Coarse-grained security first authorizes whether or not a user is allowed to log in into the application.
- After allowing a user to log in, an enterprise application generally shows various menus and submenus to the user.
- These menus and submenus represent mutually exclusive business functions that a user can perform after navigating through these menus.
- The conditions and rules under which a user in a particular role can see the menu should be configured in the policy document.
- At the time of login, requests should be made to the XACML policy document to check whether or not a particular menu should be shown to the user.

Fine-grained Security

Fine-grained security performs validations at the data level — validating whether or not a user can access specific data. Fine-grained security can be further broken down into three levels:

- **Fine-grained security for "read" operations**
  - o At this level, data that is shown in any particular business functionality is filtered according to the user's credentials.
  - o These filters should be placed in pluggable components that add the filter criteria to the data query depending upon the requested business function.
- **Fine-grained security for updates**
  - o This level imposes additional validations before data is created, updated and/or deleted in the database.
  - o This level is important because it is possible that the user's authority to create, delete and/or update data may change after data is shown to the user.
  - o Fine-grained security is mostly valid in cases where users are allowed to export data to Microsoft Excel or XML.
  - o Such exported data can be modified and re-uploaded by user, but time may elapse between export and import; during that time, the user's permissions to update that data could be revoked.

CASE STUDY

The back offices of large investment banks maintain huge repositories of trade and reference data. Data is generally reconciled to verify the data present in the destination. After correctly reconciling the data, the bank generates a balance sheet, which describes its profit and loss figures. This balance sheet data is composed of data generated at various trading desks and thus only the traders who created that data can reconcile whether or not it is correct. In a typical process, there would be provisions to show balances to a user for verification.

The following steps should be executed to secure the complete application:

a. Define all the policies in "plain English" in an Excel File.
b. Create a policy set structure to depict the structure of policies.
c. Create a policy XACML file for login business functionality.
d. Create policy XACML files for all other coarse-grained policies.
e. Create policy XACML files for all fine-grained policies.
f. Convert client's request into an XACML format.
g. Get response in XACML format.

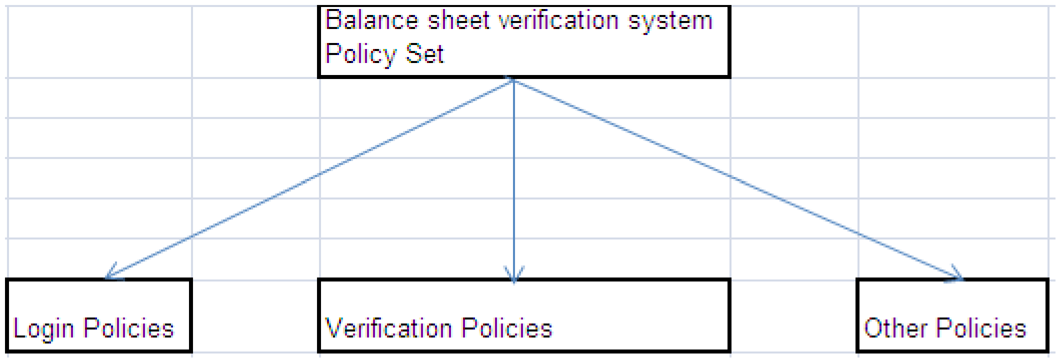
Step 1:

Define all the policies in "plain English" in an Excel File. For the case study, some of the policies would be as follows:

#	Subject	Role	Resource	Action	Obligations
1	Any active user	Trade-admin	Balance sheet	Navigate	No Obligations
2	Any active user	Trade-admin	Balance sheet	Read	Only authorized balances can be read
3	Any active user	Trade-admin	Balance sheet	Update	Only authorized balances can be updated
4	Any active user	Super-admin	Balance sheet	Navigate	No Obligations
5	Any active user	Trade-admin	Balance sheet	Read	Only authorized balances can be read
6	Any active user	Super-admin	Balance sheet	Update	Only authorized balances can be updated

Case study (continued)

Step 2:
Create a policy set structure (example below).



Step 3:
Create a policy XACML file for login business functionality (example below).

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
     xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation = "urn:oasis:names:tc:xacml:2.0:policy:schema:os       access_control-xacml-2.0-policy-
schema-os.xsd"

PolicyId="urn:oasis:names:tc:xacml:2.0:company:company_balances_verification_sytem_policies:policy"

RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
   <Description/>
   <Target/>
   <Rule RuleId = "urn:oasis:names:tc:xacml:2.0:company: balances_verification_sytem_rules:rule"
   Effect="Permit">
   <Description>
   </Description>
   <Target>
   <Subjects>
     <Subject>
         <SubjectMatch MatchId = "urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">trade_admin</AttributeValue>
   <SubjectAttributeDesignator
   AttributeId="urn:oasis:names:tc:xacml:1.0:subject:role-id"
   DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
```

```
    </Subject>
  </Subjects>
<Resources>
<Resource>
  <ResourceMatch
MatchId=”urn:oasis:names:tc:xacml:1.0:function:anyURI-equal”>
  <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#anyURI”>urn:company:balances</AttributeValue>
  <ResourceAttributeDesignator
AttributeId=”urn:oasis:names:tc:xacml:1.0:resource:resource-id”
DataType=”http://www.w3.org/2001/XMLSchema#anyURI”/>
  </ResourceMatch>
 </Resource>
</Resources>
<Actions>
 <Action>
   <ActionMatch
MatchId=”urn:oasis:names:tc:xacml:1.0:function:string-equal”>
      <AttributeValue
DataType=”http://www.w3.org/2001/XMLSchema#string”>access</AttributeValue>
      <ActionAttributeDesignator
AttributeId=”urn:oasis:names:tc:xacml:1.0:action:action-id”
DataType=”http://www.w3.org/2001/XMLSchema#string”/>
   </ActionMatch>
 </Action>
</Actions>
</Target>
</Rule>
</Policy>
```

The sample policy document specifies that any user of role trade_admin can access balances in the balances verification system.

Optionally, policies can be supplemented with obligations that mandate a PEP to perform certain operations before sending a response back to the client (example below).

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Policy
     xmlns=”urn:oasis:names:tc:xacml:2.0:policy:schema:os”
     xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
     xsi:schemaLocation =
“urn:oasis:names:tc:xacml:2.0:policy:schema:os
      access_control-xacml-2.0-policy-schema-os.xsd”
```

Case study (continued)

```
PolicyId="urn:oasis:names:tc:xacml:2.0:company: balances_verification_system_policies:policy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description/>
  <Target/>
  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:company: balances_verification_system_rules:rule"
     Effect="Permit">
    <Description/>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">John Smith</AttributeValue>
            <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:company:peopleportal:vox</
AttributeValue>
            <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">access</AttributeValue>
            <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

```
                </ActionMatch>
              </Action>
           </Actions> </Target>  </Rule>
<Obligations>
 <Obligation
            ObligationId="urn:company: balances_verification_sytem:trade_admin_obligation_1"
            FulfillOn="Permit">
            <AttributeAssignment
            AttributeId="urn:company:context:function"
            DataType="http://www.w3.org/2001/XMLSchema#string">restrictBalances(subjectId,
roleid)</AttributeAssignment>
                        </Obligation>
            </Obligations>
 </Policy>
```

The sample policy document specifies that any user of role trade_admin can access balances in the balances verification system. However, an obligation defined by the "restrict balances" method shows only relevant balances to the user of trade_admin role.

Step 4:
Convert client's request into an XACML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request
     xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="urn:oasis:names
:tc:xacml:2.0:context:schema:os
       access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
       <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:role-id"
           DataType="http://www.w3.org/2001/XMLSchema#string">
           <AttributeValue>trade_admin</AttributeValue>
       </Attribute>
   </Subject>
   <Resource>
       <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>urn:company:balances_verification_system</AttributeValue>
</Attribute>
</Resource>
<Action>
```

Case study (continued)

```
  <Attribute
    AttributeId=”urn:oasis:names:tc:xacml:1.0:action:action-id”
          DataType=”http://www.w3.org/2001/XMLSchema#string”>
        <AttributeValue>access</AttributeValue>
     </Attribute>
  </Action>
  <Environment/>
</Request>
```

The above request states that a user of role trade_admin is asking permission to access the balances verification system.

Step 5:
Get response in XACML format.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Response
      xmlns=”urn:oasis:names:tc:xacml:2.0:context:schema:os”
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:policy=”urn:oasis:names:tc:xacml:2.0:policy:schema:os”
      xsi:schemaLocation=”
      urn:oasis:names:tc:xacml:2.0:context:schema:os access_control-xacml-2.0-context-schema-os.xsd
      urn:oasis:names:tc:xacml:2.0:policy:schema:os access_control-xacml-2.0-policy-schema-os.xsd”>
   <Result ResourceId=”urn:company:balances_verification_system”>
      <Decision>Permit</Decision>
      <Status>
         <StatusCode
             Value=”urn:oasis:names:tc:xacml:1.0:status:ok”/>
      </Status>
   </Result>
</Response>
```

The sample response states that the user is permitted to access the balances verification system.

For policies appended with obligation, the response would contain obligations (example below).

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Response
           xmlns=”urn:oasis:names:tc:xacml:2.0:context:schema:os”
           xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
           xmlns:policy=”urn:oasis:names:tc:xacml:2.0:policy:schema:os”
           xsi:schemaLocation=”
           urn:oasis:names:tc:xacml:2.0:context:schema:os
```

```
access_control-xacml-2.0-context-schema-os.xsd
            urn:oasis:names:tc:xacml:2.0:policy:schema:os access_control-
xacml-2.0-policy-schema-os.xsd">
            <Result
ResourceId="urn:company:balances_verification_system">
                        <Decision>Permit</Decision>
                        <Status>
                                    <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
<Obligations>
 <Obligation
            ObligationId="urn:company: balances_verification_sytem:trade_admin_obligation_1"
            FulfillOn="Permit">
            <AttributeAssignment
            AttributeId="urn:company:context:function"
            DataType="http://www.w3.org/2001/XMLSchema#string">restrictBalances(subjectId,
roleid)</AttributeAssignment>
                        </Obligation>
            </Obligations>
            </Result>
</Response>
```

This response states that although the user is allowed to access the balances verification system, there is an obligation that filters the balances that can be shown to the user.

Authorization of bulk requests

The approach described above suits most authorization requirements. However, it suffers from some inherent performance drawbacks for bulk authorization, as multiple requests need to be made to XACML PDP to verify whether or not a client is allowed to proceed. Hence, for every XACML implementation, carefully assess the following approaches:

- **Approach 1: Single-threaded, multiple thin requests to XACML engine**
  This approach makes many requests to the XACML rule engine—addressing each and every record in the client's request in a single-threaded manner. This approach is the simplest and easiest and represents the most typical way of using the XACML language.

  For bulk requests, however, this approach suffers the inherent drawback of repetitive calls to the XACML rule engine, which can slow collective response time. The problem is further aggravated when a decentralized XACML rule engine is available remotely (not locally) — causing an enormous load on network traffic that further slows response time to the client.

- **Approach 2: Single-threaded, single-thick request to XACML engine**
  This approach makes a single request to the XACML rule engine and carries details of all the records of

a client's request. While the approach may appear better than the first, it suffers from performance issues due to the bulky size of the XACML request document created. Transferring the bulky XACML request document over the network causes significant performance deterioration due to marshalling and un-marshalling of the huge request document.

Another functional issue results from the XACML engine producing a single response to the entire request document. This causes problems when partial permit/deny responses from XACML are considered as valid use cases. With a single request document, XACML produces a single response mandating either "permit" or "deny"— thereby restricting processing based on individual responses of individual records within the client's request.

- **Approach 3: Multithreaded thin requests to XACML Engine**
  This approach makes many requests to the XACML rule engine for each distinct record in the client's request in parallel processes. Using a Thread Pool executor of Java 1.5 package, multiple threads can be spawned for a bunch of records for the client's requests. This approach gives a reasonable degree of performance enhancement but still suffers from a number of functional, performance and technical issues.

  Instances where a single denial of any record in the client's request should deny the entire request create a challenge around breaking the processing of threads that run largely asynchronously. In and of itself, producing multithreading can be technically challenging. That is especially true when it is to be achieved in a web container wherein the servlet specifications have discouraged use-defined threads because it would interfere with thread management of the web container. In addition, there would still be multiple network calls depending upon the number of threads configured in the thread pool.

- **Approach 4: Single-threaded, single thin request to XACML engine using middleware**
  This approach tries to achieve optimal performance by introducing middleware between the client and the XACML engine. The middleware acts as the mechanism to transfer functional requests and responses while limiting actual invocation to the engine to only once. The request details could be written to middleware, and a reference of the data written on the middleware could be passed to the XACML engine. The XACML PEP would pass the reference to the PDP, which, in turn, would invoke PAP to get a relevant policy. In this case, the policy would simply authorize the request in a coarse-grained way by validating whether or not the user is allowed to perform a business function. For fine-grained security at the individual data level, the policy would include obligations containing a way to read data from the middleware. The obligations processing could be done to update the response to each and every record at the same middleware. In the end, a single response could be passed back to the client containing reference of the updated middleware date.

  After receiving the response, the client could read the data from the middleware using the reference and could undertake a subsequent decision about whether or not to continue processing the request.

CONCLUSIONS & RECOMMENDATIONS

The case study describes a step-wise approach to decoupling authorization needs of an enterprise application using XACML technology. Because it is based on XML, XACML is an obvious choice to create a centralized environment for hosting all enterprise policies.

Key conclusions and recommendations are summarized below:

- Authorization decisions must be decoupled from the core application architecture for centralized control of enterprise application security.
- The decoupled architecture should support easy integration with multiple discrete platforms and technologies.
- XACML provides a simple and powerful mechanism to implement a centralized security control system.
- The XML-based XACML can be easily "plugged in" to multiple interfaces.
- It is important to ensure that all implicit and explicit authorization rules are specified in the XACML policy document, which should be accessed each time an authorization decision must be made during a client request.
- Policies are difficult to specify and maintain; one approach is to break policies into coarse-grained and fine-grained levels of security.
- Fine-grained security offers further segregation for read and update operations.
- For bulk updates, implementation should be carefully examined for performance.
- An optimal approach for bulk authorizations is to use middleware to write all the client requests and ask the obligation service through PEP to access the client records from that middleware.

REFERENCES

- http://en.wikipedia.org/wiki/XACML
- http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
- http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- http://sunxacml.sourceforge.net/
- http://www.ibm.com/developerworks/xml/library/x-xacml/

About Sapient Global Markets

Sapient Global Markets, a division of Sapient® (NASDAQ: SAPE), is a leading provider of services to today's evolving financial and commodity markets. We provide a full range of capabilities to help our clients grow and enhance their businesses, create robust and transparent infrastructure, manage operating costs, and foster innovation throughout their organizations. We offer services across Advisory, Analytics, Technology, and Process, as well as unique methodologies in program management, technology development, and process outsourcing. Sapient Global Markets operates in key financial and commodity centers worldwide, including Boston, Chicago, Houston, New York, Calgary, Toronto, London, Amsterdam, Düsseldorf, Geneva, Munich, Zurich, and Singapore, as well as in large technology development and operations outsourcing centers in Bangalore, Delhi, and Noida, India.

For more information, visit www.sapientglobalmarkets.com.

Global Offices

Amsterdam
Prins Bernhardplein 200
1097 JB Amsterdam
Netherlands
Tel: +31 (0) 20 330 0011

Bangalore
Salarpuria GR Tech Park
6th Floor, "VAYU" Block
#137, Bengaluru 560066
Karnataka
India
Tel: +91 (080) 410 47 000

Boston
131 Dartmouth Street
3rd Floor
Boston, MA 02116
Tel: +1 (617) 621 0200

Calgary
888 3rd Street SW
Suite 1000
Calgary, Alberta T2P 5C5
Canada
Tel: +1 (403) 444 5574

Chicago
30 West Monroe,
12th floor
Chicago, IL 60603
Tel: +1 (312) 458 1800

Delhi
Towers D & E,
DLF Cyber Greens
DLF Phase III
Sector 25-A
Gurgaon 122002
Haryana
India
Tel: +91 (124) 416 7000

Düsseldorf
Speditionstrasse 21
40221 Düsseldorf
Germany
Tel: +49 (0) 211 540 34 0

Geneva
Succursale Genève
c/o Florence Thiébaud, avocate
rue du Cendrier 15
1201 Geneva
Switzerland
Tel: +41 (0) 58 206 06 00

Houston
Heritage Plaza
1111 Bagby Street Suite 1950
Houston, TX 77002
Tel: +1 (713) 493 6880

London
Eden House
8 Spital Square
London, E1 6DU
United Kingdom
Tel: + 44 (0) 207 786 4500

Los Angeles
1601 Cloverfield Blvd.
Suite 400 South
Santa Monica, CA 90404
Tel: +1 (310) 264 6900

Munich
Arnulfstrasse 60
80335 München
Germany
Tel: +49 (0) 89 552 987 0

New York
40 Fulton Street
22nd Floor
New York, NY 10038
Tel: +1 (212) 206 1005

Singapore
Air View Building #02-01
2 Peck Seah Street
Singapore 079305
Republic of Singapore
Tel: +65 (3) 104 5308

Toronto
129 Spadina Avenue
Suite 500
Toronto, Ontario M5V 2L3
Canada
Tel: +1 (416) 645 1500

Zürich
Seefeldstrasse 35
8008 Zürich
Switzerland
Tel: +41 (58) 206 06 00

Sapient® Global Markets