XACML Policy Enforcement

Under Construction

This page is in the process of being composed. – DWD

Unable to render {include} The included page could not be found.

1. Introduction

A major feature of the Fedora security architecture is the eXtensible Access Control Markup Language (XACML) and an XACML-based policy enforcement module. Developed by the OASIS Consortium, XACML is an XML-based markup language to encode access control policies. The policy language is flexible and enables the specification of fine-grained, machine-readable policies that can be used to control access to Fedora web services, Fedora digital objects, datastreams, disseminations, and more. Since a policy is only worth its salt if it can be enforced, Fedora 2.1 introduces a new Authorization module implemented as part of the core Fedora repository service. The Authorization module is built upon the Sun XACML engine. Each XACML policy defines: (1) a "target" describes what the policy applies to (by referring to attributes of users, operations, objects, datastreams, dates, and more), and (2) one or more "rules" to permit or deny access. There is a Fedora-specific policy vocabulary for referring Fedora operations and Fedora-specific entities within XACML policies. Regarding policies, Fedora supports both repository-wide policies (that specify broad access controls that apply to the entire repository), and object-specific policies (that specify rules for a single object, and can even be stored within the object in a special datastream). Fedora 2.1 comes out-of-the-box with a set of default repository-wide policies that establish baseline access controls that are equivalent to what was provided in Fedora 2.0. These default policies restrict access to the Fedora management web service (API-M) to only the Fedora administrator, permit open access to the Fedora access web service (API-A), and establish some other basic controls (e.g., allow access to API-M only from localhost; restrict policy management to administrator). In addition to the default policies (which can be modified), any number of custom XACML policies can be written and loaded into Fedora. For assistance in creating new policies for your repository and for your objects, see the Fedora XACML Policy Writing Guide.

This guide is intended to give a general overview of XACML-based Policy Enforcement Module to support authorization in Fedora repositories. This guide provides Instructions on how to configure XACML-based policy authorization in Fedora as well as a discussion of how the Fedora Policy Enforcement module works. ** For more information consult the following sources:

Fedora Security Architecture

Securing Your Fedora Repository: documentation on security options and configuring user authentication sources for Fedora repositories

Fedora XACML Policy Writing Guide: documentation for details on writing XACML policies for Fedora repositories

OASIS (for policy writers)

OASIS XACML Specification: this is the official specification and a good reference document

A Brief Introduction to XACML: this is nice introduction to the XACML concepts

XACML Technical Committee: this home page of the technical committee provides access to other documents on XACML

SUN (for developers)

Sun XACML Home Page: general information on the Sun Java reference implementation of XACML

Sun XACML Programmer Guide: technical details of the reference implementation

Sun XACML Javadocs: interface definitions of the reference implementation

2. Configuring the XACML Authorization Module

Note that to do identity-based policies (user login id or user attributes), you must have authentication configured. Run fedora-setup to choose which Fedora service interfaces will be configured for authentication. If a policy is written that makes reference to required (MustBePresent="true", either explicitly or by default) user identity or attributes (Subject attributes in Policy target) and authentication has not been configured, the policy will be evaluated as "indeterminate" and the service request will fail with an authorization exception. Be sure to enable authentication for API-A if you intend to write policies for accessing objects based on user identity/attributes.

By default, Fedora XACML-based authorization is enabled. Configuration of the Fedora XACML-based Policy Enforcement Module is done in the Fedora server configuration file (fedora.fcfg). Depicted below is the section of the configuration file for the Authorization module that controls XACML-based policy enforcement.

```
<module role="fedora.server.security.Authorization"
        class="fedora.server.security.DefaultAuthorization">
    <comment>Builds and manages Fedora's authorization structure.</comment>
    <param name="REPOSITORY-POLICIES-DIRECTORY"
           value="/fedora-xacml-policies/repository-policies"/>
    <param name="XACML-COMBINING-ALGORITHM"
           value="com.sun.xacml.combine.OrderedDenyOverridesPolicyAlg"/>
    <param name="ENFORCE-MODE" value="enforce-policies"/>
    <param name="POLICY-SCHEMA-PATH" value="xsd/cs-xacml-schema-policy-01.xsd"/>
    <param name="VALIDATE-REPOSITORY-POLICIES" value="true"/>
    <param name="VALIDATE-OBJECT-POLICIES-FROM-DATASTREAM" value="false"/>
    <param name="OWNER-ID-SEPARATOR" value=","/>
</module>
```

2.1 Enabling/Disabling XACML Policy Enforcement

To enable/disable XACML policy enforcement in Fedora, use the Fedora configuration file (fedora.fcfg). Whether Fedora uses XACML for authorization decisions is controlled by the `ENFORCE-MODE` parameter in the Authorization module:

```
<param name="ENFORCE-MODE" value="enforce-policies"/>
```

The `ENFORCE-MODE` parameter can contain one of three values, with the following meanings:

1. enforce-policies – enable XACML enforcement to determine whether a request is permitted or denied
2. permit-all-requests – disable XACML enforcement; PERMIT every request by default
3. deny-all-requests – disable XACML enforcement; DENY every request by default

The `enforce-policies` setting is used to enable the enforcement of XACML policies, and is the default setting for a Fedora repository. The `permit-all-requests` setting can facilitate testing code independent of security. The `deny-all-requests` setting can be used to quickly shut down access to the server, but requires a server restart to affect this.

Tomcat container security is, of course, still a first barrier to authentication/authorization (i.e., Fedora's Tomcat web.xml specifies access protection earlier than XACML. Tomcat container security is always in place regardless of the setting for parameter `ENFORCE-MODE`.

2.2 Configuring the Storage Locations of Policies

Active policies are those policies that are "in play" for the repository. There are two places that active policies can be stored: (a) in a file system location configured for the repository, or (b) in digital objects within the special "POLICY" datastream. It should be noted that Fedora comes out-of-the-box with a set of default repository-wide policies. In addition to these default policies, it is possible to create any number of custom XACML policies for a repository and for specific digital objects. Consult the Fedora XACML Policy Writing Guide for instructions on authoring new policies and a set of sample policies for Fedora.

2.2.1 Storing Repository-Wide Policies in a Configured Location

Repository-wide policies are broad policies that are intended to be in play for the entire Fedora repository. By saying that these policies are *broad* does not mean they must be course-grained. Repository-wide policies can be fine-grained and they can be written to control access to any Fedora API operation, to groups of digital objects, or even to sets of specifically identified digital objects. Repository-wide policies are distinguished from object-specific policies (described below) in that ***they are stored in memory when the server is started and they are evaluated for their applicability on every Fedora service request***.

In the Authorization module section of the Fedora server configuration file (fedora.fcfg), there are parameters to set the storage location for active repository-wide policies. ***The Fedora repository will only know about policies that are stored within the configured policy directory location***. Policies stored in any other location will be invisible to the repository and will not be loaded into the repository upon server startup! The storage location for active repository-wide policies can be set by the repository administrator using the follow parameter in Authorization module (in fedora.fcfg):

```
<param name="REPOSITORY-POLICIES-DIRECTORY" value="/fedora-xacml-policies/repository-policies"/>
```

The configuration parameters in the Authorization module provide the repository administrator with a choice as to where XACML repository-wide policy files are stored. If you keep the default path value, then the repository-wide policy directory will be created at: FEDORA_HOME/data/fedora-xacml-policies /repository-policies. You can override this default with either a fully-qualified directory path or a relative directory path of your choice. The policy storage location work just like the digital object and datastream storage locations that are also configured for the server (in fedora.fcfg). It is expected that repository administrators appropriately protect these directories so that they cannot be tampered with.

Default Repository-Wide Policies:

It should be noted that Fedora comes out-of-the-box with a set of default repository-wide policies. The default policies are automatically copied into the active policies storage location the first time the Fedora repository server is started (i.e., copied into a subdirectory named "/default"). It is recommended that changes to the default repository-wide policies are made with extreme care. These policies establish the baseline authorization rules for Fedora. Refer to the documentation below on default repository policies for a description of what each policy does.

NOTE! Any changes to the default policies or your own custom policies should be made in the /default subdirectory of the configured location for repository-wide policies. The original versions of the default policies are also maintained in the internal fedora staging directory named "FEDORA_HOME /server/fedora-internal-use/" from which the default policies are copied into the configured location. ***Do not edit the policies that are stored in this internal staging directory!*** They are your record of how the default policies shipped with Fedora.

Custom Repository-Wide Policies:

You can create your own custom repository-wide policies and put them in the repository-wide policy directory. You can also create your own subdirectories under the repository-wide directory to organize your policies. It is recommended that you do NOT put your custom policies in the /default subdirectory that was created by the Fedora server. This location is reserved for the policies that are distributed with Fedora. Consult the Fedora XACML Policy Writing Guide for instructions on authoring new custom policies and to view a set of sample policies for Fedora.

Custom repository-wide policies can be written to deny/permit access to Fedora API operations, to control access to groups of objects, or to control access to objects with certain attributes under certain conditions. It should be noted that custom repository-wide policies can be also written to address a specific digital objects (identified by PID). This is in contrast to storing or referencing an object-specific policy inside the digital object's POLICY datastream (described below). The disadvantage of putting an object-specific policy in the repository-policies storage location is that the policy is unnecessarily evaluated for requests that do not pertain to the specific digital object it is about. This is because repository-wide policies are stored in memory and are evaluated for every Fedora API request. As described below, there are better places to store a policy that pertains to a single digital object (or a small set of named digital objects).

2.2.2 Storing Object-Specific Policies in a POLICY Datastream

An object-specific policy can be stored inside a digital object within the special reserved datastream whose ID is "POLICY". There are several benefits to storing object-specific policies in the POLICY datastream. First, these policies are only evaluated for those Fedora API requests that pertain to the digital object in which the policy "resides." Policies that are stored in the POLICY datastream are not loaded into memory upon startup of the Fedora server (as are the repository-wide policies); therefore, the POLICY datastream approach may be a way to enhance performance in cases where a large number of object-specific policies will exist. Another benefit is that the POLICY datastream storage strategy may provide for easier distribution of policy management responsibilities For example, authors or owners of particular digital objects can be granted the rights to view and modify the POLICY datastream of their objects, without having to obtain repository administrator privileges to modify policies in a configured policy storage locations external to a repository.

Unlike when an object-specific policy is placed in the repository-wide policies directory, it is not necessary to specify the an object PID in the target of the XACML policy target when the policy resides in a POLICY datastream. The Fedora system will automatically make this association during policy enforcement.

Policies as stored as Inline-XML (X) or Managed-Content (M) Datastreams: One benefit of putting an object-specific policy inside a digital object as a type X or M datastream is that the policy is stored within the either in the object's XML wrapper file (i.e., FOXML), or it is stored in the managed content datastream area of the repository. In either case, the POLICY file itself is under the direct custodianship of the Fedora repository. As such, these policies are managed like any other X or M datastreams, and the policies become ***portable*** with their parent digital objects (e.g., the polcies goes with with an object, as datastream content, when the object is exported from the repository).

Policies as Externally Referenced (E) or Redirected (R) Datastreams: One benefit of putting object-specific policies in the POLICY datastream in a by-reference fashion is that multiple digital objects can point to the same policy. In this case, you probably do not want to have the target of the XACML policy make reference particular digital object PIDs. In this scenario, you can store the policy external to the digital objects to which it pertains, and store the URL of the external policy location in the E or R type datastream. Fedora resolves such URLs at runtime, so when the policy enforcement module needs to evaluate the XACML policy, Fedora will automatically retrieve the policy file from its external location and pass it on the the policy enforcement module for evaluation.

2.3 Activating and Loading Repository-Wide Policies

Repository-wide policies are not considered ***active*** unless they are placed in the configured storage location specified in the Authorization module configuration (see fedora.fcfg). Once policies are placed either directly in this directory, or within subdirectories under it, they are able to be ***loaded*** by the Fedora repository server. To put new custom policies into play, simply add them to the configured repository-wide directory, or subdirectories within it. To inactivate a policy, remove it from the directory. To modify an existing policies, edit the policy in the configured policy directories (or preferably, edit in another location and replace the existing policy in the configured policy directory).

To activate and load repository-wide policies take the following steps:

1. **Validate** your policies using the validate-policy command line utility *(optional, but recommended!)*
2. **Activate** your policies by copying them into the configured storage location for repository-wide policies (configured in the Authorization module in fedora.fcfg)
    - REPOSITORY-POLICIES-DIRECTORY – put your custom repository-wide policies in any sub-directory under this configured directory
3. **Load** your policies by starting the Fedora server. If the server is already started, run the fedora-reload-policies command line utility.

All repository-wide policies are loaded into the Fedora server's memory every time the server is started. For obvious performance reasons, object-specific policies stored in the POLICY datastreams of digital objects are not pre-loaded; instead they are loaded on an as-needed basis (i.e., when a request for that object is made).

WARNING: It should be noted that the _**first time** _ the Fedora server is started, the _**default** _ repository-wide policies are copied into a subdirectory named "/default" under the repository-wide policy storage location. To make sure this happens correctly, you should initially start the server with the default value for the server hostname (see "fedoraServerHost" in fedora.fcfg). The default is 127.0.0.1 (or "localhost"). This initial starting of the server as "localhost" is necessary to make sure that the default XACML policies are properly configured and placed in the repository-wide policy directory. Once the default policies are in place, you can shutdown the server and change the hostname at any time. If you subsequently change the hostname to something other than "localhost" you must manually update some of the default policies to reflect the fixed IP address of your server. In the following default policies, you will see the loopback IP address of 127.0.0.1 (and ::1 for IPv6) in the policy condition:

1. deny-apim-if-not-localhost.xml
2. deny-reloadPolicies-if-not-localhost.xml
3. deny-serverShutdown-if-not-localhost.xml

Simply add the new IP address to the policy rule as follows:

```
<Rule RuleId="1" Effect="Deny">
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
        <EnvironmentAttributeDesignator
          AttributeId="urn:fedora:names:fedora:2.1:environment:httpRequest:clientIpAddress"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">::1</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">111.22.333.4</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
```

2.4 Enabling/Disabling Policy Validation

There are several parameters in the Authorization module configuration that control whether Fedora will attempt to validate policies against the XML schema for XACML.

```
<param name="POLICY-SCHEMA-PATH" value="xsd/cs-xacml-schema-policy-01.xsd"
<param name="VALIDATE-REPOSITORY-POLICIES" value="true"/>
<param name="VALIDATE-OBJECT-POLICIES-FROM-DATASTREAM" value="false"/>
<param name="VALIDATE-SURROGATE-POLICIES" value="false"/>
```

By default all repository-wide policies will be validated by the repository when they are loaded. In general it is recommended that all new policies are validated before they are put into the active policy location. Use the validate-policy command line utility to validate an XACML policy file. A policy failing validation would result in the following situation, according to the stated case.

- repository policy: authorization module failure at server startup time, server not properly started

Object policies are not set to validate by default because they are loaded dynamically at each use, and neither testing to prove adequate performance nor caching to ensure it has been done. This policy validation check can be enabled by the appropriate fedora.fcfg setting.

2.5 Configuring Multiple Owner IDs

The `OWNER-ID-SEPARATOR` value specifies the delimiter used between multiple values within the ownerId property of the objects stored in the repository. It is specified as a regular expression, and the default value is ',' (comma). This delimiter allows the use of policies that act on objects with multiple possible ownerId values. For more information, see the sample policies in the Fedora XACML Policy Writing Guide.

2.6 Surrogate Feature No Longer Requires Policies.

This is an ***experimental feature*** continued with Fedora 2.2 that enables a higher level application or service (e.g., a Web front end, middleware, or other web service) to authenticate to Fedora with its own identity, but pass along the identity of an end-user that authenticated with the service. The higher level application or service is thus acting as a "surrogate" for the ultimate end user. The surrogate user can pass through to Fedora the identity of the end-user it is representing in the HTTP request header (using the From: attribute). If Fedora authenticates the identity of the service-as-surrogate, then the identity of the ultimate end user can be used in XACML policies.

Refer to the Authentication and User Attributes section of the Securing Your Fedora Repository for information on configuring servlet filters in support of the surrogate feature.

3. Implementation of the Fedora Policy Enforcement Module

3.1 Policy Determination Point (PDP) and Policy Enforcement Point (PEP)

According to the OASIS XACML specification, an application functions in the role of the Policy Enforcement Point (PEP) if it guards access to a set of resources and asks the Policy Determination Point (PDP) for an authorization decision. The PEP MUST abide by the authorization decision in the following way: A PEP SHALL allow access to the resource only if a valid XACML response of Permit is returned by the PDP. The PEP SHALL deny access to the resource in all other cases. An XACML response of Permit SHALL be considered valid only if the PEP understands all of the obligations contained in the response.

The Fedora Policy Enforcement Module fulfills the responsibilities of both the PEP, except that for 2.1, obligations are ignored. (But neither are any coded in the default policies.) The Fedora module wraps the Sun XACML implementation of the PDP. Fedora module also implements custom attribute finders and a custom PEP.

The PDP determines the set of policies that are applicable to any given Fedora service request. Remember the PDP determines whether a policy is applicable by comparing the Subject/Resource/Action attribute designations in a Policy Target to the context of an incoming Fedora service request (i.e., attributes that describe the Fedora service request, the user/subject, the desired object/datastream/dissemination, and the runtime environment). Repository-wide policies are always in play and will be evaluated by the PDP to determine whether they are applicable to the particular incoming Fedora service request. Object-specific policies if the incoming request refers to an object by its PID and there exists an object-specific policy mentioning that PID. ***PDP makes the decision of deny/permit/indeterminate, and then the PEP makes sure to enforce this decision for the incoming Fedora API request.***

NOTE: To quiet the Sun XACML engine's INFO messages, a logging.properties file is included with the Fedora distribution, edited to print only SEVERE messages to the console. Fedora then starts Java using that logging.properties file. The original file, as distributed with JDK 1.4.2.8, prints INFO (and more severe) messages, making the console a bit chatty. To revert to the original logging behavior, edit fedora-start.bat (Windows) or fedora.sh (Unix), and remove the logging.properties from the java vm arguments (see -D flags).

3.2 Understanding the XACML Policy Combining Algorithm

Policy writers must understand the interaction effect of multiple XACML policies that are in scope for any particular action. The Fedora configuration file (fedora.fcfg) sets the policy combining algorithm that will be used by the Policy Enforcement Module in evaluating sets of policies. In the the Fedora server configuration file (fedora.fcfg) note the following parameter in the section for the Authorization module::

```
<param name="XACML-COMBINING-ALGORITHM" value="com.sun.xacml.combine.OrderedDenyOverridesPolicyAlg"/>
```

This parameter sets the XACML policy combining algorithm that controls how the Fedora Policy Enforcement Module will deal with multiple policies that may be applicable to a Fedora service request. The default value in Fedora is the Ordered Deny Overrides policy combining algorithm. It allows a single evaluation of ***deny to take precedence*** over any number of permit, not applicable or indeterminate results. Note that this uses the regular Deny Overrides implementation since it is also ordered. Consult the OASIS and Sun XACML documentation for a description of alternative combining algorithms. Note that the default policies assume and require this algorithm. Generally policies are developed with an algorithm in mind.

In the PDP, policies are matched based on their applicability to an incoming service request. All policies that are applicable are combined programmatically and dynamically per request into a PolicySet. To estimate the number of policies that may be in the PolicySet for a given service request, consider N to be the number of policies configured in Fedora repository-wide policy storage location. Then we have the following possible number of policies in a PolicySet for the PDP to consider:

N :	# of policies if a service request ***does not refer to a particular digital object***
N :	# of policies if a service request ***refers to an object, but there is no object-specific policy for that object***
N+1:	# of policies if a service request ***refers to an object that has an object-specific policy datastream, but has no object-specific policy in the object policies directory***
N+1:	# of policies if a service request ***refers to an object that has an object-specific policy in the object policies directory, but has no object-specific policy datastream***
N+2:	# of policies if a service request ***refers to an object that has an object-specific policy datastream and a policy in the object-policies directory***

3.3 A Simplified Understanding of the Authorization Decision

For an incoming service request to succeed, there must be an explicit permit and the absence of a deny; the absence of a deny is not enough to permit an action. By default, if any of the applicable policies in a Policy Set yield a deny, the requesting subject will be denied access, even if some other policy permitted the action. In other words, deny will prevail over permit. Also, if there is a policy in the set that is evaluated as "Indeterminate," then the result of that policy evaluation will be considered a deny. A policy can be evaluated as Indeterminate if there was an error during policy evaluation. Also, a policy can be evaluated as Indeterminate if there is a required attribute specified in the policy that did not exist in the context of the incoming requests. See the section of Required vs. Optional Attributes for more details.

For the purposes of a simple understanding of the Fedora Policy Enforcement module, things work like this:

NOTE:	*=0 means NO policy in a policy set evaluated to that result*
	=1 means one or more policies in a policy set evaluated to that result

========= POLICY SET RESULTS =======	=== FINAL DECISION ===
DENY=0 INDETERMINATE=0 PERMIT=0	result is DENY
DENY=1 INDETERMINATE=0 PERMIT=0	result is DENY
DENY=1 INDETERMINATE=0 PERMIT=1	result is DENY (denial trumps permit)
DENY=0 INDETERMINATE=0 PERMIT=1	result is **PERMIT**
DENY=0 INDETERMINATE=1 PERMIT=1	result is DENY (indeterminate is treated as denial which trumps permit)
DENY=0 INDETERMINATE=1 PERMIT=0	result is DENY

Put another way... assuming the default policy combining algorithm for Fedora is "Deny Overrides", an action is Permitted or Denied depending on the evaluation of the various policies, as follows:

Permit requires ALL of the following conditions to be TRUE:

- at least one policy was evaluated to Permit the action
- NO policy must evaluate to explicitly Deny the action
- NO policy must evaluate as Indeterminate for the action
- NO error or unknown result is returned by the Sun XACML engine

Deny only requires ONE of the following conditions to be TRUE:

- at least one policy was evaluated to explicitly Deny the action
- at least one policy was evaluated to be Indeterminate for the action
- the Sun XACML engine returned a unknown result (an error or a return value that is not in the XACML specification)

3.4 PDP Implementation Details

We can understand the results of the PDP's evaluation of a policy set from three perspectives: (1) the Sun XACML engine, (2) the Fedora wrapper of Sun XACML, and (3) the bottom line outcome.

(1) PDP (Sun XACML engine perspective):

The Sun XACML engine, which underlies the Fedora Policy Enforcement Module, will evaluate a Policy Set and return a decision. Refer to the OASIS XACML Specification and the Sun XACML documentation for details.

In making its authorization decision, the Sun XACML engine will return a single result from its evaluation of a Policy. The result will be one of the following:

- **Permit** – returned if a policy rule was applicable and thus it returned its permit effect.
- **Deny** – returned if a policy rule was applicable and thus returned its denial effect.
- **Indeterminate** – returned if an attribute value that was needed to evaluate a rule could not be found, or another error prevented processing.
- **NotApplicable** – returned if no rule applied and so no effect could be returned.

Given the default policy combining algorithm of "Ordered Deny Overrides," the PDP will make its final decision for a policy set such that * DENY* will prevail over PERMIT. If one or more policies evaluate to Deny, Sun XACML gives a verdict of DENY. If one or more policies evaluate to Indeterminate, and no policies evaluate to Deny, the verdict is INDETERMINATE. If one or more policies evaluate to Permit, and no policies evaluate to either Deny or Indeterminate, the verdict is PERMIT. If one or more policies evaluate to NotApplicable, and no policies evaluate to Permit or Deny or Indeterminate, the verdict is NOTAPPLICABLE.

(2) PEP (Fedora wrapper perspective):

The Fedora wrapper respects the verdict of the Sun XACML PDP, which should usually be PERMIT or DENY, given our default set of policies. But, for safety, the Fedora wrapper code imposes a DENY result in any of several extraordinary cases: (1) if somehow Sun XACML returned an unexpected final result of Indeterminate or NotApplicable, perhaps due to a policy written incorrectly, (2) if Sun XACML returned no result at all, (3) if Sun XACML returned a result which is not defined by OASIS XACML or Sun XACML standards (the sunxacml Java interface uses int to code the results, so sunxacml could return bad results), or (4) if authorization processing results in a exception being thrown.

3) Combined (Bottom line perspective):

As a rule, a policy set evaluates to PERMIT when at least one policy in the set evaluates to Permit and no policies evaluating to Deny or Indeterminate. Otherwise, the policy set evaluates to DENY.

Specifically:

a) DENY occurs when there is at least one policy that evaluates to Deny or an Indeterminate.

b) DENY occurs if no policy evaluates to Permit. A DENY also occurs in several exceptional situations:

a) DENY occurs when no policies were found to be applicable (all evaluate to NotApplicable) or the related case of there being no policies at all configured with Fedora

b) DENY occurs when errors occurred during authorization processing, including no or bad results obtained.

3.5 Fedora PEP Implementation Details

Fedora's Policy Enforcement Point (PEP) builds a minimal request for the Sun XACML engine to evaluate. One job of the Fedora PEP is to gather up all of the Subject/Resource/Action/Environment attributes that are relevant for an incoming service requests. The values of this attributes are the key to determining what policies are in scope for an incoming service request. To gather up all relevant attributes, the Fedora PEP has two custom "attribute finder" modules that interact with the Sun XACML engine.

- The **ContextAttributeFinderModule** has the job of obtaining attributes that are stored in the enhanced Fedora Context object that is associated with an incoming Fedora service request. Attributes that originate with an incoming service requests can be Subject attributes (i.e., attributes of the requesting user/agent), Action attributes (i.e., the identity of the Fedora API operation that is the basis of the request), Resource attributes (i.e., attributes that identify specific objects/datastreams/disseminations that are being requested, and Environment attributes (i.e., attributes that describe the runtime environment of the incoming request, like the current date/time or the server IP address). It will honor a callback, even for an attribute which hasn't been explicitly coded, so can provide arbitrary attributes (e.g., from LDAP lookup in a JAAS login module, from Shibboleth via a servlet filter). [There are a few attributes which it explicitly doesn't serve, to prevent stack overflow on improper recursion, or because the attributes are known to be provided in the xacml request itself.]

- The **ResourceAttributeFinderModule** has the job of obtaining all relevant attributes about Fedora resources (i.e., digital objects, datastreams, and disseminations) for resources that are in scope for an incoming request. While a few attributes of a resource are picked up from the incoming request itself (typically, the identity of a resource like a PID or a datastream Id), the ResourceAttributeFinderModule gets all other attributes about such resources by introspecting on actual digital objects in the repository.

- The ContextAttributeFinderModule supplies environment values, named into the Fedora XACML URN namespace. An **EnvironmentAttributeFinderModule** may be added for full compliance with XACML requirement that environment attributes be serviced which are named into its namespace XACML's namespace itself.

4. How to Bind User Attributes into the Fedora Policy Enforcement Module

The availability of user identity and attributes depends on the authentication configuration option selected for the repository. Refer to the Authentication and User Attributes guide for information on authentication configuration options. In terms of understanding what user attributes can be referenced for Subjects in an XACML policy, you must first know what the sources of authentication information are for the repository. The Fedora XACML-based Policy Enforcement module will automatically be able to obtain attributes from one or more of following authentication sources when they are configured as described in Authentication and User Attributes.

4.1 fedora-users attributes

In Fedora, XACML policies can refer to user identity and attribute information that is specified within the fedora-users.xml file. This file is one of the Fedora configuration files and is created initially on running the installer. It contains username and password information about "authenticated" users. Attributes can also be assigned to user entries. These attributes are then available as attributes in related XACML policies.

By default, the fedora-users file contains the following users, passwords, and attributes:

```
<users>
  <user name="fedoraAdmin" password="fedoraAdmin">
    <attribute name="fedoraRole">
      <value>administrator</value>
    </attribute>
  </user>
  <user name="fedoraIntCallUser" password="changeme">
    <attribute name="fedoraRole">
      <value>fedoraInternalCall-1</value>
      <value>fedoraInternalCall-2</value>
    </attribute>
  </user>
</users>
```

Notice that each attribute has a name and can have multiple values. The cases of one or two values are shown; not shown here is that an attribute can have any number of values, including none, depending on your need. The second of these entries is for Fedora internal use, and should be left intact as installed. You will probably leave the first entry in place, also as-is. But if you change the default policies, you may require additional values of the *fedoraRol* attribute, or additional named attributes with their own values. Additional users (with unique names) can be added with attributes as needed. This is a convenient place to define repository managers, a small number of users, or a surrogate user. An example of an additional user follows:

```
<users>
  <user name="testuser1" password="testuser1">
    <attribute name="someAttribute">
      <value>xyz</value>
    </attribute>
    <attribute name="fedoraRole">
      <value>researcher</value>
    </attribute>
  </user>
</users>
```

The above example indicates that the user has two attributes (an attribute named* "someAttriibute"* whose value is *xyz*, and an attribute named* "fedoraRole"* whose value is *researcher*). These attribute names can be used in SubjectAttributeDesignator specifications in XACML policies and the values can be used in AttributeValue specifications in policies. Below is a snippet of an XACML policy that refers to attributes from fedora-users (refer to the Fedora XACML Policy Guide for more info on policy syntax):

```
<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="*fedoraRole*" MustBePresent="false"
                                  DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
</Subjects>
```

4.2 LDAP attributes

In Fedora, XACML policies can refer to user attribute names and values that are registered in an LDAP that is configured with Fedora's Ldap servlet filter. Refer to the Authentication and User Attributes section of the Securing Your Repository guide for information on servlet filter configuration options. Given that an LDAP is properly configured with Fedora, the Fedora XACML-based Policy Enforcement module will be able to access LDAP user attributes, which means that you can refer to LDAP attributes in reference to a Subject in a policy, as shown in the following XACML snippet (refer to the Fedora XACML Policy Guide for more info on policy syntax):

```
<Rule RuleId="1" Effect="Deny">
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
    <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="*ou*"/>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Lb-Info Technology
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Lb-Univ Librarian-General
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>
```

In the above example, the SubjectAttributeDesignator refers to an LDAP attribute name ("ou"), which refers to the university organizational department that a user belongs to. The policy rule applies a function to set up the condition that the value of the "ou" attribute must be one of the listed values ("Lb-Info Technology" or Lb-Univ Librarian-General").

4.3 Shibboleth attributes via an HTTP Servlet Filter

There are cases when an application may obtain authenticated user attributes in an application or service layer outside the context of the Fedora repository service. Fedora provides a simple means of getting these attributes into the Fedora repository service so they can be used by the Fedora XACML policy enforcement module. To support the ability for upstream applications or services to send these user attributes into a repository, Fedora will recognize a special HTTP servlet request attribute named after the value of static final String constant fedora.server.Context. FEDORA_AUX_SUBJECT_ATTRIBUTES.

The Fedora repository service will automatically look for an HTTP servlet request attribute named after this constant. The Fedora code now takes a request attribute found under that name, as a Map giving name and values of subject attributes. Currently, name must be a String and this is unlikely to change. Value must be a String, and later this might be relaxed to include String[], to allow attributes with multiple values. Other types of value are not serviced. The effect of having a key => value pair "a" => "b" in the Map is the same as having a Tomcat role "a=b", with the exception that the effect of having the same attribute key redefined both places is right now undefined. So your servlet filter needs only create the map and populate it, and put it into an HTTP servlet request as attribute named after fedora.server.Context.FEDORA_AUX_SUBJECT_ATTRIBUTES. Fedora will then look for it, and use the attributes in XACML-based authorization.

This approach was initially developed to support the OhioLink Shibboleth servlet filter that will be available to the Fedora community from the OhioLink implementation. Although full Shibboleth integration with Fedora will be in future releases of Fedora, the ability to send attributes into Fedora via a servlet filter is a way to get started with using Shibboleth-acquired attributes in Fedora XACML policy enforcement. It should be noted that this means of getting attributes into Fedora can be used with any subject attribute source (i.e., it remains source-neutral and doesn't favor Shibboleth or any other particular scheme).

5 Default Repository Policies for Fedora

5.1 Default Access Control Policies

Out-of-the-box, the Fedora repository will have a default set of access control policies that provide for a highly restricted management service (API-M), an open access service (API-A), and an open OAI provider service. The default access control policies establish the same level of out-of-the-box security on the repository that was previously configured for Fedora 2.0 release; however, as of Fedora 2.1 these basic access controls are now specified as XACML policies. The default access control policies can be found within the Fedora software distribution in the following directory:

FEDORA_HOME/data/fedora-xacml-policies/repository-policies/default

The first time the Fedora repository server is started, these policies will be automatically copied into the official repository-wide policy storage location that was specified in the Fedora configuration file (fedora.fcfg). The policies are activated once they are copied into this location.

Rule	Service	XACML Policy File	Policy Description
1	any	permit-anything-to-administrator.xml	This is a "positive policy" that permits the Fedora administrator to have access to any operation on any Fedora repository service (API-M, API-A, OAI, RISearch). By default the Fedora administrator is configured in the default Tomcat user credentials file (tomcat-users.xml).
2	API-M	deny-apim-if-not-localhost.xml	This is a "negative policy" that denies access to API-M operations that are not made from the IP address of the machine on which the Fedora repository is running on. In other words, the policy will not allow API-M requests from hosts other than "localhost."
3	API-A	permit-apia-unrestricted.xml	This is a "positive policy" that permits unrestricted access to API-A. In other words, API-A operations are completely open for use by any user/agent.
4	OAI	permit-oai-unrestricted.xml	This is a "positive policy" that permits unrestricted access to the default OAI provider interface to the Fedora repository. In other words, OAI-PMH operations are completely open for use by any user/agent. (Note, this does not control access to the stand-alone PROAI service that is distributed with Fedora 2.1. PROAI is a stand-alone web application that must be secured separately.

A review of how the policy combining algorithm works, will reveal that access to a service operation cannot occur unless access is ***expressly permitted.*** The net effect of the default access control policies is that the administrator is expressly permitted to do anything (with the restriction of having to make API-M requests from the same IP address that the server runs on), and all users are expressly permitted access to API-A and OAI service requests.

5.2 Default Utility Policies

Generally, the default repository utility policies ***should not be removed***. They enforce core and crucial protections of the repository. Considerate understanding of how they work should proceed any (unlikely) needed editing. For example, consider and edit to permit other IPs than localhost, as opposed simply to deleting the policy.

Rule	Service	XACML Policy File	Policy Description
1	serverAdmin	deny-policy-management-if-not-administrator.xml	
2	any	deny-inactive-or-deleted-disseminations-if-not-administrator.xml	This is a "negative policy" that will deny all access to inactive/deleted disseminations if the user/agent is not the Fedora administrator. Unlike purged disseminations, inactive/deleted disseminations still exist, but they are just marked as inactive/deleted. As such they should not be available to users. The exception is that the Fedora administrator is allowed to access them.
3	any	deny-inactive-or-deleted-objects-or-datastreams-if-not-administrator.xml	This is a "negative policy" that will deny all access to inactive/deleted datastreams if the user/agent is not the Fedora administrator. Unlike purged objects/datastreams, inactive/deleted objects/datastreams still exist, but they are just marked as inactive/deleted. As such they should not be available to users. The exception is that the Fedora administrator is allowed to access them.

4	API-M	deny-purge-datastream-if-active-or-inactive.xml	This is a "negative policy" that will ensure that datastreams cannot be purged (permanently removed) unless they are in the deleted state. Purging of active or inactive datastreams is not allowed.
5	API-M	deny-purge-object-if-active-or-inactive.xml	This is a "negative policy" that will ensure that objects cannot be purged (permanently removed) unless they are in the "deleted" state. Purging of active or inactive objects not allowed.
6	serverAdmin	deny-reloadPolicies-if-not-localhost.xml	This is a "negative policy" that will deny requests to reload policies (i.e., policy reactivation) if this requests is not initiated from the IP address of the machine on which the repository is running (i.e., localhost).
7	serverAdmin	deny-serverShutdown-if-not-localhost.xml	This is a "negative policy" that will deny requests to shutdown the Fedora server if this requests is not initiated from the IP address of the machine on which the repository is running (i.e., localhost).
9	serverAdmin	permit-serverStatus-unrestricted.xml	This is a "positive policy" that permits unrestricted access for obtaining the Fedora server status.

6 Sample Policies for Typical Fedora Use

The Fedora Policy Enforcement Module is intended to provide a flexible means of creating access control for a repository and for digital objects within a repository. Therefore, it is expected that each Fedora repository will have XACML policies appropriate for specific contexts and use cases. All repositories will start off, out-of-the-box, with the set of default repository policies. These policies set up a world where (1) users in the Fedora administrator role are permitted to do anything (see permit-anything-to-administrator.xml), (2) access to the API-M service is restricted to localhost (deny-apim-if-not-localhost.xml), and (3) the Fedora API-A service is totally unrestricted (see permit-apia-unrestricted.xml).

Given this out-of-the-box starting point, the perspective that can be taken to easily understand how to write new custom policies is:

- write new policies to *_tighten up _*access controls for the API-A service (i.e., start to selectively deny access)
- write new policies to*_ loosen up_* access controls for the API-M service (i.e., start to selectively permit access)

Please consult the XACML Policy Writing Guide which describes a reference collection of sample policies that would be useful for Fedora repositories. The collection contains examples of repository-wide and object-specific policies for restricting access to groups of digital objects based on certain attributes of the objects, for restricting access to certain kinds of datastreams and disseminations, for selectively permitting access to different API-M operations, and more. These policies can be found within the Fedora software distribution in the following directory: **FEDORA_HOME/userdocs/server/security/xacml-policies/examples**

Unable to render {include} The included page could not be found.