ViewDS Access Sentinel: Installation and Reference Guide

Published: March 2024
Version: 7.5.2

ViewDS Access Sentinel: Installation and Reference Guide

For ViewDS Directory release 7.5.2

March 2024

Document Lifecycle

ViewDS may occasionally update documentation between software releases. Therefore, please visit www.viewds.com to ensure you have the PDF with most recent publication date. The site also hosts the most recent version of this document in HTML format.

ABN 19 092 422 47

Contents

About this guide

This guide introduces Access Sentinel and the ViewDS implementation of XACML. It also includes how to install Access Sentinel, and how to write and manage XACML policy.

This section describes:

- Who should read this guide
- Related documents
- How this guide is organized

Who should read this guide

Read this guide if you need to install Access Sentinel and become familiar with writing and managing XACML policy for applications. Before using this guide, you should first read the 'System Overview' in the *ViewDS Directory: Installation and Operation Guide*.

Related documents

Other documents relating to Access Sentinel are:

- ViewDS Directory: Installation and Operation Guide
- ViewDS Directory Server: Technical Reference Guide
- ViewDS Access Presence: Technical Reference Guide
- ViewDS Access Proxy: Installation Guide
- ViewDS Management Agent in-application help
- ViewDS Authorization Policy Manager in-application help
- ViewDS Access Sentinel: Application Integration Kit for Java or .NET

How this guide is organized

This guide contains the following:

About this guide
Provides an overview of this guide.

About ViewDS Access Sentinel
Provides an overview of the ViewDS XACML framework and of Access Sentinel, along with an introduction to XACML.

Installing and configuring
Provides the instructions to install and configure Access Sentinel.

About XACML framework and policy

Provides information about Access Sentinel's implementation of XACML.

XACML tutorials

Provides the steps to define and apply an XACML policy to a resource.

XACML attributes provided by a PEP

Provides a technical reference for the XACML attributes provided by each Policy Enforcement Point (PEP).

Operational attributes

Provides a technical reference for Access Sentinel's operational attributes.

About ViewDS Access Sentinel

This chapter introduces the ViewDS XACML framework and Access Sentinel, and provides a brief overview of XACML (eXtensible Access Control Markup Language).

It describes the following:

What is Access Sentinel?

The **XACML framework** is part of ViewDS Directory. It allows you to apply the **XACML Access Control scheme** by defining XACML policy that controls access to the directory.

ViewDS **Access Sentinel** is an extension of the XACML framework that allows you to apply XACML policy to applications external to ViewDS Access Sentinel requires additional licencing beyond that of the core ViewDS Directory.

The XACML framework and Access Sentinel conform to the XACML Version 3.0 standard.

Why use XACML access controls?

The ViewDS XACML framework and Access Sentinel allow a fine-grained enterprise-wide approach to managing access-control policy across all of an organisation's applications and data sources.

Fine-grained access-control policy goes beyond previous models of access control. These policies not only control 'who can do what with which resources', but also control the why, when, where and how of entitlement.

Enterprise-wide access controls allow an organization to define, enforce, and audit their access-control policies. This is of increasing importance in the face of regulatory pressures and is discussed in more detail below.

Enterprise-wide access control

Traditionally, each application within an organisation has its own access-control mechanism. The access controls are therefore duplicated across applications and must be managed individually. As well as creating administrative inefficiencies, this approach also complicates the task of imposing enterprise-wide access-control policies.

An alternative is to remove access control from the applications and run it as a discrete service shared across many disparate applications.

This approach has many benefits:

- Consistent access-control policies can be applied to all applications and data sources
- Support and maintenance is streamlined
- Auditing and compliance are simplified

Additionally, enterprise-wide access control allows security to be managed more efficiently. The moment a policy is created or updated, it can be applied across all relevant applications. These applications become less complex and easier to maintain without their entitlement layer – a change to a security policy requires no modification to the application's code.

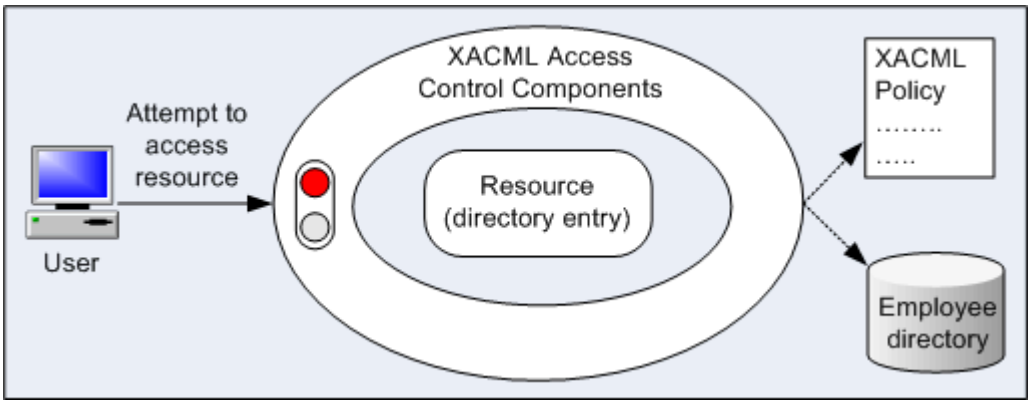
Brief introduction to XACML

XACML Version 3.0 is a standard that provides a framework for fine-grained, enterprise-wide access control. The standard describes two languages, both written in XML: an access-control *policy* language, and an access-control *decision* language.

The policy language describes access-control requirements by defining policies that describe, for example, who can access what and when. The decision language is used to form requests and responses. A request asks whether a given action by a given entity should be allowed; and a response provides the answer, which is determined according to an XACML policy.

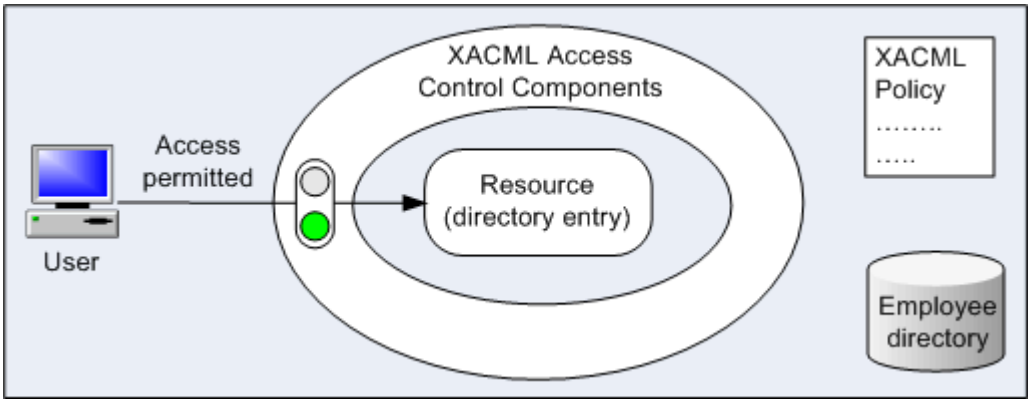
Simplified XACML implementation

The following illustrates a simplified XACML implementation.



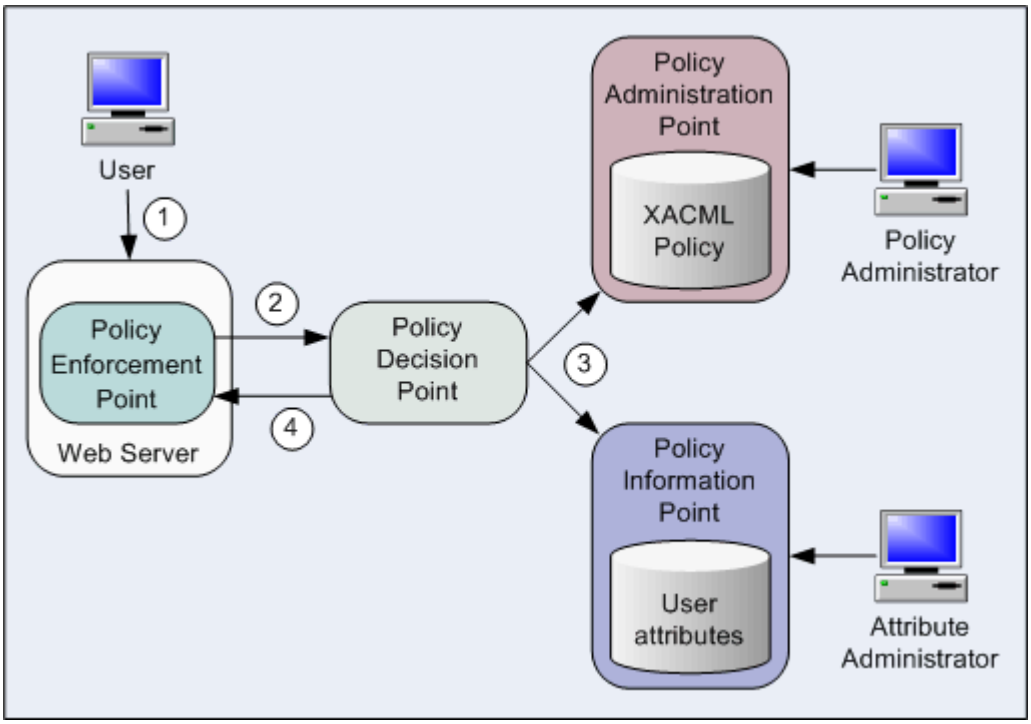
In the following illustration, a user attempts to view a document file protected by an XACML access-control implementation. The implementation determines whether the user should be permitted or denied access by interrogating the appropriate XACML policy.

The policy might include considerations such as the user's security level, department, role, position, location and the time of day. All combine to determine whether the user should be allowed access to the resource (as shown below).



XACML access control components

An implementation of XACML access control has four logical components (shown below).



The logical components are:

- Policy Enforcement Point (PEP)
  Protects a resource from unauthorized actions.
- Policy Decision Point (PDP)
  Determines whether access should be granted to a protected resource.
- Policy Administration Point (PAP)
  Allows policies to be created and stored in a repository.

- Policy Information Point (PIP)
  Stores additional information, such as user attributes, that can be used by the PDP to make access-control decisions.

In the illustrated example the resources protected by a Policy Enforcement Point (PEP) are the web pages available through a web server.

The steps shown in the illustration are as follows:

1. A user requests access to a web page.
2. The web server asks the Policy Enforcement Point (PEP) to send an 'authorization decision request' to the Policy Decision Point (PDP). The request includes *XACML attributes* that identify (among other things) the user, the resource they are attempting access, the action they are attempting to perform, and the environment (for example, date and time).
3. The Policy Decision Point (PDP) determines whether access should be permitted. It looks at the appropriate XACML policy in the Policy Administration Point (PAP), and the appropriate user attributes in the Policy Information Point (PIP). The information in the PIP allows the PDP to identify the user attempting to access the resource.
4. The PDP returns an 'authorization decision response' to the Policy Enforcement Point (PEP), which then acts on the decision to permit or deny access to the user.

Controlling access to the PIP and PAP

Many organizations implement an XACML solution with the intention to provide a single point for policy management and enforcement. However, most XACML solutions fail to meet this expectation because the PAP and PIP are accessed by users and require their own separate access controls.

Therefore, many XACML solutions introduce a requirement for three new, separate access-control systems: one for the PAP, a second for the PIP, and a third for the enterprise XACML access-control system.

An alternative, however, that avoids the complexity of this recursive hierarchy is to unify the PDP, PAP and PIP into a single policy server. This is the approach adopted by ViewDS and is discussed in the next section *Access Sentinel architecture*.

Repositories for the PIP and PAP

The repository for the Policy Information Point (PIP) is typically an existing LDAP directory because it usually already contains the organization's user attributes. However, as most directories cannot manage XML, the repository for the Policy Administration Point (PAP) is typically a relational database that supports XML.

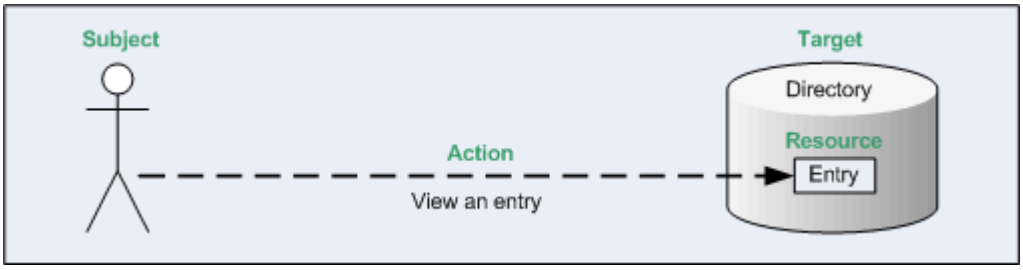
An improved approach that makes policy management and implementation more efficient is to store both PIP and PAP data in a single directory that fully supports XML. This makes the administrator's job much easier as they can search on the individual XML components within policy. Again, this approach is discussed in the next section *Access Sentinel architecture*.

XACML terms to remember

There are a couple of important XACML terms to remember:

- **Target** – the set of resources protected by the XACML policy (for example, a directory or a web site)
- **Resource** – the specific item (for example, an entry, attribute or value in the directory or a specific web page) within the target that the subject is attempting to access
- **Subject** – the user attempting to access a resource
- **Action** – the action attempted by the subject (e.g. view or modify an entry or web page)

These terms are illustrated below for XBAC where the target is the ViewDS directory and the resource is an individual directory entry.



Access Sentinel architecture

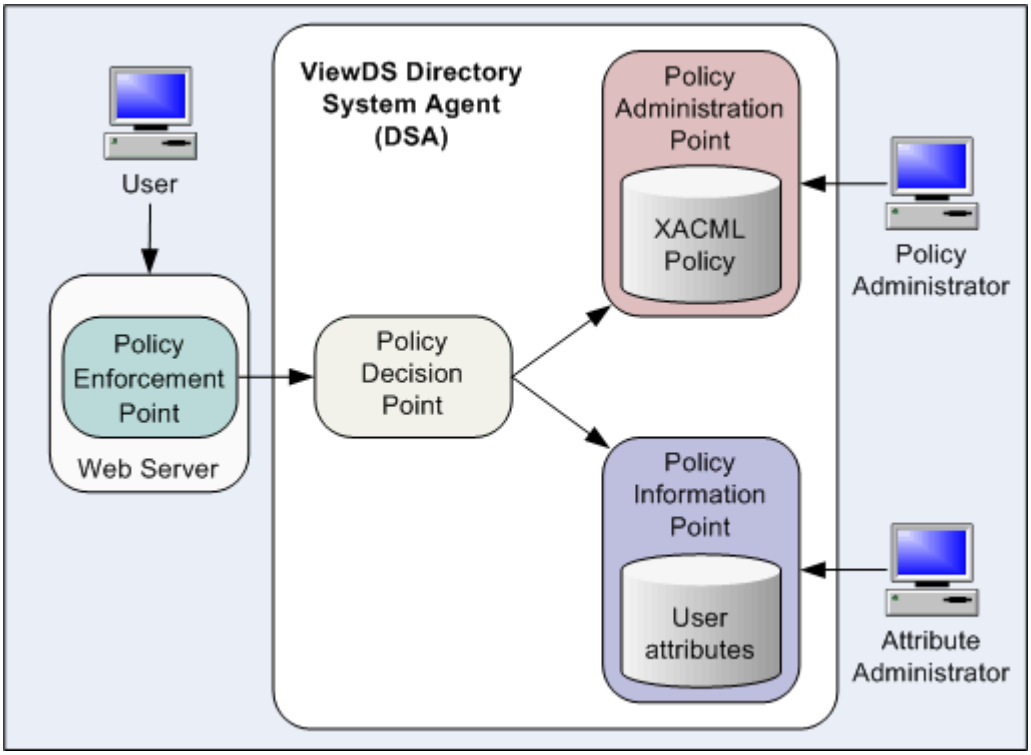
Access Sentinel extends the XACML framework that is installed as part of the Directory System Agent (DSA).

The XACML framework comprises a PDP that accepts authorization decision requests from an internal PEP, which protects the directory from unauthorized access. It also includes a PIP, a PAP, and a user interface to the PAP, which is integrated into the ViewDS Management Agent.

Access Sentinel extends the XACML framework as follows:

- It extends the PDP's functionality to accept authorization decision requests from an external PEP.
- It includes PEPs to protect applications that are external to ViewDS.
- It includes a dedicated PAP application, the Authorization Policy Manager, for administration of XACML policy.

These features are illustrated below.



The remainder of this section describes some of the key features of the framework and Access Sentinel.

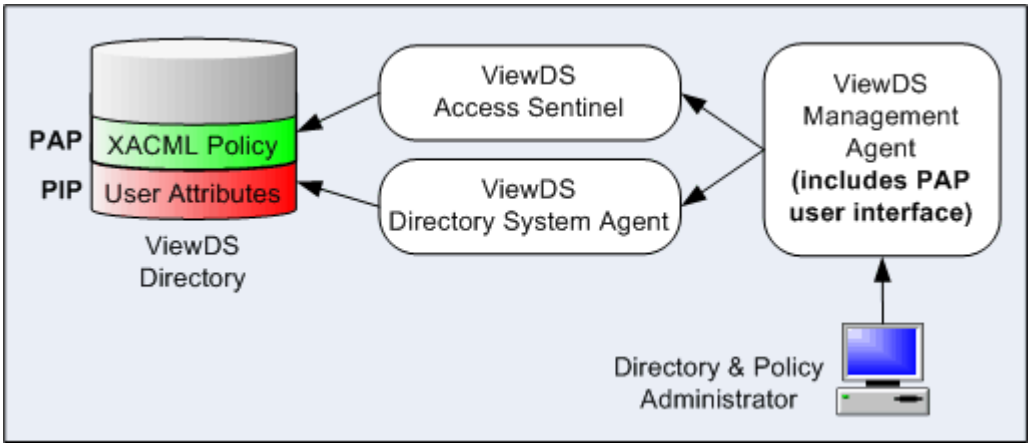
Unified policy server

An important capability of the ViewDS XACML framework is that it unifies the Policy Decision Point (PDP), Policy Administration Point (PAP) and Policy Information Point (PIP). Access to the PAP and PIP is therefore controlled internally, eradicating the complexities and performance overheads associated with the recursive hierarchy described previously.

Unified PIP and PAP user interface

The PAP user interface allows XACML policy to be defined and managed. There are two options for accessing the user interface – the ViewDS Management Agent and the Authorization Policy Manager.

The ViewDS Management Agent is a windows-based application supplied with ViewDS, which allows you to manage multiple implementations remotely. It allows you to manage user attributes stored in the Policy Information Point (PIP), and manage policy in the Policy Administration Point (PAP). You can therefore manage both the PAP and PIP from the same application.



The PAP user interface is also available as a Java-based application, the Authorization Policy Manager, which provides the same PAP functionality as the ViewDS Management Agent. It can be distributed to the most appropriate people in an organisation to help ensure policies are maintained efficiently.

Versioning of access-control policy

Users of either PAP interface can create a new version of a policy and then apply it at their discretion. Users can define when a new version should be enabled allowing them to phase in the new version or roll back to a previous one.

Options for integrating external applications

While ViewDS includes an internal PEP to protect the directory from unauthorized actions, Access Sentinel provides PEP solutions to protect external applications.

The following options are available for integrating external applications with Access Sentinel:

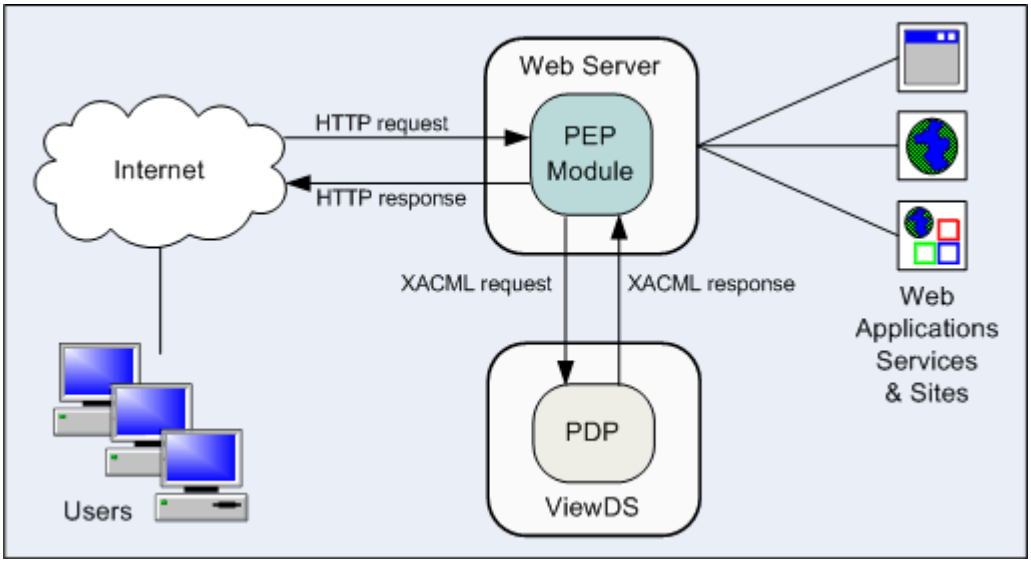
- PEPs: HTTP PEPs for Apache and IIS
- Application Integration Kits for Java and .NET
- SAML
- REST
- JSON over REST

HTTP PEPs

The HTTP PEPs allow XACML policy to be applied to the Microsoft IIS and Apache web servers. Their main tasks are to:

- allow the web server to ask the PEP to enforce authorization decisions for the HTTP requests it receives
- send an XACML authorization decision request to the PDP for each HTTP request, and receive an XACML authorization decision response
- permit or deny access based on the authorization decision

These tasks are illustrated below.



Application Integration Kits for Java and .NET

The Access Sentinel's Application Integration Kits (AIKs) help streamline development of a PEP. They are C# .NET and Java class libraries that abstract the communication between a bespoke PEP and the PDP.

Attempting to communicate with the PDP without the library is complex. There are the intricacies of building the XACML authorization decision request, wrapping and sending it in a SOAP envelope, and intercepting the PDP's response. In contrast, the Application Integration Kits simply require a PEP to make calls that supply the attributes needed to make an authorization decision.

The AIKs are included in the Access Sentinel distribution.

SAML

Access Sentinel supports the SAML 2.0 Profile of XACML, Version 2.0 OASIS standard, allowing any external applications that also support this standard to interact with Access Sentinel for authorization decisions.

The implementation allows the use of SAML 2.0 to carry XACML authorization decisions, authorization decision queries, and authorization decision responses. The method uses HTTP and SOAP as part of the authorization request/response interaction.

REST

Access Sentinel supports the REST Profile of XACML v3.0, Version 1 OASIS standard, allowing any external applications that also support this standard to interact with Access Sentinel for authorization decisions.

The implementation allows the use of XACML in a RESTful architecture, enabling interoperability of RESTful Authorization-as-a-Service (AZaaS) solutions. Unlike the SAML profile, this method does not require the use of SOAP and allows XML-based authorization requests and responses to be transported directly over HTTP.

JSON over REST

Access Sentinel supports the Request / Response Interface based on JSON and HTTP for XACML 3.0, Version 1.0 (Working Draft 14) OASIS draft standard, allowing any external applications that also support this draft standard to interact with Access Sentinel for authorization decisions.

The implementation allows the use of JSON to represent authorization request and response messages that are sent via REST.

Installing and configuring

This section includes the instructions for installing and configuring ViewDS Access Sentinel.

> **NOTE**: The XACML framework, and therefore ViewDS Directory, is a prerequisite for installing Access Sentinel.

> **NOTE**: An Access Sentinel licence is also required.

To install and configure ViewDS Access Sentinel:

1. If ViewDS Directory is not installed, see the *ViewDS Directory: Installation and Operation Guide*.
2. Add the Access Sentinel licence to the DSA's configuration – see the ViewDS Management Agent help topic, *Import licence information*.
3. Read about the XACML configuration parameters.
4. Modify the XACML configuration parameters as required.
5. Optionally, install and configure the Authorization Policy Manager:
   a. Installing the Authorization Policy Manager
   b. Configuring the Authorization Policy Manger
6. Perform one of the following tasks:
   - Deploying the IIS PEP
   - Deploying the Apache PEP

XACML configuration parameters

This subsection describes the XACML configuration parameters that apply to XACML policy, and includes the steps to modify them through the ViewDS Management Agent.

- Combining algorithm
- Default version
- RFC822 name attribute
- User base object

- User attributes
- Resource attributes
- Policy base object
- Allowed origins

Combining algorithm

ViewDS can evaluate policies from different sources: native ViewDS XACML policy (defined using the ViewDS Management Agent or the Authorization Policy Manager) and non-native XACML policy (either declared in the viewDSXACMLPolicySet attribute or supplied in the request).

When an internal decision request is made only native policies are evaluated. If there is more than one native policy, the results are combined using a deny override combining algorithm.

However, when an external decision request is made both native AND non-native policies are evaluated. If a request instructs Access Sentinel to use only policies supplied within that request (CombinePolicy=false), then the evaluation of other policies (for example native policies) will result in a Not Applicable outcome.

If a request instructs Access Sentinel to combine polices supplied within that request and other policies (CombinePolicy=true) then native polices are evaluated using a deny override combining algorithm and non-native policies are evaluated using the combining algorithm specified for that non-native policy set.

The results (native and non-native) are then combined using a Combining Algorithm:

- urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides
  Deny overrides: If any nested item (rule, policy or policy set) evaluates to deny, then the container (policy or policy set) evaluates to deny; otherwise, if any item evaluates to permit, then the container evaluates to permit; otherwise, the container evaluates to not-applicable.
- urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides
  Permit overrides: If any nested item evaluates to permit, then the container evaluates to permit; otherwise, if any item evaluates to deny, the container evaluates to deny; otherwise, the container evaluates to not-applicable.
- urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit
  Deny unless permit: If any nested item evaluates to permit, then the container evaluates to permit; otherwise, the container evaluates to deny.
- urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny
  Permit unless deny: If any nested item evaluates to deny, then the container evaluates to deny; otherwise, the container evaluates to permit.

For further information see the XACML 3.0 specification.

Default version

Every XACML policy has a version number.

By default, when there are multiple policies or policy sets with the same identifier, the Policy Decision Point (PDP) uses the one with the highest version number. Alternatively, if a Default Version is defined, the PDP uses the policy or policy set with the highest version number less than or equal to this value.

This parameter only applies to XACML policy that was not defined through the ViewDS Management Agent or the Authorization Policy Manager.

RFC 822 name attribute

This configuration parameter identifies a directory attribute that conforms to RFC 822 format. It allows the Policy Decision Point (PDP) to identify a subject by its email address.

This parameter applies to all XACML policy.

For more information, see Attribute look-up.

User attributes

The directory attributes the Policy Decision Point (PDP) will pre-fetch when it needs to obtain a directory attribute from a user's entry.

This parameter applies to all XACML policy.

Resource attributes

The directory attributes the Policy Decision Point (PDP) will pre-fetch when it needs to obtain a directory attribute from a resource entry (see Attribute look-up).

This parameter applies to all XACML policy.

User base object

The root of the subtree in the directory that the Policy Decision Point (PDP) will search in order to find a user entry. (The directory acts as a Policy Information Point by storing information that can influence an access decision.)

This parameter applies to all XACML policy.

Policy base object

The root of the subtree in the directory that the Policy Decision Point (PDP) will search in order to find a policy or policy set.

This parameter applies to all XACML policy.

Allowed origins

Defines a cross-origin resource sharing (CORS) policy.

A CORS policy specifies the origins in HTTP requests that the Policy Decision Point (PDP) accepts.

Each item in the list is a regular expression (see https://www.w3.org/TR/xmlschema-2/#regexs) that is matched against the origin in a request. The origin is accepted if it matches at least one expression and rejected if it matches none of the expressions.

Consider the following example:

```
http://.*\.example\.com(:[0-9]+)?
```

This regular expression matches origins that specify any port, or no port, in any sub-domain of `example.com`.

If no expressions are defined then all origins are rejected.

If an HTTP request does not specify an origin it is always accepted.

Modifying the XACML configuration

To set the XACML configuration parameters through the ViewDS Management Agent:

1. At the bottom of the left pane, click **Server View**.
2. In the left pane, click the appropriate server.
3. In the right pane, click the **XACML Config** tab.
4. Complete the boxes in the XACML Config tab as required.
5. At the bottom of the tab, click **Set XACML Configuration**.

Installing the Authorization Policy Manager

The Authorization Policy Manager is a stand-alone PAP that can be installed on any platform. It provides the same XACML access control functionality as the ViewDS Management Agent.

To install the application:

1. Install Java SE Runtime (32-bit).
2. From the Access Sentinel distribution media, unzip the file `PAPui.zip`

Starting the Authorization Policy Manager

To start the application either:

- In the extracted folder, double-click `PAPui.jar`
- From a command shell, enter `PAPui.jar`

Trusted mode

When the application is in 'trusted mode', appropriate access is granted to a non-administrative user who has been delegated administration rights to XACML policy (see Composition-time delegation).

To start the application in trusted mode, enter either of the following from a command shell:

```
PAPui.jar --trusted
PAPui.jar -t
```

Configuring the Authorization Policy Manager

Configuring the application involves the following tasks:

- Installing a user certificate
- Connecting to a ViewDS Directory
- Getting started
- Setting local file security

Installing a user certificate

A user must be authenticated before they can connect the Authorization Policy Manager to a ViewDS server.

There are two options available:

- **Simple authentication** – the user enters the username and password assigned to their entry in the ViewDS directory. With simple authentication, the user has non-administrator access to XACML policy. However, the user will have administrator access to specific XACML policy if the application is started in trusted mode and an administrator has delegated trust to the user through composition-time delegation.
- **Certificate based authentication** – a user certificate needs to be installed and imported into the Authorization Policy Manager. With certificate based authentication, the user has administrator access to all XACML policy.

To install a user certificate (PKCS #12 file):

1. Add the user certificate to the `trusted` subdirectory below the ViewDS install directory. For example, on a Windows installation:

   `%VFHOME%\setup\trusted`

   Where `%VFHOME%` is the ViewDS install directory.

   For further details, see *Installing credentials* in the *ViewDS Directory: Installation and Operation Guide*.
2. Import the PKI credentials into the Authorization Policy Manager:
   a. From the Authorization Policy Manager's menu bar, click **Tools** followed by **Keystore**. A new window is displayed.
   b. Click the **Yes with password** button to create a keystore for the application.
   c. Enter a password and click **OK**. The Key Store window is displayed.
   d. In the **Key Store** window, click **Import** and follow the prompts to import the user certificate into the keystore.

Connecting to a ViewDS Directory

To connect the Authorization Policy Manager to a ViewDS Directory Server Agent (DSA):

1. Click **File** followed by **New Session**. The Session window is displayed.
2. Enter a session **Name** of your choice to appear in the left pane.
3. In the **Host** box, enter the address of your Directory Server (DSA). For example, if the Authorization Policy Manager is on the same host as the DSA, enter localhost.
4. Enter the **Port** number to connect to on the DSA (by default, 3000).
5. For simple authentication, enter your **Username** and **Password** in the Simple tab. Otherwise, for strong authentication, click the **Certificate** tab and select a **Key Alias** and enter your **Password**.
6. Select the **Connect immediately** checkbox.
7. Click **Save**. The session is displayed in the left pane.

Getting started

This task introduces you to the interface:

1. In the left pane, right-click **Deltawing** and from the drop-down menu click **Add XACML Access Control Domain**. Three tabs are displayed in the right pane: Policy Versions, Attributes, and Roles. The interface is equivalent to the XACML AC tab in the ViewDS Management Agent. Both allow you to perform exactly the same tasks.
2. In the right pane, click the **New** button. The **New XACML Policy** window is displayed.
3. In the **Label** box, enter a name, such as 'test'.
4. Click **Save** to accept the defaults. A new policy is listed in the Policy Versions tab.
5. To remove the XACML Access Control Domain, right-click the **Deltawing** entry followed by **Remove XACML Access Control Domain**.

Setting local file security

The in-application help is displayed in the user's browser. The browser must allow javascript to interact with the local file system for hypertext links in the help to work. This may require additional configuration of the browser.

If the hypertext links in the Authorization Policy Manager help do not work in the Chrome browser for Windows:

1. Close all instances of Chrome.
2. Enter at the command line: `chrome.exe --allow-file-access-from-file`

For Firefox:

1. In the browser's address bar, enter `about:config` and press Enter. The browser's preferences are displayed.
2. In the Search box, enter `privacy.file_unique_origin`.
3. Set the Value of `privacy.file_unique_origin` to **false**.

Deploying the IIS PEP

The IIS PEP is an IIS managed module that allows a Microsoft IIS web server to delegate authorization of HTTP requests to Access Sentinel. It can be deployed to protect access to specific sets of pages in a site.

Deploying the IIS PEP module involves:

- Enabling .NET extensibility for IIS
- Adding the PEP to the IIS
- Configuring the IIS PEP
- Configuring for anonymous access
- Testing the deployment

Enabling .NET extensibility for IIS

To enable .NET extensibility for IIS on Windows 10:

1. From the Windows Control Panel, select **Programs and Features**.
2. Click **Turn Windows features on or off**. The Windows Features window is displayed.
3. Expand **Internet Information Services**, then **World Wide Web Services**, and **Application Development Features**.
4. Select the **.NET Extensibility 3.5** checkbox.

To enable .NET extensibility for IIS on Windows Server 2019:

1. Open Server Manager.
2. From the Dashboard, click **Add roles and features**. The Add Roles and Features Wizard opens.
3. Click **Next** until the Features list is displayed.
4. Select **.NET Framework 3.5 Features** then click **Next** followed by **Install**.

Adding the PEP to IIS

To add the PEP module to a website:

1. From the Access Sentinel distribution media, copy `IISpepModule.dll` and `pdpLiaison.dll` to the site's bin folder (create a bin folder if one does not exist).
2. Add `IISpepModule.dll` to the required website as a managed module. For example, to add the PEP as a managed module through IIS Manager:
   a. From IIS Manager, click the required website in the Connections pane.
   b. In the central pane, double-click **Modules**. The modules are listed.
   c. In the **Actions** pane on the right, click **Add Managed Module**. The Add Managed Module window is displayed.
   d. In the **Name** box, enter `Access Sentinel PEP`.
   e. In the **Type** box, enter `IISpepModule.PEP`, then click **OK**.

Configuring the IIS PEP

To configure the IIS PEP:

1. Create a folder for the PEP's log file (for example, `c:\peplog`).
2. Grant full access to the PEP's log file:
   a. From Windows Explorer, right-click the log-file folder (for example, `c:\peplog`) and click **Properties**. A Properties window is displayed.
   b. Click the **Security** tab.
   c. Click the **Edit** button. The Permissions window is displayed.
   d. Click the **Add** button. The Select Users or Groups window is displayed.
   e. In the text box, enter Network Service and then click **OK**. The window closes and NETWORK SERVICE is added to the Security tab.
   f. Click **NETWORK SERVICE** followed by the **Allow** checkbox for Full control.
   g. Click **Apply** and then **OK**.
3. From the Access Sentinel distribution media, copy `pepConfig.txt` to the IIS folder (for example, `c:\Windows\System32\inetsrv\`).
4. Set the configuration-file parameters (see below) in the `pepConfig.txt` file as required.

IIS PEP configuration-file parameters

The IIS PEP has a configuration file `pepConfig.txt` with the following parameters:

`XACMLHost` The host name or IP address of the host on which the ViewDS DSA is running. For example: `localhost`

`XACMLPort` The soapAddress on the ViewDS DSA where the PDP listens for authorization decision requests (see Modifying the SOAP address). Default: `3009`

The following is an example configuration file:

```
XACMLHost localhost
XACMLPort 3009
```

Configuring for anonymous access

To configure the ViewDS DSA for access by the PEP as an anonymous user:

1. Open the ViewDS Management Agent.
2. At the bottom of the left pane, click **Server View**.
3. In the left pane, click the appropriate server.
4. In the right pane, click the **Trust** tab.
5. Within the **Trust** tab, click the **Anonymous Privilege** tab.
6. Select the **XACML Protocol** checkbox.
7. In the **Access Rights** box, click **read**.
8. Click **Save**.

Test the deployment

To test the deployment:

1. Attempt to access the protected website. You should be denied access, which is the default behaviour if no XACML policy has been defined.
2. Optionally, if required, perform the task Tracing decision making.
3. Define an XACML policy by following the HTTP PEP tutorial.

Deploying the Apache PEP

The Apache PEP protects web pages hosted by an Apache web server, which implement HTTP authentication. It requires Apache HTTP Server version 2.2.

Deploying the Apache PEP module involves:

- Installing and configuring the Apache PEP
- Configuring for anonymous access
- Testing the deployment

Installing and configuring the Apache PEP

1. From the Access Sentinel distribution media, copy the PEP module `mod_authz_xacml.so` to the Apache modules directory. For Windows this may be, for example, `\Program Files (x86)\Apache Software Foundation\Apache2.2\modules`.
2. In the Apache configuration file, add a LoadModule directive for the Apache PEP: `modules/mod_authz_xacml.so`
3. Each directory that has HTTP authentication and will be protected by the Apache PEP requires the following parameters in the Apache configuration file:

```
XACMLHost "localhost"
XACMLPort 3009
XACMLTrace on
Require permit
```

The parameters are described below.

Apache PEP configuration parameters

The following PEP configuration parameters can appear in the Apache configuration file:

`XACMLHost`	The host name of the ViewDS server (includes the PDP).
`XACMLPort`	The ViewDS server's soapAddress (default, 3009) where the PDP listens for authorization decision requests (see Modifying the SOAP address).
`XACMLTrace`	Optional. Determines whether the PEP's authorization decision requests will switch on decision tracing in the PDP. (Tracing is written to the server's query log.)

`Require permit`	Is required to invoke PEP. It is a standard Apache directive, but the value of permit is specific to Access Sentinel.
`XACML Authoritative`	Optional. Determines whether this module is the authoritative authorisation module. When absent, the default is on (the recommended setting).

Example configuration

This example configuration applies the Apache PEP to a directory that has basic authentication in a Windows environment:

```
LoadModule authz_xacml_module "modules/mod_authz_xacml.so"
  <IfModule authz_xacml_module>
    <Directory "<path to directory with basic HTTP authentication>">
      AuthType Basic
      AuthName "Basic"
      AuthUserFile "<path to directory with basic HTTP auth>/users"
      XACMLHost "localhost"
      XACMLPort 3009
      XACMLTrace on
      Require permit
      AllowOverride None
      Options FollowSymLinks
    </Directory>
  </IfModule>
```

This example references a users file, which is described in Apache's documentation for HTTP basic authentication (see http://httpd.apache.org/docs/2.2/mod/ mod_authn_file.html and http://httpd.apache.org/docs/2.2/mod/ mod_authz_groupfile.html).

Configuring for anonymous access

To configure the ViewDS DSA for access by the PEP as an anonymous user:

1. Open the ViewDS Management Agent.
2. At the bottom of the left pane, click **Server View**.
3. In the left pane, click the appropriate server.
4. In the right pane, click the **Trust** tab.
5. Within the **Trust** tab, click the **Anonymous Privilege** tab.
6. Select the **XACML Protocol** checkbox.
7. In the **Access Rights** box, click **read**.
8. Click **Save**.

Test the deployment

To test the deployment:

1. Attempt to access the protected website. You should be denied access, which is the default behaviour if no XACML policy has been defined.
2. Optionally, if required, perform the task Tracing decision making.
3. Define an XACML policy by following the HTTP PEP tutorial.

Modifying the SOAP address

The IIS and Apache PEPs exchange authorization decision requests and responses with the PDP. Each is wrapped in a SAML assertion, inserted into a SOAP envelope, and then added to the payload of an HTTP request or response.

The PDP listens for authorization decision requests on the SOAP address declared in the ViewDS server's configuration. By default, the SOAP address is 3009 (the server's baseport address, 3000, plus 9).

To modify the SOAP address:

1. Start the ViewDS Management Agent.
2. At the bottom of the left pane, click **Server View**.
3. In the left pane, click the appropriate server.
4. In the right pane, click the **Configuration** tab.
5. Within the **Configuration** tab, click **Addresses**.
6. Double-click in the **Value** column next to **SOAP Address**.
7. Enter the appropriate address and then click **Set**.

Tracing decision making

When the PEP sends an authorization decision request and tracing is enabled:

- The PDP generates a trace of the policies evaluated and the result of each. It logs the trace in its query log (see the ViewDS Management Agent help topic, *Working with the query log*).
- The PDP also returns the trace in its authorization decision response. The PEP then logs the trace in the directory identified by the PEP's configuration-file parameter `LogPath` This functionality is currently only available for the IIS PEP.

To enable tracing:

1. Set the PEP's configuration-file parameter `XACMLTrace` to `on` (see the IIS PEP configuration parameters or Apache PEP configuration parameters).
2. Enable the DSA's query log:
   a. From the ViewDS Management Agent, click the **Server View** button.
   b. In the left pane, click your DSA.
   c. In the right pane, click the **Configuration tab** followed by **Runtime Settings**.

d. For the 'Query logging' setting, select **on** in the **Current** and **On Start Up** columns.

e. Click **Set**.

3. Define an XACML attribute with the following settings:
   - Label equals `tracing` (for example)
   - Category equals `urn:oasis:names:tc:xacml:3.0: attribute-category:action`
   - Identifier equals `urn:oasis:names:tc:xacml:1.0: action:action-id`
   - Data Type equals `anyURI`
4. Create a new rule within the policy. The rule's condition should be that the above XACML attribute is equal to `http://viewDS.com/xacml/environment/trace`

About XACML framework and policy

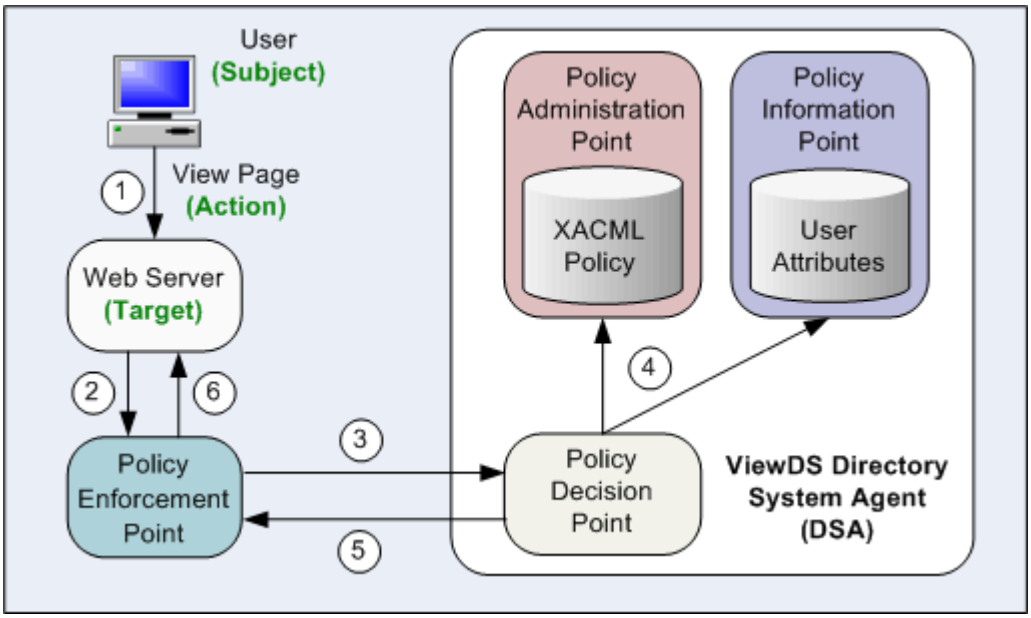
This section provides the background information you will need to write XACML policy.

It describes the following:

XACML components

We'll start by looking at how Access Sentinel's implementation of XACML can be used to protect web pages.

Consider the following illustration.



The steps shown in the illustration are described below.

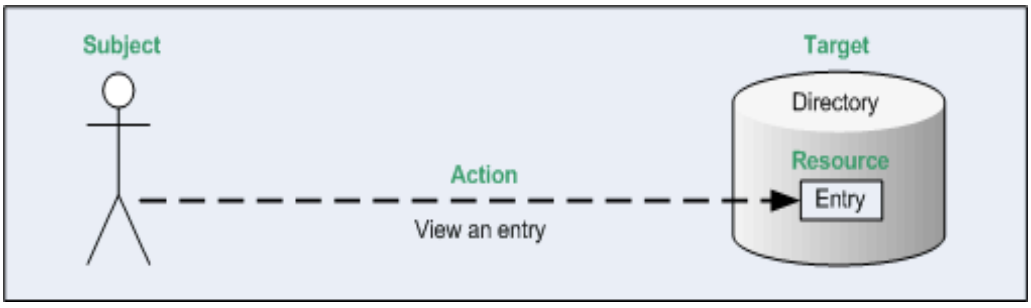
1. A user attempts to view a web page hosted by a web server.
2. The web server asks the Policy Enforcement Point (PEP) to form an 'authorization decision request'.
3. The PEP sends the 'authorization decision request' to the Policy Decision Point (PDP). The authorization decision request includes XACML attributes that identify, among other things, the user and the web page they are attempting to access. (See XACML attributes provided by a PEP for details.)
4. The Policy Decision Point (PDP) determines whether access should be permitted. It does so by accessing the appropriate XACML policy. The policy instructs the PDP to consider which web page is being accessed and by which user. The user is identified according to directory attributes in the Policy Information Point (PIP).
5. The PDP returns an 'authorization decision response' to the PEP.
6. The web server acts on the decision to permit or deny access to the web page.

XACML terms to remember

There are a couple of important XACML terms to remember:

- Target – the set of resources protected by the policy.
- Resource – the specific item (e.g. web page) within the target that the subject is attempting to access.
- Subject – the user attempting to access a resource.
- Action – the action attempted by the subject (e.g. view a web page).

These terms are illustrated below:



Introduction to XACML policy

The ViewDS implementation of an XACML policy comprises:

- XACML Access Control Domain
- Status and version
- XACML attributes
- Rules

Each is described below.

XACML Access Control Domain

An XACML Access Control Domain is a specific area of a DIT that contains one or more XACML policies.

> **NOTE**: In the ViewDS XACML framework, the default behaviour is to deny access to the entities within an Access Control Domain. (This does not apply to administrative users of the ViewDS Management Agent, who bypass all access controls.)

For example, when working with the ViewDS directory and the internal PEP, an XACML Access Control Domain is an area of the directory where the XACML access controls apply. The entry at the top of the domain is termed the access control administrative point. By default, ViewDS denies access to all entries within the domain.

Status and version

Every XACML policy has a status and version.

A policy can have multiple versions, each with a unique version number. A version also has a status that identifies whether it is 'locked' and 'active'.

Only one version of a policy can be 'active'. This is the version that currently applies. You can therefore test a new version of a policy and then roll-back to a previous version if necessary.

A 'locked' version cannot be modified. However, you can create a new version based on an existing locked version. This offers a level of version control.

Status: `active,locked`
Version: `1.1`

XACML attributes

XACML is based on the concept of attributes.

The PAP uses XACML attributes to identify the subject, resource, action and environment information within a rule. The PEP sends requests made up of XACML attributes to the PDP to convey information about the subject, resource, action and environment. The PDP then compares these to attribute values in a policy to make access decisions.

The XACML standard defines four categories for attributes:

- Subject – identifies the subject attempting to access a particular resource.
- Resource – identifies the resource the subject is attempting to access.
- Action – identifies the action the subject is attempting to perform on the resource (for example, read, modify).
- Environment – identifies environmental factors such as day of the week and time of day.

It is permissible within the XACML standard for any of these four categories to be sub-divided or for other new attribute categories to be added.

For details of the XACML categories and data types of the attributes provided by the PEP, see XACML attributes provided by a PEP.

For an XACML attribute to be included in policy rules, it must first be declared in the XACML Access Control Domain. Declaring an XACML attribute involves giving it a 'user-friendly' name. This is important because XACML attributes are identified by long URIs or complex XPath expressions that are unwieldy when creating rules.

You can declare two different types of attributes: attribute designators and selectors.

Attribute Designators

An attribute designator comprises the `Category`, `AttributeId` and `DataType` URIs of a particular XACML attribute.

For some XACML attributes, the declaration also includes a mapping to a directory attribute in an entry that uniquely identifies a subject or resource.

Attribute designators allow a policy to specify an attribute value with a given category, identifier and data type. The PDP will then look for that value in the request, or elsewhere, if no matching values can be found in the request (see Attribute look-up).

Attribute Selectors

In addition to attributes, XACML requests can contain XML documents for each category. For example, an XML document might describe the subject or be the actual resource being accessed.

Attribute selectors allow a policy to look for attribute values in an XML document using XPath queries. XPath is a language, based on a tree representation of XML documents, which provides the ability to navigate the tree and select nodes using a variety of criteria.

An attribute selector comprises a category, data type and an XPath expression. Together these are used to resolve a set of attribute values in the request document.

Attribute selectors can be used within XACML policy expressions in the same way as attribute descriptors. For example, consider an XACML request that contains an XML document which is the resource a user is attempting to access. An attribute selector can include an XPath expression to find elements in the document named PublicationDate. An XACML policy can then include a condition that denies access if the PublicationDate is more than five years ago.

The following are currently supported:

- the definition of attribute selectors within the Authorization Policy Manager (and the ViewDS Management Agent)
- the ability to use and evaluate attribute selectors within XACML policies

NOTE: Attribute selectors are not applicable to the ViewDS XACML framework or HTTP PEPS. This is because they do not make use of XML documents within authorization decision requests.

Rules

Every XACML policy includes a rule.

A rule allows the Policy Decision Point (PDP) to determine whether a subject should be permitted or denied access to a resource. Each has a target, scope, an effect (permit or deny access) and a condition.

The *target* identifies the resources protected by the policy. The scope is used when defining policy for hierarchical resources, such as directory entries. It determines whether the policy applies to a single target resource, or to a target resource and all its subordinates.

The condition incorporates XACML attributes which the PDP uses to identify the resource and subject. It determines whether the rule's effect should be applied.

A simple example rule is shown below.

Rule
Target: Documents
Scope: subtree
Effect: Permit access (if the following condition is true)
Condition:
resource has attribute `webpage` = '`index.html`' AND
subject has attribute `role` = `Board Member` AND
action = READ

The *condition* is true if the subject is a Board Member attempting to view the resource 'index.html'.

For further information about rules, see More about XACML policy.

Also to consider is that there are two kinds of rules: attribute- and role-based access control (ABAC and RBAC).

More about XACML policy

This section describes attribute look-up along with the following concepts relating to rules:

- Obligations and advice
- Delegation
- Precedence

Attribute look-up

Access Sentinel has the ability to look-up subject and resource attributes that are not provided in an authorization decision request. (This functionality only applies to requests from applications external to ViewDS.)

Subject attributes

If subject attributes are not provided in an authorization decision request, the Policy Decision Point (PDP) will attempt to look them up in the Policy Information Point (the ViewDS directory). For this to occur the request must include the following XACML attribute:

`urn:oasis:names:tc:xacml:1.0:subject:subject-id`

The PDP will look up the subject-id XACML attribute definition from within the XACML Access Control Domain to identify if it has been mapped to a directory attribute. If it has, then the PDP will used this directory attribute to search ViewDS for the subject. If the subject-id does not have a directory attribute mapping, it will use the following defaults based on the subject-id data type (see XACML Configuration Parameters):

- String – the Policy Decision Point looks for a directory entry whose viewDSUserName attribute equals the string value specified by subject-id.
- x500Name – the Policy Decision Point looks for a directory entry whose LDA Distinguished Name equals the specified X500 name specified by subject-id.
- rfc822Name – the Policy Decision Point looks for a directory entry that has a value of the attribute type identified by the rfc822Name-attribute that is configured within the XACML Configuration setting.

The PDP only expects to find a single subject entry within ViewDS. If multiple entries are located it will consider the situation to be ambiguous and will not use any of the subject attributes from within the PIP.

Resource attributes

If required resource attributes are not provided in an authorization decision request, the PDP will attempt to look them up in the PIP (the ViewDS directory). For this to occur the request must include the following XACML attribute:

`urn:oasis:names:tc:xacml:1.0:resource:resource-id`

The PDP will look up the resource-id XACML attribute definition from within the XACML Access Control Domain to identify if it has been mapped to a directory attribute. If it has, then the PDP will used this directory attribute to search ViewDS for the resource.

The PDP only expects to find a single resource entry within ViewDS. If multiple entries are located it will consider the situation to be ambiguous and will not use any of the resource attributes from within the PIP.

Obligations and advice

Obligations and advice are features of XACML 3.0 that have been implemented in ViewDS so that it can be used to convey directives to applications that define them within an XACML response. An obligation is a mandatory directive whereas advice is optional.

To illustrate, an obligation to add a log entry might be associated with permitting access to a highly restricted resource. In this case, when the application is told that access is permitted it is

also told that it is obliged to log the access for auditing purposes. If the application cannot perform the logging operation, it will refuse access to the resource.

Advice is similar to an obligation, except execution of advice by the application is optional.

For example an XACML response might deny access to a document on the weekend and come with the advice to show a message to the user that access is only available on week days.

The specific obligations and advice implemented by a given application are defined by that application. Access Sentinel merely enables you to associate such obligations and advice with authorization rules and so use them in access control decisions.

NOTE: Neither ViewDS nor the HTTP PEPs define any obligations or advice for use in creating access control policy. So, if a policy that grants access contains an obligation, then ViewDS and the HTTP PEPs will not permit the operation due to their inability to process the obligation. Both PEPs ignore advice.

Delegation

ViewDS Access Sentinel allows an administrator to delegate trust to non-administrative users.

The primary component of delegation is the *administrative rule*. For context, compare its purpose with that of an *access rule*:

- **Access rule** - determines whether ViewDS should grant or deny access to a resource.
- **Administrative rule** - determines the who, what and where of a delegation. That is, who it applies to, along with the scope and constraints.

To expand on this distinction, consider that Access Sentinel ignores an *access rule* unless:

- it was written by an administrator; or
- it is authorized by a chain of *administrative rules*, where the final rule in the chain was written by an administrator.

The rule is then deemed 'trusted'.

Only an administrator can manage trusted rules. An administrator is a trusted user of the Authorization Policy Manager or ViewDS Management Agent. (Note that the ViewDS Management Agent can only be accessed by an administrator, as it requires certificate-based authentication. The Authorization Policy Manager offers both certificate-based and simple authentication to allow access by non-administrative users.)

However, an administrator can delegate authorization to manage trusted rules. The administration of rules can therefore be decentralised by delegating trust to users of the Authorization Policy Manager.

Access Sentinel provides two ways to delegate trust:

- Evaluation-time delegation
- Composition-time delegation

Each is discussed below.

Evaluation-time delegation

An *administrative rule* can be used to delegate trust to a non-administrative user of the Authorization Policy Manager.

The user would be trusted to maintain the rules in a policy, but within a specified scope and under specified constraints.

To illustrate, an *administrative rule* might be written to authorize *access rules* written by a Sales Manager (a non-administrative user). However, the *administrative rule* could also specify that the Sales Manager's rules only apply to the 'Sales and Marketing' area of the directory.

Later, when Access Sentinel evaluates an authorization request to access a resource, it considers all relevant rules, including those relating to delegation. For the above example, Access Sentinel would determine whether to apply the rules written by a Sales Manager. If the authorization request related to the 'Sales and Marketing' area of the directory, then Access Sentinel would apply the Sales Manager's rules. Otherwise, it would simply ignore them.

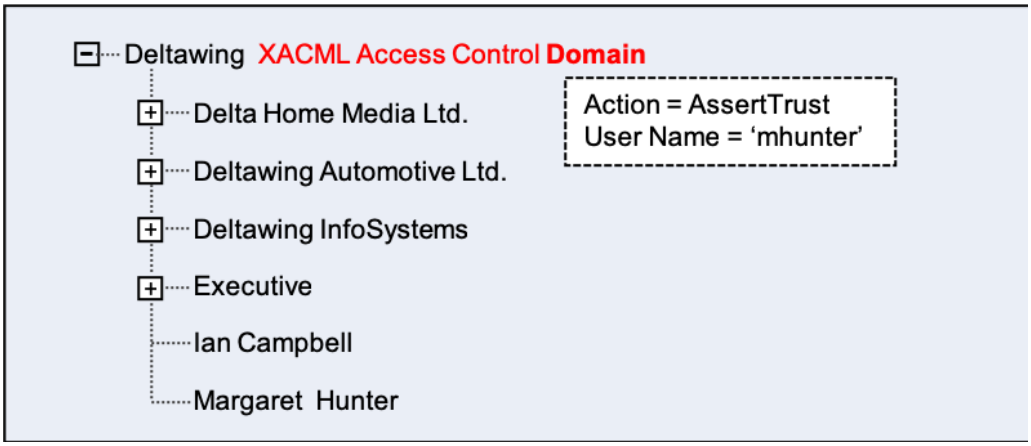
Composition-time delegation

This is another way for an administrator to delegate trust to a non-administrative user.

An administrator can create an access rule that delegates administrator rights within an *XACML Access Control Domain* or within an *XACML Access Control Subdomain*. Each option is described below.

XACML Access Control Domain

To illustrate composition-time delegation within a domain, consider the following illustration.



In this example an administrator has created:

- an *XACML Access Control Domain* at the Deltawing entry

And written an access rule that:

- delegates trust by permitting the action 'AssertTrust' by a non-administrative user, Margaret Hunter

Consequently, after starting the Authorization Policy Manager with the '-trusted' switch, Margaret Hunter would be considered an administrator for the purpose of managing policy within the XACML Access Control Domain. There would, however, be no restrictions on the non-

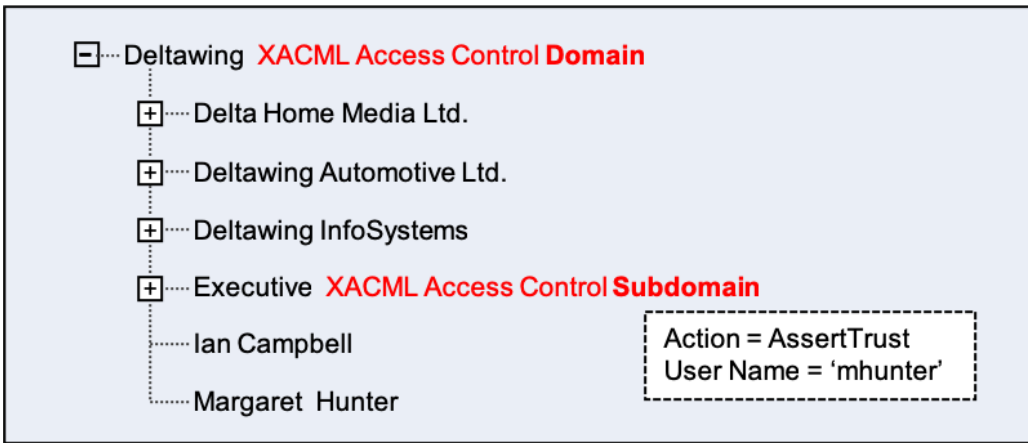
administrative user. Margaret Hunter would be able to modify every aspect of the access controls in the domain: rules, attributes, versions, policies, roles, and named expressions.

The only way to impose a restriction is to use precedence. For example, the administrator could amend the access policy so that the non-administrative user can only manage policy containing rules with a precedence greater than 1. Therefore, a rule with a precedence of 0 or 1 could only be modified by an administrator, and would always override those managed by the non-administrative user.

This restriction would only apply to a policy's rule as attributes, versions, roles and named expressions cannot be assigned a precedence.

XACML Access Control Subdomain

To illustrate composition-time delegation within a subdomain, consider the following illustration.



As in the previous example, the administrator has created an *XACML Access Control Domain* at the Deltawing entry. However, this time, they have also created:

- an *XACML Access Control Subdomain* at the Executive entry

The administrator has taken the same access rule shown in the previous example, and this time applied it to the XACML Access Control Subdomain. Consequently, after starting the Authorization Policy Manager with the '-trusted' switch, Margaret Hunter would be considered an administrator for the purpose of managing policy within the subdomain.

As well as any restrictions declared by the access rule, there are inherent restrictions imposed by this type of delegation. The non-administrative user can create versions, policy and named expressions within the subdomain, but they cannot create attribute and role definitions. The only attribute and role definitions available to the non-administrative user are those inherited by the sub-domain.

Precedence

By default the rule in an XACML policy has a precedence of 0 (zero).

When the Policy Decision Point (PDP) receives an 'authorization decision request' it evaluates the rules with a precedence of 0. This gives a result of either 'permit', 'deny', 'indeterminate' or 'not applicable'.

When the result is 'not applicable', the PDP then evaluates rules with a precedence of 1. If this evaluation returns the same result, the PDP then moves onto rules with a precedence of 2, and so on. At any stage, if the result is anything but 'not applicable', the evaluation ends and PDP returns the result to the Policy Enforcement Point (PEP).

A rule's precedence can be set through either the ViewDS Management Agent or Authorization Policy Manager. It can be set to zero or any integer value (they do not need to be sequential) in order to override rules with a higher precedence value.

In summary, a rule with a precedence of zero overrides a rule with a precedence of 1, for example.

Attribute- versus role-based access control

The ViewDS XACML framework supports both attribute-based access control (ABAC) and role-based access control (RBAC) rules.

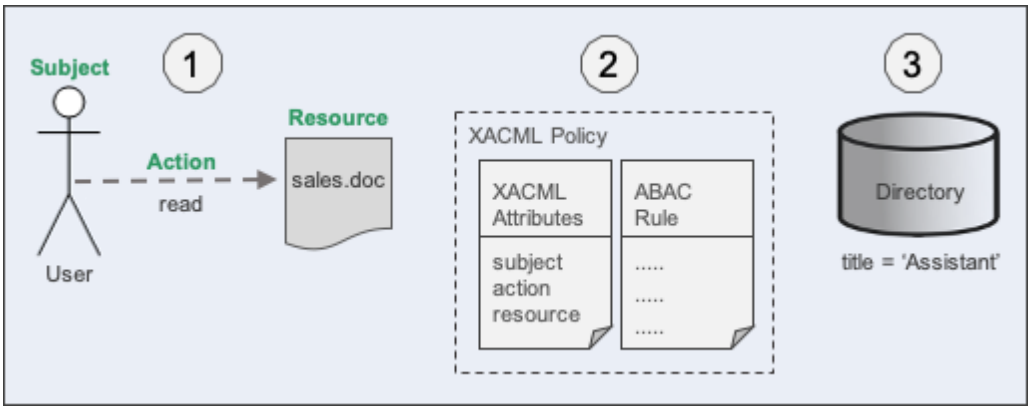
Attribute-based access control

With ABAC, the attributes associated with the subject, action, resource or environment are used to construct conditions. These conditions compare attributes to static values, or to one another (relation-based access control), to determine whether access should be permitted or denied.

To illustrate, here is an example ABAC rule:

```
Permit access if:
   action = read AND
   resource = sales.doc AND
   subject's title = 'Assistant'
```

The following illustration includes the example rule.



The steps in the illustration are as follows:

1. A user attempts to read the `sales.doc` file.
2. ViewDS evaluates the XACML policy. The XACML Attributes definition tells ViewDS to obtain the 'subject's title' from the value of the `title` attribute in the user's directory entry.
3. As the user's `title` attribute is set to 'Assistant', all the clauses in the ABAC rule are true and ViewDS permits access to the document.

Role-based access control

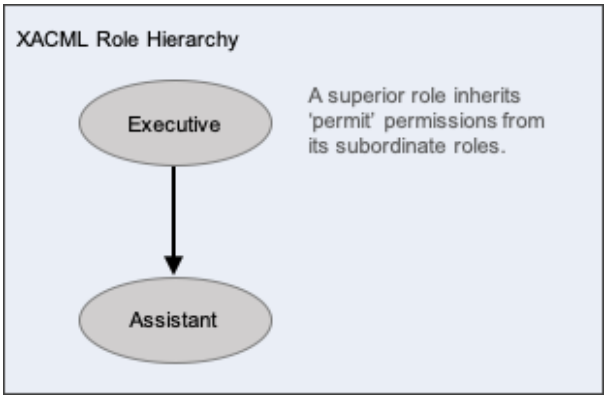
RBAC also uses attributes to construct conditions. However, unlike an ABAC rule, an RBAC rule includes a role condition that is evaluated against a role hierarchy.

For example, here is an example RBAC rule:

```
Permit access if:
 action = read AND
 resource = sales.doc AND
 subject's role = Assistant
```

Aside from the 'role' clause, this is identical to the ABAC example. However, another important distinguishing feature of RBAC is that it includes a role hierarchy.

A role hierarchy defines permission inheritance. It allows ViewDS to evaluate a user's access rights according to their role's position in the hierarchy. However, while each role inherits access rights from its subordinates, this only applies to the 'permit' permissions. In the following example, the role 'Executive' inherits permissions from the role 'Assistant'.



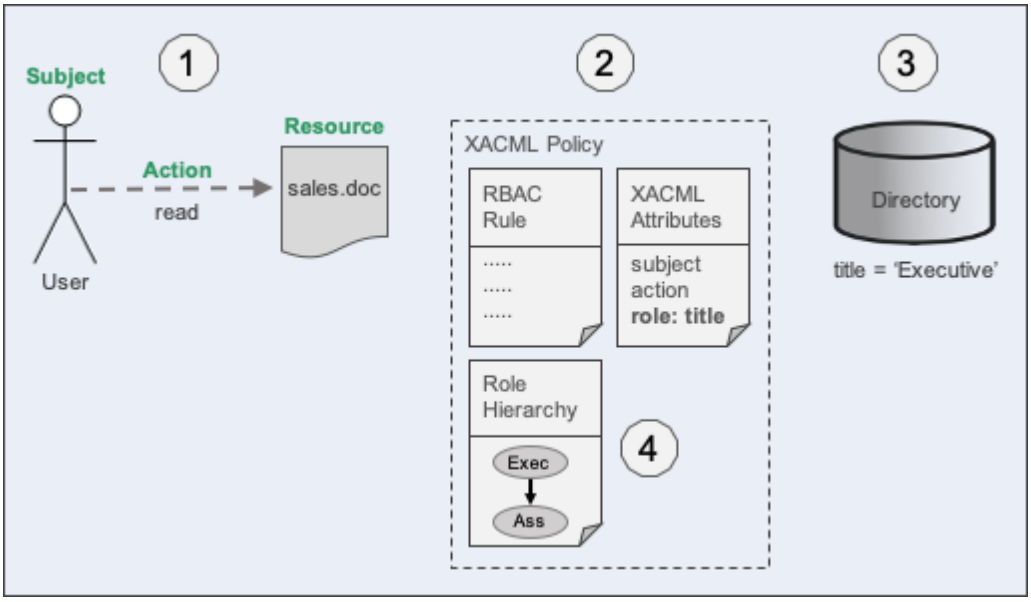
When ViewDS evaluates the previous rule that permits access to an 'Assistant', it also evaluates the role hierarchy which defines that an 'Executive' also has this permission. However, before ViewDS can evaluate the role hierarchy, it first needs to map the user (the subject) to a role. This role mapping can be either:

- **Static** - static roles are obtained from a directory entry or the XACML request.
- **Dynamic** - dynamic roles are allocated at run-time by evaluating **role enablement rules**.

Both options are described below.

Static roles

With static roles, the policy's role definition identifies an attribute in the ViewDS directory.



The above illustration shows an XACML policy that includes a static role, which is mapped to the `title` attribute in a user's directory entry.

The steps in the illustration are as follows:

1. A user attempts to read the `sales.doc` file.
2. ViewDS evaluates the XACML policy. The definition of `role` tells ViewDS that the subject's role is the value of their `title` attribute in the directory.
3. From the value of the user's 'title' attribute, ViewDS determines that user's role is 'Executive'.
4. From the policy's role hierarchy, ViewDS identifies that a 'Executive' inherits permissions from an 'Assistant'. ViewDS therefore grants the user access to the document.

With static roles, the names of the roles in the role hierarchy must exactly match values in the corresponding directory attribute.

As you can probably see, the only real difference between RBAC with static roles and ABAC is the permission inheritance defined by a role hierarchy.

Role enablement

Role enablement provides greater flexibility to RBAC by allowing ViewDS to assign roles dynamically. It does this by evaluating a policy's *role enablement rules*.

To illustrate, consider these example role enablement rules:

> Assign the role of **Executive** if:
> subject's `title` attribute contains 'Director' OR 'Chief Executive Officer' OR 'Board Member'

> Assign role of **Assistant** if:
> subject's `title` attribute contains 'Executive Assistant' OR 'Personal Assistant' OR 'Administrative Assistant'

When ViewDS assesses the first rule, it will assign the role of 'Executive' to the subject (user) if their `title` attribute contains either 'Director', 'Chief Executive Officer' or 'Board Member'.

Finally, another example that illustrates the flexibility provided by role enablement:

> Assign the role of **Contractor** if:
> subject's `emailAddress` attribute contains '@third-party-contractor'

With this example, ViewDS will assign the role of 'Contractor' to the user if the `emailAddress` stored in their directory entry contains the specified string.

XACML tutorials

This chapter's tutorials take you through defining and applying XACML policy to a resource:

- ViewDS PEP tutorial for ABAC
- ViewDS PEP tutorial for RBAC
- HTTP PEP tutorial

ViewDS PEP tutorial: ABAC

This tutorial takes you through how to define an XACML policy that includes attribute-based access control (ABAC). The policy will apply to an area of the Deltawing directory that is provided with the ViewDS Directory.

The tutorial includes the following stages:

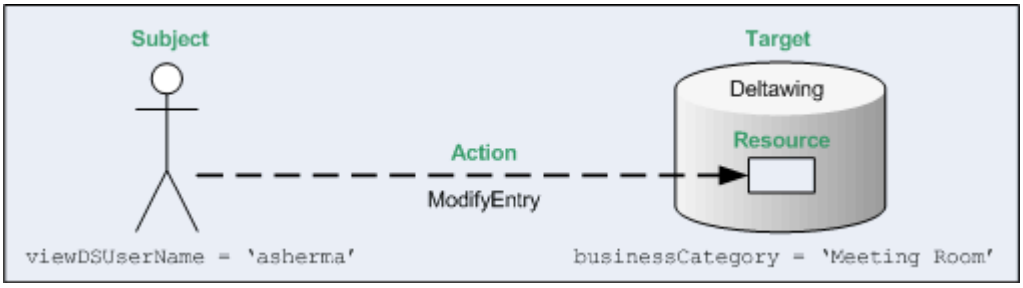
1. Requirements
2. Create an XACML Access Control Domain
3. Create an XACML policy
4. Declare XACML attributes
5. Define the first rule
6. Define the first rule's condition
7. Define the second rule
8. Define the second rule's condition
9. Activate the policy
10. Test the policy
11. Lock the policy

Requirements

This tutorial's requirement is for a policy that gives one user, Andrew Sherman, the privileges to modify meeting room entries in the Deltawing directory.

Both Andrew Sherman and a meeting room can be identified in the Deltawing directory by their entries' directory attributes:

- Andrew can be identified by his entry's `viewDSUserName` attribute which is set to 'asherma'.
- a meeting room entry can be identified by its `businessCategory` attribute which is set to 'Meeting Room'.



When a directory user (subject) attempts to modify an entry (resource), the Policy Enforcement Point (PEP) will send an authorization decision request to the Policy Decision Point. The request includes the values of directory attributes in the subject and resource entries, plus a value to identify the attempted action. These values are held in XACML attributes.

XACML attributes

Before an XACML attribute can be used by the PAP, it must first be declared in the XACML Access Control Domain.

Each declaration has a 'Label' that will appear in a rule's condition, an XACML category, and may also require a mapping to a directory attribute.

In this tutorial, the following declarations are required.

Label	XACML attribute category	XACML attribute identifier	XACML data type
User Name	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	urn:oasis:names:tc:xacml:1.0:subject:subject-id	string
Action	urn:oasis:names:tc:xacml:3.0:attribute-category:action	urn:oasis:names:tc:xacml:1.0:action:action-id	string
Business Category	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	businessCategory (urn:oid:2.5.4.15)	string

Note that an XACML attribute's category corresponds to its purpose, as shown in the illustration above.

Also note that `Business Category` is mapped to the directory attribute `businessCategory` through its XACML Attribute Identifier. However, `User Name` does not need to be mapped to a directory attribute because it is one of three values the PEP provides to identify the subject (see XACML attributes provided by the ViewDS PEP).

Rules

Two rules are required. The first will permit Andrew Sherman to modify meeting room entries in the directory. The second will permit all users to search and view entries in the directory. This is necessary because the default behaviour is to deny access within an Access Control Domain, unless explicitly permitted.

Rule 1

The first rule's target, scope, effect and condition are shown below.

Rule 1:
Target: `Deltawing`
Scope: `subtree`
Effect: `Permit` (if the condition is true)
Condition:
resource has attribute `Business Category` = 'Meeting Room' AND
subject has attribute `User Name` = 'asherma' AND
(Action = 'ModifyEntry' OR Action = 'AddType' OR
Action = 'RemoveType' OR Action = 'AddValue' OR Action = 'RemoveValue')

The rule's target will be the entry at the root of the Deltawing directory; its scope will be the entire subtree below the root entry; and its effect will be to permit access if the condition is true.

The condition will be true when the user with the `User Name` 'asherma' (subject) attempts one of the actions on a meeting room entry (resource).

Note that omitting the resource clause would make the rule more general so that it applied it to all entries in the directory.

Rule 2

The second rule's target, scope, effect and condition are shown below.

Rule 2:
Target: `Deltawing`
Scope: `subtree`
Effect: `Permit` (if the condition is true)
Condition:
Action = 'ReadEntry' OR Action = 'BrowseEntry' OR
Action = 'ReturnDN' OR Action = 'ReadType' OR
Action = 'FilterMatchType' OR Action = 'ReadValue' OR Action = 'FilterMatchValue

It has the same target and scope as the first rule. It also permits access if the condition is true.

The condition will be true when any user (subject) attempts one of the search or read actions on any directory entry (resource).

Create an XACML Access Control Domain

An XACML Access Control Domain is a specific area of a DIT that contains one or more XACML policies. The entry at the top of the domain is termed the *access control administrative point*.

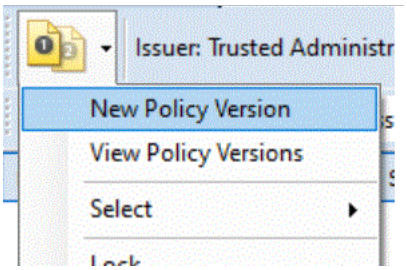
To create an XACML Access Control Domain below the `Deltawing` entry in the Deltawing directory:

1. At the bottom of the left pane, click **Server View**.
2. In the left pane, click your ViewDS server. The Status tab displays the status of your ViewDS server.
3. Ensure that the ViewDS Management Agent is connected to your ViewDS server, and that your server is running.
4. At the bottom of the left pane, click **Global DIT View**.
5. In the left pane, expand the **Deltawing** entry.
6. Right-click the **Deltawing** entry. A submenu is displayed.
7. From the submenu, click **Add XACML Access Control Domain**. The XACML AC tab is added to the right pane.

Create an XACML policy

To create the policy:

1. In the right pane, click the **XACML AC** tab and then the **Policy Versions** tab.
2. In the right pane, click the **Version Management** button followed by **New Policy Version**. The XACML Policy Version window is displayed.



3. Accept the default values by clicking **Save**. The new policy's version and status are displayed next to the Version Management button.

NOTE: The policy is marked as open, which indicates that it can be modified. Once a policy has been locked it cannot be modified. You can, however, create a new policy based on it.

Define the first rule

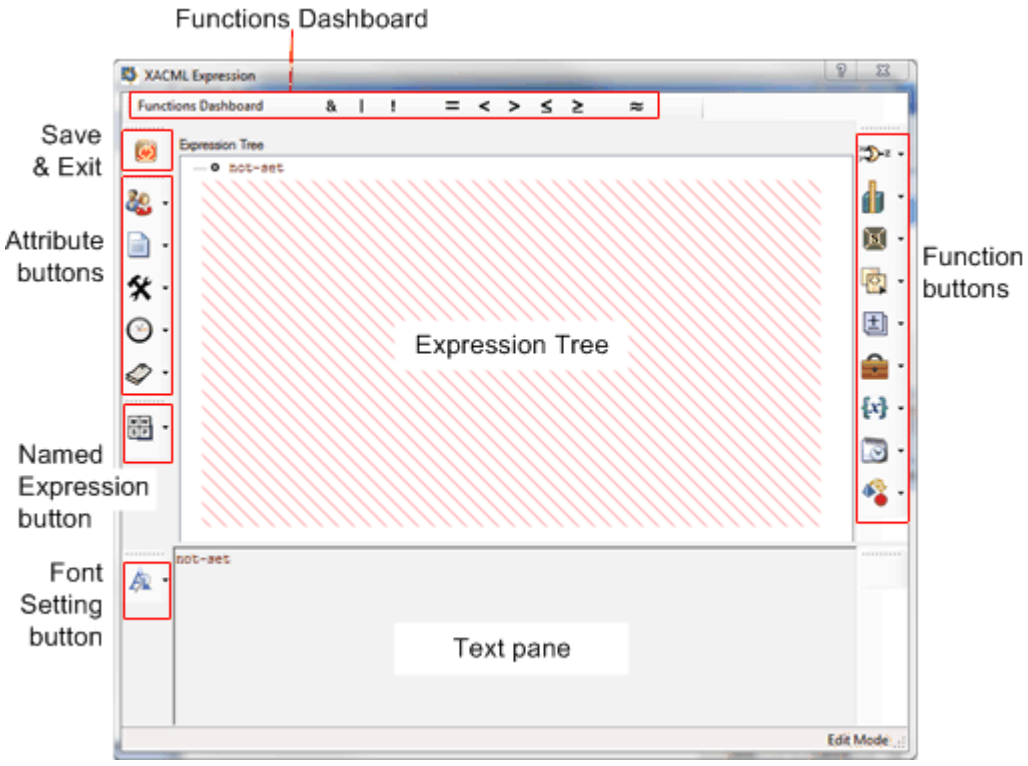
To create the first rule in the policy:

1. In the right pane, click the **Policy Versions** tab.
2. With 'ABAC Rules' and 'Access' selected in the filter boxes, click the **New** button. The XACML Rule window is displayed. It allows you to define a rule for the current policy.

3. In the **Label** box, enter `Andrew Sherman meeting room access`.
4. Enter a short **Description** of the rule, such as `ABAC that permits Andrew Sherman full access to meeting room entries`. Note that the Target is set to Deltawing and its Scope is subtree. Hence, the target is all subtrees and entries subordinate to Deltawing. Also note that the Effect is set to the default, permit.
5. Click the **Edit** button. The XACML Expression window is displayed (described below).

XACML Expression window

This window allows you to define the expressions that constitute a rule's condition.



The window has two areas:

- Expression Tree
  This is the window's main work area and allows you to build expressions in a tree format.
- Text pane
  This area shows the contents of the Expression Tree in a plain text format.

There are five sets of buttons:

- Functions Dashboard
  These buttons allow you to add one of the frequently used functions to the Expression Tree. The functions are also available through the function buttons.
- Save and Exit button
  This button allows you save the Expression Tree and exit the Expression Builder window.
- Attribute buttons
  These buttons allow you to add XACML attributes to the Expression Tree. Only the XACML attributes declared in the current Access Control Domain are available. There is a button for each category of XACML attribute: subject, resource, action, environment and other attributes.
- Font Setting button
  This button allows you to change the font for the attributes, values, functions and named expressions displayed in the text pane.
- Named Expression button
  This button allows you to add a named expressions to the Expression Tree. Only the named expressions defined in the current Access Control Domain are available.
- Function buttons
  These buttons allow you to add a function to the Expression Tree. There are nine function categories: Boolean, Relational, XPath, String, Arithmetic, Bag, Set, Date and Time, and Conversion.

Define the first rule's condition

Each rule has a condition comprising a set of expressions. The condition for the rule in this tutorial is as follows:

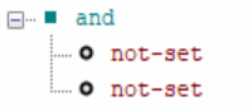
```
resource has attribute Business Category = ’Meeting Room’ AND
subject has attribute User Name = 'asherma' AND
(Action = 'ModifyEntry’ OR Action = 'AddType’ OR
Action = 'RemoveType’ OR Action = 'AddValue’ OR Action = 'RemoveValue’)
```

Each line in the condition is an expression. The three expressions are combined by Boolean 'And' functions.

Start with a Boolean 'And'

To apply a Boolean 'And' function:

- In the XACML Expression window, drag and drop the **&** from the Functions Dashboard to the node at the top of the expression tree. The function is displayed in the expression tree with two empty nodes below it.

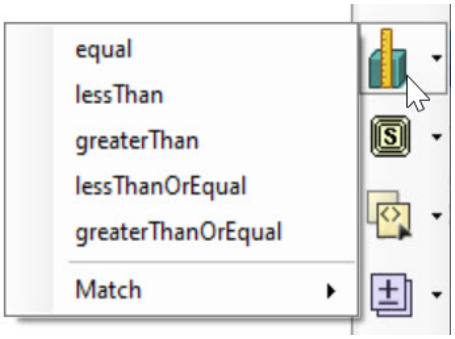


Note: To replace a function, drag and drop another function on top of it.

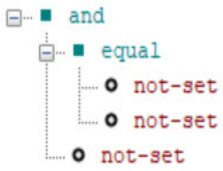
Defining the first expression

You can now define the first expression in the condition:

1. Click the **Relational Functions** button. A list of functions is displayed.

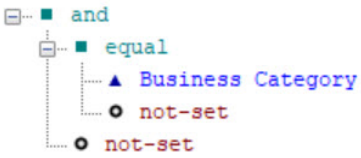


2. Drag and drop **equal** onto the first empty node in the expression tree. The equal function is added to the tree with two new empty nodes below it.

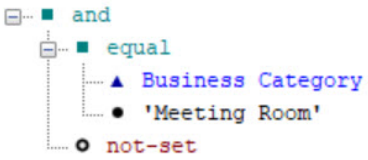


3. On the left of the window, click the **Resource Attributes** button.

4. Drag and drop **Business Category** onto the fist **not-set** node below the equal function.



5. Double-click the **not-set** node below Business Category. The XACML Value (String) window is displayed.
6. In the **Value** box, enter `Meeting Room` and then click **OK**. The string is added to the expression.

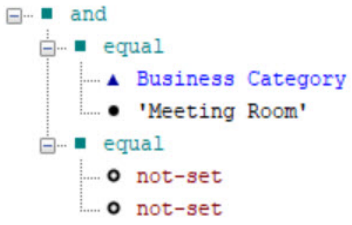


Now define the second expression in the condition.

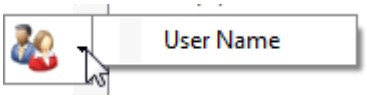
Defining the second expression

To define the second expression in the condition:

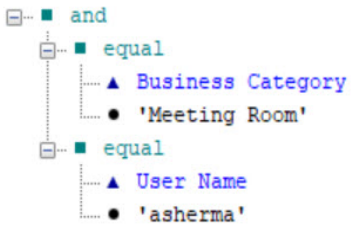
1. From the Functions Dashboard, drag and drop the **=** function onto the remaining **not-set** node. The equal function is added to the tree with two new empty nodes below it.



2. Click the **Subject Attributes** button.



3. Drag and drop **User Name** onto the fist node below the equal function.
4. Double-click the **not-set** node below User Name. The XACML Value (String) window is displayed.
5. Enter `asherma` and then click **OK**. The string is added to the expression.



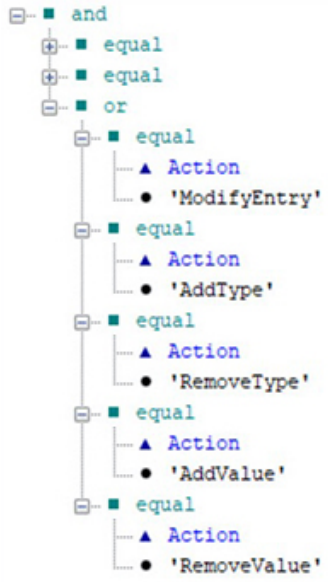
Now define the remaining expression.

Defining the third expression

To define the remaining expression:

1. Right-click the **and** function at the top of the Expression Tree, then click **Add New Argument**. A new 'not-set' node is added to the bottom of the Expression Tree.
2. From the Functions Dashboard, drag and drop the **|** function onto the new **not-set** node. The 'or' function is displayed with two new 'not-set' nodes below it.
3. In the Expression Tree, right-click the **or** function, and then click **Add New Argument**. A third 'not-set' node is displayed below the 'or' function.
4. Repeat step 3 until there is a total of five 'not-set' nodes below the **or** function in the Expression Tree.
5. From the Functions Dashboard, drag and drop the **=** function onto the first **not-set** node below the 'or' function. The equal function is added to the tree with two new 'not-set' nodes below it.
6. Click the **Actions** button. The XACML attribute 'Action' is displayed.

7. Drag and drop **Action** onto the fist **not-set** node below the equal function.
8. Double-click the **not-set** node below 'Action'. The XACML Value (Enumerated) window is displayed.
9. Choose **ModifyEntry** from the dropdown list of values and click **OK**.
10. Repeat steps 5 through 9 to add the following actions to the Expression Tree: AddType, RemoveType, AddValue, RemoveValue.



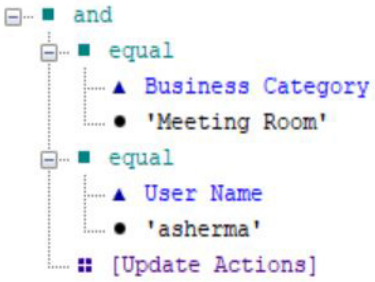
Working with named expressions

A named expression is an expression that is saved and can then be reused in different rules. If you modify a named expression, the change will affect every rule it appears in.

These steps are not required to define the first rule, but are included in this tutorial to familiarize you with named expressions:

1. In the Expression Tree, right-click the **or** function.
2. Click **Save as a Named Expression**. A window is displayed.
3. Enter `Update Actions` and then click **OK**.
4. Right-click the **or** function, then click **Delete**. The node is deleted from the tree.
5. Right-click the **and** function at the top of the Expression Tree, then click **Add New Argument**. A new 'not-set' node is added.
6. Click the **Named Expressions** button. The named expression 'Update Actions' you just created is displayed.

7. Drag and drop **Update Actions** onto the **not-set** node in the Expression Tree.



   You can view the text version of the named expression by hovering your mouse over it.

8. Click the **Save and Exit** button. The XACML Expression window closes and the condition is displayed in the Condition box of the XACML Rule window.
9. Click **Save**. The rule is added to the Rules area of the Policy Versions tab.
10. To view the named expression:
    a. In the right pane, click the **Policy Versions** tab.
    b. In the first filter box, click **Named Expressions**. The named expressions are listed in the summary area of the tab.
    c. Click the named expression and then click the **Open** button. The XACML Named Expression window is displayed.
    d. Click the **Edit** button. The named expression is displayed in the XACML Expression window.

Define the second rule

To create the second rule in the policy:

1. With **ABAC Rules** and **Access** selected in the filter boxes, click the **New** button. The XACML Rule window is displayed.
2. In the Label box, enter `Search & Read access control`.
3. Enter a short **Description** of the rule, such as `ABAC that permits all users search and read access to all entries`.
4. Click the **Edit** button.The XACML Expression window is displayed.

Define the second rule's condition

The second permits all users to search and view directory entries. It is required because the default behaviour within an Access Control Domain is to deny access, unless explicitly permitted.

The second rule's condition is as follows:

```
Action = 'ReadEntry' OR Action = 'BrowseEntry' OR
Action = 'ReturnDN' OR Action = 'ReadType' OR
Action = 'FilterMatchType' OR Action = 'ReadValue' OR
Action = 'FilterMatchValue' OR Action = 'DiscloseEntryOnError' OR
Action = 'DiscloseTypeOnError' OR Action = 'DiscloseValueOnError'
```

To define these expressions:

1. From the Functions Dashboard, drag and drop the **|** function onto the **not-set** node at the top of the Expression Tree. The 'or' function is added to the Expression Tree with two empty nodes below it.
2. In the Expression Tree, right-click the **or** function, then click **Add New Argument**. A 'not-set' node is added to the tree.
3. Repeat the above step until there are ten 'not-set' nodes.
4. From the Functions Dashboard, drag and drop the **=** function onto the first **not-set** node below the **or** function. The 'equal' function is added to the tree with two new empty nodes below it.
5. Click the **Action Attributes** button. The XACML attribute Action is displayed.
6. Drag and drop **Action** onto the fist **not-set** node below the 'equal' function.
7. Double-click the **not-set** node below **Action**. The XACML Value (Enumerated) window is displayed.
8. Choose **ReadEntry** from the dropdown list and click **OK**.
9. Repeat steps 4 and 8 in order to add the following to the remaining not-set nodes:
   - Action = BrowseEntry
   - Action = ReturnDN
   - Action = ReadType
   - Action = ReadValue
   - Action = FilterMatchValue
   - Action = FilterMatchType
   - Action = DiscloseEntryOnError
   - Action = DiscloseTypeOnError
   - Action = DiscloseValueOnError
10. Click the **Save and Exit** button, followed by **Save**.

Activate the policy

For a policy to take effect it must be activated. Only one version of a policy can be active at any time. This ensures that after writing a new version of a policy, you can activate it at an appropriate time and also have the option to roll back by activating the previous version if necessary.

To activate the policy:

1. In the **Policy Versions** tab, click **Version Management** followed by **Activate**. A warning is displayed.
2. Click **Yes**. The policy's Status is now `Active, Open`. This signifies that the rule is in use (active) but can still be modified (open).

Test the policy

You can test the policy by attempting to modify a meeting room entry through Access Presence, first as Andrew Sherma and then as another user. (For the instructions to configure for Access Presence, see *Configuring for Access Presence* in the *ViewDS Directory: Installation and Operation Guide*.)

To test the policy:

1. Open the URL: `http://host:8090/directoryservices/viewds/webdua.cgi`
2. Log on with the user name `asherma` and password `testpass`.
3. In the drop-down box, click **Function Search** and then click **Access**. The Advanced Search page is displayed.
4. In the function box, enter `meeting room` and press the return key. A list of meeting rooms is displayed.
5. Click the third meeting room in the list. The entry for the Sales Meeting Room is displayed.
6. Click **Modify**. The Modify page is displayed.
7. Modify the contents of the **Description** box and then click **Save**.
8. Log off by closing the browser session.
9. Repeat this task from step 1, logging on with the user name `rturnbu` and password `testpass`. This user will not be able to modify any entries.

ViewDS PEP tutorial: RBAC

This tutorial takes you through how to define an XACML policy that includes role-based access control (RBAC). The policy will apply to an area of the Deltawing directory that is provided with the ViewDS Directory.

It involves adding an RBAC rule to the XACML Access Control Domain that was defined during the previous tutorial.

> **NOTE**: Before starting this tutorial, you should have read Attribute versus role-based access control and completed the Tutorial: Attribute-based access controls.

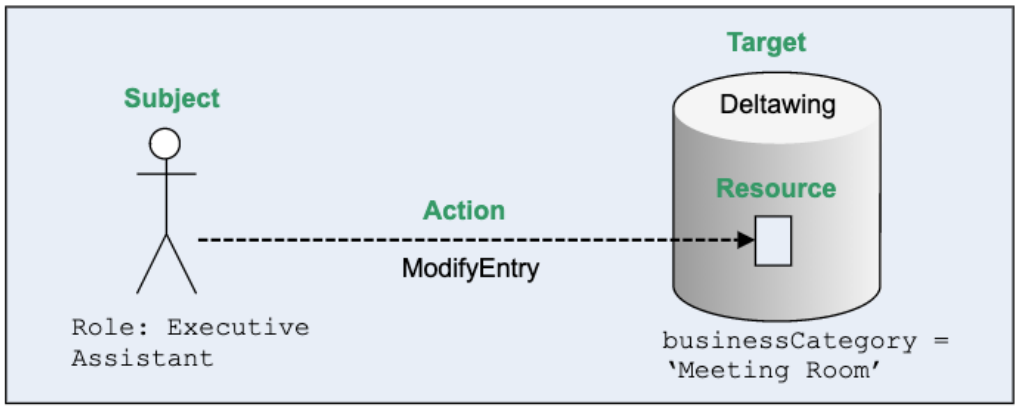
This tutorial has the following stages:

1. Requirements
2. Create a new XACML policy version
3. Define the role hierarchy
4. Define a role attribute
5. Define the RBAC rule
6. Activate the policy
7. Test the policy
8. Lock the policy

Requirements

This tutorial involves creating a new version of the XACML policy that was defined in the Tutorial: Attribute-based access controls.

The requirements include an RBAC rule that permits the role of 'Executive Assistant' to modify meeting room entries in the Deltawing directory.



The policy also requires a role hierarchy that defines that a 'Chief Executive Officer' inherits permissions from an 'Executive Assistant'.

XACML attributes

An XACML attribute must be declared before it can be used in an XACML rule.

Each declaration includes a label that will appear in a rule, plus an XACML category, identifier, and data type. An XACML attribute can also be mapped to a directory attribute.

This tutorial's rule will use XACML attributes declared in the Tutorial: Attribute-based access controls (Action and Business Category) plus an XACML attribute with the following definition:

Label	XACML attribute category	XACML attribute identifier	XACML data type	Directory attribute
Role	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	urn:oasis:names:tc:xacml:2.0:subject:subject:role	string	title

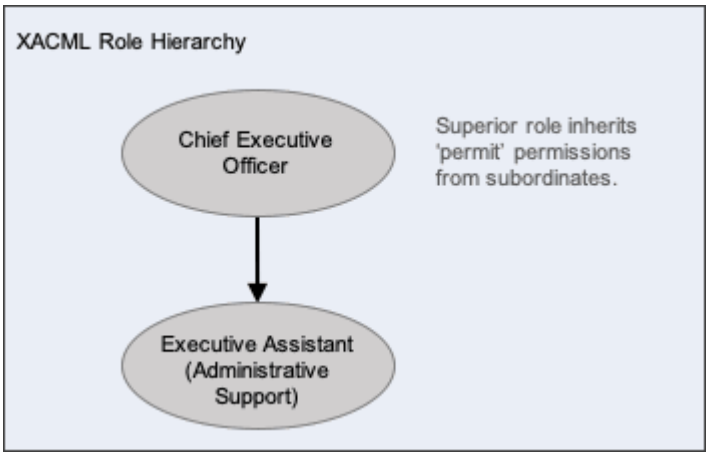
The definition maps the XACML attribute to a directory attribute called `title`.

The value of `title` in a user's directory entry identifies their role. This means that if a user's `title` is 'Executive Assistant', for example, then their role is also 'Executive Assistant'.

Role hierarchy

A role hierarchy is required that includes the role 'Chief Executive Officer' which inherits permissions from the role 'Executive Assistant'.

The 'Executive Assistant' role should also have an alternative name declared. The effect will be that all users with the `title` 'Administrative Support' or 'Executive Assistant' will have the same role and will be granted the same permissions.



RBAC rule

One RBAC rule is required. Its target, scope, effect, role and condition are shown below.

RBAC Rule:
Target: `Deltawing`
Scope: `subtree`
Effect: `Permit`
Role: `Executive Assistant`
Condition:
resource has attribute `Business Category` = 'Meeting Room' AND

```
(Action = 'ModifyEntry' OR Action = 'AddType' OR
Action = 'RemoveType' OR Action = 'AddValue' OR Action = 'RemoveValue')
```

The rule's target will be the entry at the root of the Deltawing directory; its scope will be the entire subtree below the root entry; and its effect will be to permit access if the user's role is 'Executive Assistant' and the condition is true.

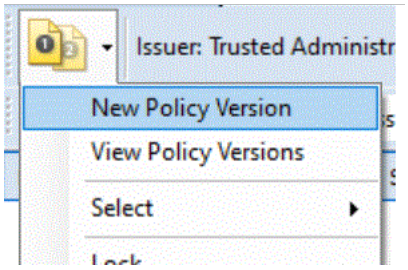
The condition will be true when an 'Executive Assistant' attempts one of the actions on a meeting room entry.

NOTE: A prerequisite of this tutorial is to have completed the Tutorial: Attribute-based access controls. This is because it includes an ABAC rule that permits all users to search and view all directory entries.

Create a new XACML policy version

To create a new version of the policy that was declared and then locked during the previous tutorial:

1. In the left pane, click the **Deltawing** entry at the top of the DIT. The XACML AC tab is displayed in the right pane. This is because the entry is the administrative point for the XACML Access Control Domain created in the previous tutorial.
2. In the right pane, click the **XACML AC** tab and then the **Policy Versions** tab.
3. Click the **Version Management** button followed by **New Policy Version**. The XACML Policy Version window is displayed.

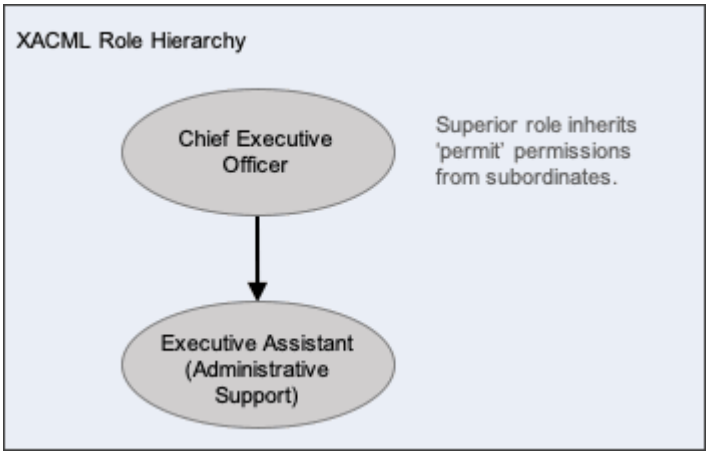


4. Accept the default values by clicking **Save**. A new policy version with the status of open is displayed next to the Version Management button.

Define the role hierarchy

Defining a role hierarchy involves two broad steps. First, declare the roles, and then declare the hierarchical relationship between them.

The roles required for this tutorial are 'Chief Executive Officer' are 'Executive Assistant'. However, you will also declare an alternative name of 'Administrative Support' for 'Executive Assistant'. This will mean that the same role and permissions will be granted to any user whose `title` is 'Administrative Support' or 'Executive Assistant'.



To declare the 'Chief Executive Officer' role:

1. In the right pane, click the **Roles** tab.
2. At the bottom of the Roles tab, click **New**. The New XACML Role window is displayed.
3. Click **Add**. The XACML Role Name window is displayed.
4. Enter the role name `Chief Executive Officer`, then click **OK** followed by **Save**. The role is added to the Roles tab.

To declare the second role:

1. Click **New**. The New XACML Role window is displayed.
2. Click **Add**. The XACML Role Name window is displayed.
3. Enter the role name `Executive Assistant`, then click **OK**.
4. Repeat steps 2 and 3 to add another name for the same role, `Administrative Support`.
5. Click **Save**. Two names for the single role are added to the Roles tab.

To declare that the 'Chief Executive Officer' should inherit permissions:

1. Right-click **Chief Executive officer** and then click **Open**. The XACML Role window is displayed.
2. Below the **Directly inherits permissions from** box, click **Edit**. The XACML Role Selection window is displayed.
3. Click either role name followed by the right-arrow button. Both role names are moved to the box on the right.
4. Click **OK** followed by **Save**.

Declare a role attribute

To declare the XACML attribute for `Role`:

1. In the right pane, click the **Attributes** tab.
2. At the bottom of the right pane, click **New**. The XACML Attribute window is displayed.
3. In the **Label** box, enter `Role`. This is the name that will appear in the rule.

4. In the **Category** box, click **urn:oasis:names:tc:xacml:1.0:subject-category:access-subject**.
5. In the **Identifier** box, click **urn:oasis:names:tc:xacml:2.0:subject:role**.
6. In the **Directory Attribute** box, click **title**.
7. Click **Save**. The Attributes tab displays the new attribute.

Also see XACML attributes provided by the ViewDS PEP.

NOTE: Every attribute in an XACML domain must have a unique combination of Category, Identifier and Data Type.

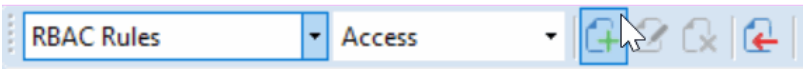
Define the RBAC rule

The required RBAC rule is as follows:

RBAC Rule:
Target: `Deltawing`
Scope: `subtree`
Effect: `Permit`
Role: `Executive Assistant`
Condition:
resource has attribute `Business Category` = 'Meeting Room' AND
(Action = 'ModifyEntry' OR Action = 'AddType' OR
Action = 'RemoveType' OR Action = 'AddValue' OR Action = 'RemoveValue')

To create an RBAC rule:

1. In the right pane, click the **Policy Versions** tab.
2. With **RBAC Rules** and **Access** selected in the filter boxes, click the **New** button. The XACML Rule window is displayed.



3. In the **Label** box, enter `Exec Assistant & Admin Support meeting room access`.
4. Enter a short **Description** of the rule, such as `RBAC rule that permits Exec Assistant & Admin Support full access to meeting room entries.` Note that the Target is Deltawing and its Scope is subtree. Hence, the target is all subtrees and entries subordinate to Deltawing. Also note that the Effect is set to the default, permit.

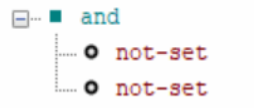
To define the role to which the rule applies:

1. Click the **Edit** button next to the **Role Expression** box, The XACML Role Expression window is displayed. The names of the roles you declared earlier are listed on the left.

2. Click **Executive Assistant** followed by the right arrow. The role name is moved to the box on the right.
3. Click **OK**. The role is added to the Role Expression box. Note that this name applies to a single role, which also has the name 'Administrative Support' assigned to it.

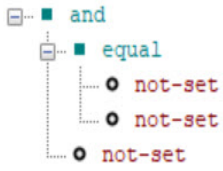
The remaining expressions are defined through the XACML Expression window:

1. Click the **Edit** button next to the **Condition** box. The XACML Expression window is displayed.
2. Drag and drop the **&** from the Functions Dashboard to the node at the top of the expression tree. The function is displayed in the expression tree with two empty nodes below it.



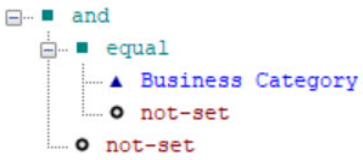
Note: To replace a function, drag and drop another function on top of it.

3. Drag and drop the **=** from the Functions Dashboard onto the first **not-set** node in the expression tree.

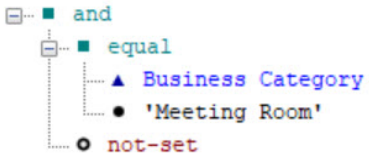


4. On the left of the window, click the **Resource Attributes** button.

5. Drag and drop **Business Category** onto the fist **not-set** node below the equal function.

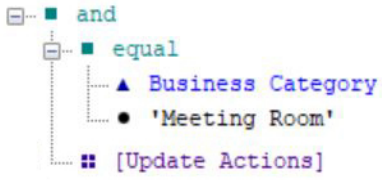


6. Double-click the **not-set** node below Business Category. The XACML Value (String) window is displayed.
7. In the **Value** box, enter `Meeting Room` and then click **OK**. The string is added to the expression.



8. Click the **Named Expressions** button. The named expression 'Update Actions' that was created during the previous tutorial is displayed.

9. Drag and drop **Update Actions** onto the **not-set** node in the Expression Tree.



   You can view the text version of the named expression by hovering your mouse over it.

10. Click the **Save and Exit** button. The XACML Expression window closes and the condition is displayed in the Condition box of the XACML Rule window.
11. Click **Save**. The rule is added to the Rules area of the Policy Versions tab.

Activate the policy

For a policy to take effect it must be activated. Only one version of a policy can be active at any time. This ensures that after writing a new version of a policy, you can activate it at an appropriate time and also have the option to roll back by activating the previous version if necessary.

To activate the policy:

1. In the **Policy Versions** tab, click **Version Management** followed by **Activate**. A warning is displayed.
2. Click **Yes**. The policy's Status is now `Active, Open`. This signifies that the rule is in use (active) but can still be modified (open).

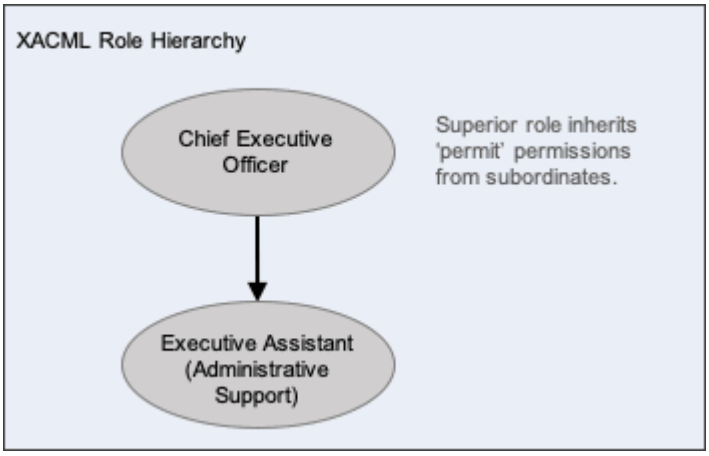
Test the policy

You can test the policy by attempting to modify a meeting room entry through Access Presence. (For the instructions to configure for Access Presence, see *Configuring for Access Presence* in the *ViewDS Directory: Installation and Operation Guide*.)

Testing involves logging onto Access Presence as the following users:

Deltawing user	Role (title)	Username	Password
Maria Guglielmino	Administrative Support	mguglie	testpass
Robert Turnbull	Executive Assistant	rturnbu	testpass
Margaret Hunter	Chief Executive Officer	mhunter	testpass
Karen Johannesen	Director	kjohann	testpass

All the above users, except Karen Johannesen, will be able to modify a meeting room entry. Note that Margaret Hunter has the role 'Chief Executive Officer', which inherits permissions through the role hierarchy.



To test the policy:

1. Open the URL: `http://host:8090/directoryservices/viewds/webdua.cgi`
2. Enter a username and password `testpass`.
3. In the drop-down box, click **Function Search** and then click **Access**. The Advanced Search page is displayed.
4. In the function box, enter `meeting room` and press the return key. A list of meeting rooms is displayed.
5. Click a **meeting room** entry. The entry's details are displayed. If the user has permission to modify the entry, then the Modify button is displayed.
6. Log off by closing the browser session.
7. Repeat this task for users with different roles.

Lock the policy

Once you lock a policy you cannot delete or modify it. You can, however, create a new policy based on an existing policy by clicking the **New** button in the Policy Versions tab.

To lock the policy:

1. In the **Policy Versions** tab, click **Version Management** followed by **Lock**. A warning is displayed.
2. Click **OK**. The policy's Status is now `Active, Locked`.

HTTP PEP tutorial

This tutorial takes you through how to apply an XACML policy, which includes attribute-based access control (ABAC), to web pages hosted by either an Apache or IIS web server.

The tutorial includes the following stages:

1. Requirements
2. Set the policy base object
3. Create tutorial files and configure the web server
4. Create an XACML Access Control Domain
5. Create an XACML policy
6. Declare XACML attributes
7. Define the first rule
8. Define the second rule
9. Define the third rule
10. Activate the policy
11. Test the policy
12. Lock the policy

Requirements

A policy is required to control user access to a set of web pages with HTTP authentication and hosted by an Apache or IIS server.

The set of web pages is as follows:

```
/xacml/index.html
/xacml/restricted/index.html
/xacml/restricted/restricted.html
/xacml/secret/index.html
/xacml/secret/secret.html
```

HTTP authentication is also required for users with the following usernames: ‘mhunter’, ‘asherma’ and ‘rturnbu’. All should have the same password: ‘testpass’.

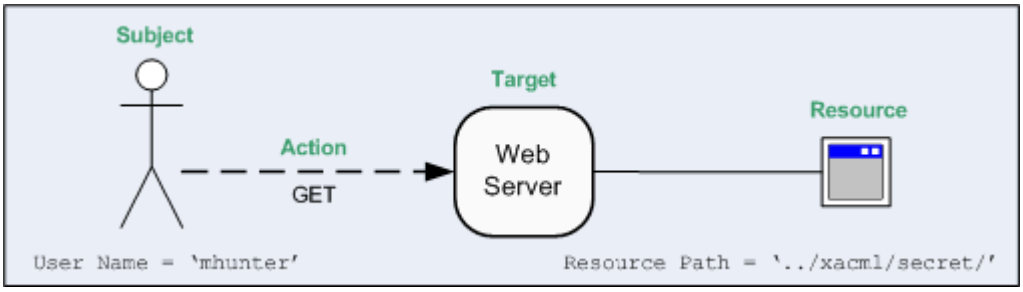
The policy will control access as follows:

1. Permit all users access to all `index.html` files

2. Permit only 'mhunter’ and 'asherma’ access to `restricted.html`

3. Permit only 'mhunter’ access to `secret.html`

The last requirement is illustrated below.



When a user (subject) attempts to access a webpage (resource), the Apache Policy Enforcement Point (PEP) will send an authorization decision request to the Policy Decision Point (PDP). The request includes values that identify, among other things, the subject, the resource and the attempted action. These values are held in XACML attributes.

XACML attributes

Before an XACML attribute can be used by the PAP, it must first be declared in the XACML Access Control Domain.

Each declaration has a 'Label' that will appear in a rule's condition, and a XACML category, identifier and type. The combination of category, identifier and type dictates the value returned by the PEP and assigned to the XACML attribute.

The XACML attribute declarations required in this tutorial are as follows.

Label	XACML attribute category	XACML attribute identifier	XACML data type
User Name	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	urn:oasis:names:tc:xacml:1.0:subject:subject-id	string
URL Path	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	http://viewds.com/http/resource/path	string

NOTE: An XACML attribute's category corresponds to its purpose, as shown in the previous illustration.

Rules

Each rule has a target, scope, effect and condition. The effect of all three rules in this tutorial will be to permit access, and their targets will be either paths or webpages. The target and scope are arbitrary as they only apply to the internal PEP.

The effect (permit) and condition for each rule in this tutorial are shown below.

Rule 1:
Permit (if the following condition is true)
`URL Path` contains 'index.html'
Rule 2:
Permit (if the following condition is true)
`URL Path` contains 'restricted.html' AND
(`User Name` = 'asherma' OR `User Name` = 'mhunter')

Rule 3:
Permit (if the following condition is true)
`URL Path` contains 'secret.html' AND
`User Name` = 'mhunter'

Set the policy base object

The policy base object is the root of the directory subtree where the PDP searches for XACML policy. In this tutorial, the policy base object is the Deltawing entry:

1. In the ViewDS Management Agent, click **Server View**.
2. In the left pane, click the appropriate DSA.
3. In the right pane, click the **XACML Config** tab.
4. Click **Browse** next to the **Policy Base Object** box. The DIT Browser is displayed.
5. Click the **Deltawing** entry (the first entry below the **Root**) and then click **OK**.
6. At the bottom of the **XACML Config** tab, click **Set XACML Configuration**.

Create tutorial files and configure the web server

Next, copy the files and set up your web server for this tutorial:

1. Create the following directories and files in the appropriate location for your web server (for example, below the `htdocs` directory for Apache, or below `wwwroot` for IIS):
   ```
   /xacml/index.html
   /xacml/restricted/index.html
   /xacml/restricted/restricted.html
   /xacml/secret/index.html
   ```
2. Configure your web server for HTTP authentication on the above files. Apply HTTP authentication for users with the following usernames: ‘mhunter’, ‘asherma’ and ‘rturnbu’. All should have the same password: ‘testpass’. For information about configuring a web server for a PEP, see either Deploying the Apache PEP or Deploying the IIS PEP.

Create an XACML Access Control Domain

An XACML Access Control Domain is an area of a DIT containing XACML policy. The entry at the top of the domain is the *access control administrative point*.

To create an XACML Access Control Domain:

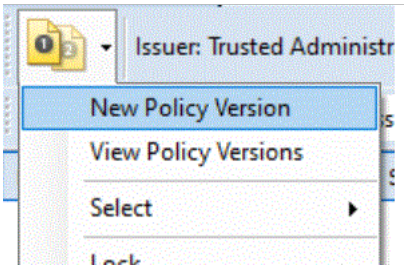
1. At the bottom of the left pane, click **Server View**.
2. In the left pane, click your ViewDS server. The Status tab displays the status of your ViewDS server.
3. Ensure that the ViewDS Management Agent is connected to your ViewDS server, and that your server is running.
4. At the bottom of the left pane, click **Global DIT View**.

5. In the left pane, expand the **Deltawing** entry.
6. Right-click the **Deltawing** entry. A submenu is displayed.
7. From the submenu, click **Add XACML Access Control Domain**. The XACML AC tab is added to the right pane.

Create an XACML policy

To create the policy:

1. In the right pane, click the **XACML AC** tab and then the **Policy Versions** tab.
2. In the right pane, click the **Version Management** button followed by **New Policy Version**. The XACML Policy Version window is displayed.



3. Accept the default values by clicking **Save**. The new policy's version and status are displayed next to the Version Management button.

> **NOTE:** The policy is marked as open, which indicates that it can be modified. Once a policy has been locked it cannot be modified. You can, however, create a new policy based on it.

Declare XACML attributes

To declare the XACML attributes for the tutorial's policy:

1. In the right pane, Click the **Attributes** tab.
2. At the bottom of the right pane, click **New**. The XACML Attribute window is displayed.
3. In the **Label** box, enter `URL Path`. This is the name that will appear in the rule.
4. In the **Category** box, click **urn:oasis:names:tc:xacml:3.0:attribute-category:resource**. The Identifier box defaults to `urn:oasis:names:tc:xacml:1.0:microprocessor-id`, and the Data Type defaults to `string`.
5. In the **Identifier** box, delete the default value and enter the following: `http://viewds.com/http/resource/path`
6. Click **Save**. The XACML attribute is added to the Attributes tab.
7. Repeat the above steps to declare the following XACML attribute:

Label	Category	Identifier	Data Type
User Name	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	urn:oasis:names:tc:xacml:1.0:subject:subject-id	string

Note that the above information is also in XACML attributes provided by an HTTP PEP.

> **NOTE:** Every attribute in an XACML domain must have a unique combination of Category, Identifier and Data Type.

Define the first rule

To define the first rule:

1. In the right pane, click the **Policy Versions** tab.
2. With **ABAC Rules** and **Access** selected in the filter boxes, click the **New** icon. The **XACML Rule** window is displayed. It allows you to define a rule for the current policy.
3. In the **Label** box, enter `Access to index.html`.
4. Optionally, enter a longer **Description** of the rule.
5. Click the **Edit** button. The XACML Expression window is displayed.

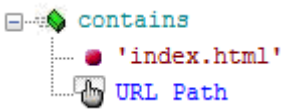
Defining the condition

Each rule has a condition comprising one or more expressions declared in an expression tree.

The condition for the first rule in this tutorial has the following expression:

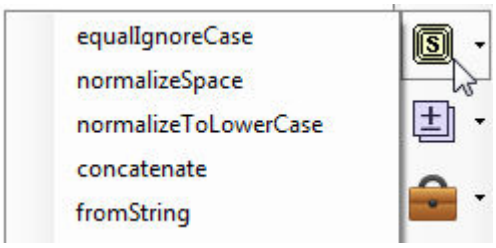
> URL Path contains 'index.html'

Every expression has a function and XACML attributes. The function is contains and the XACML attribute is Resource Path, and is represented in the expression tree as follows:

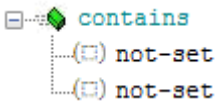


To define the first rule's condition:

1. Click the **String Functions** button. A list of functions is displayed.

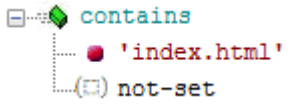


2. Drag and drop the **contains** function onto the **not-set** node in the expression tree. The `contains` function is added to the tree with two not-set nodes below it.

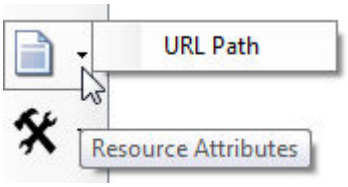


3. Double-click the first **not-set** node. The String Editor window is displayed.

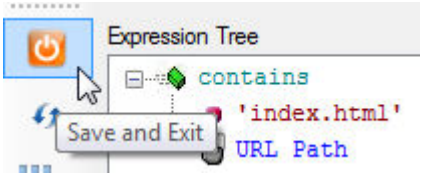
4. In the Value box, enter `index.html` and click **OK**.



5. Click the **Resource Attributes** button.



6. Drag and drop **URL Path** onto the remaining **not-set** node.
7. Click **Save and Exit**. The XACML Expression window closes and the XACML Rule window is displayed.



8. Click **Save**. The rule is displayed in the Policy Versions tab.

Define the second rule

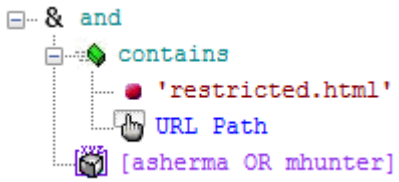
The second rule's condition is as follows:

```
URL Path contains 'restricted.html' AND
(User Name = 'asherma' OR User Name = 'mhunter')
```

The first expression is very similar to the first rule. The second is slightly more complex, and for the sake of an example you will define it as a named expression.

A named expression is an expression that is saved and can then be reused in different rules. If you modify a named expression, then the change will affect every rule it appears in.

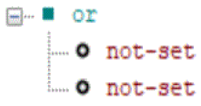
The first expression and the named expression will be tied together by a Boolean 'and' function to form the second rule.



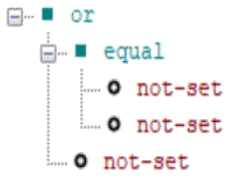
To define the named expression

1. In the first filter box in the **Policy Versions** tab, click **Named Expressions**. The named expressions are listed in the summary area of the tab.
2. Click the **New** icon. The XACML Named Expression window is displayed.

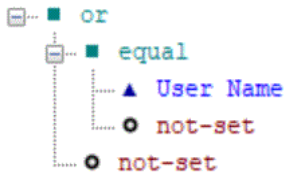
3. In the **Name** box, enter `asherma` OR `mhunter`.
4. Click the **Edit** button. The XACML Expression window is displayed.
5. Drag and drop the **|** function from the Functions Dashboard onto the **not-set** node at the top of the Expression Tree. The function is added to the expression tree with two empty nodes below it.



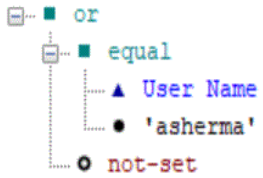
6. Drag and drop the **=** function from the Functions Dashboard onto the first **not-set** node. The function is added to the expression tree with two empty nodes below it.



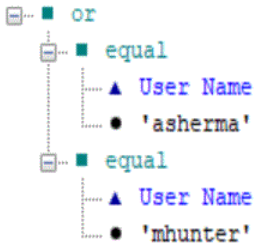
7. Click the **Subject Attributes** button, then drag and drop **User Name** onto the first **not-set** node below the **= equal** function.



8. Double-click the **not-set** node below **User Name**. The String Editor window is displayed.
9. In the Value box, enter `asherma` and click **OK**.



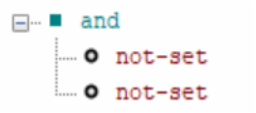
10. Repeat steps 7 through 10 above so that the Expression Tree is as follows:



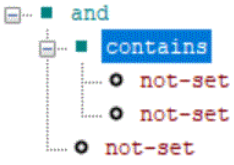
11. Click the**Save and Exit** button.
12. Click **Save**.

To define the second rule

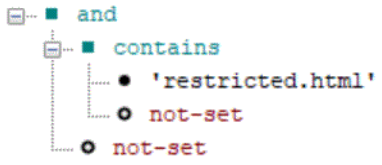
1. With **ABAC Rules** and **Access** selected in the filter boxes, click the **New** icon. The XACML Rule window is displayed.
2. In the **Label** box, enter `Access to restricted.html`.
3. Click the **Edit** button. The XACML Expression window is displayed.
4. Drag and drop the **&** function from the Functions Dashboard onto the **not-set** node at the top of the Expression Tree. The function is added to the expression tree with two empty nodes below it.



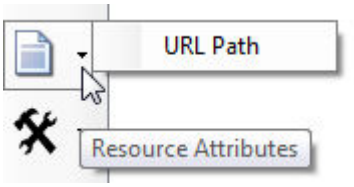
5. Click the **String Functions** button. A list of functions is displayed.
6. Drag and drop the **contains** function onto the first **not-set** node in the expression tree. The function is added to the tree with two not-set nodes below it.



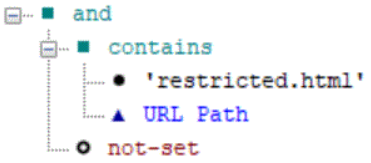
7. Double-click the first **not-set** node. The XACML Value (String) window is displayed.
8. In the **Value** box, enter `restricted.html` and click **OK**.



9. Click the **Resource Attributes** button.

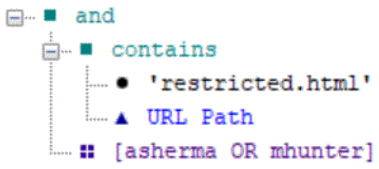


10. Drag and drop **URL Path** onto the **not-set** node below `restricted.html`.



11. Click the **Named Expressions** button.

12. Drag and drop **asherma OR mhunter** onto the remaining **not-set** node.



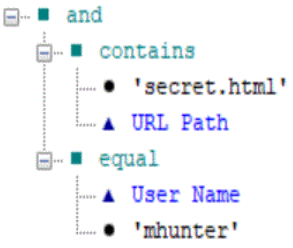
13. Click the **Save and Exit** button.
14. Click the **Save** button.

Define the third rule

The third rule's condition is as follows:

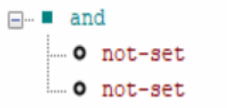
```
URL Path contains 'secret.html' AND
User Name = 'mhunter'
```

It is defined in the expression tree as follows:

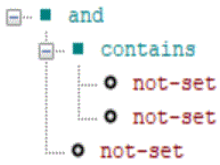


To define the rule:

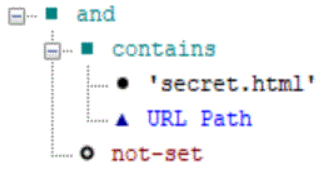
1. With **ABAC Rules** and **Access** selected in the filter boxes, click the **New**button. The XACML Rule window is displayed.
2. In the **Label** box, enter `Access to secret.html`.
3. Click the **Edit** button. The XACML Expression window is displayed.
4. Drag and drop the **&** function from the Functions Dashboard onto the **not-set** node at the top of the Expression Tree. The function is added to the expression tree with two empty nodes below it.



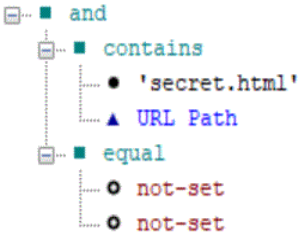
5. Click the **String Functions** button.
6. Drag and drop the **contains** function onto the first **not-set** node in the expression tree. The function is added to the tree with two not-set nodes below it.



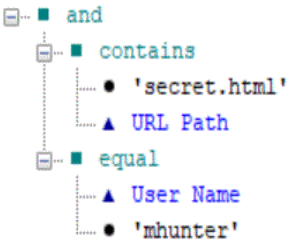
7. Double-click the first **not-set** node. The String Editor window is displayed.
8. In the **Value** box, enter `secret.html` and click **OK**.
9. Click the **Resource Attributes** button.
10. Drag and drop **URL Path** onto the **not-set** node below `secret.html`.



11. Drag and drop the **=** function from the Functions Dashboard onto the remaining **not-set** node. The function is added to the expression tree with two empty nodes below it.



12. Click the **Subject Attributes** button, then click and drag **User Name** onto the first not-set node below the `= equal` function.
13. Double-click the **not-set** node below **User Name**. The String Editor window is displayed.
14. In the **Value** box, enter `mhunter` and click **OK**.



15. Click the **Save and Exit** button.
16. Click **Save**.

Activate the policy

For a policy to take effect it must be activated. Only one version of a policy can be active at any time. This ensures that after writing a new version of a policy, you can activate it at an appropriate time and also have the option to roll back by activating the previous version if necessary.

To activate the policy:

1. In the **Policy Versions** tab, click **Version Management** followed by **Activate**. A warning is displayed.
2. Click **Yes**. The policy's Status is now `Active, Open`. This signifies that the rule is in use (active) but can still be modified (open).

Test the policy

You can test the policy by attempting to access different pages and logging on as different users when prompted.

For example, you should be able to access:

- `http://server/xacml/index.html`
  with the user name 'rturnbu'
- `http://server/xacml/restricted/restricted.html`
  with the user name 'asherma'
- `http://server/xacml/secret/secret.html`
  with the user name 'mhunter'

But you should be unable to access:

- `http://server/xacml/secret/secret.html`
  with the user name 'asherma'
- `http://server/xacml/secret/secret.html`
  with the user name 'rturnbu'
- `http://server/xacml/restricted/restricted.html`
  with the user name 'rturnbu'

Lock the policy

Once you lock a policy you cannot delete or modify it. You can, however, create a new policy based on an existing policy by clicking the **New** button in the Policy Versions tab.

To lock the policy:

1. In the **Policy Versions** tab, click **Version Management** followed by **Lock**. A warning is displayed.
2. Click **OK**. The policy's Status is now `Active, Locked`.

XACML attributes provided by a PEP

This appendix describes the attributes provided by each Policy Enforcement Point (PEP):

XACML attributes provided by an HTTP PEP

The attributes are included in an authorization decision request if the corresponding information is available in the HTTP server request context. They can be declared and included in an XACML policy.

They are in the following XACML attribute categories:

- Access-subject category
- Action category
- Environment category
- Resource category
- Requesting-machine category

Access-subject category

These attributes are in the XACML category:

`urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

The PEP includes the following values in an authorization decision request to identify the subject (the user attempting to access a site, page or application).

Value	XACML attribute identifier	XACML data type
HTTP authenticated user identifier	urn:oasis:names:tc:xacml: 1.0:subject:subject-id	string
HTTP authentication mechanism	http://viewds.com/http/subject/auth-type	string
HTTP server time (with timezone)	urn:oasis:names:tc:xacml: 1.0:subject:request-time	dateTime
HTTP browser host name	http://viewds.com/http/resource/hostname	string
HTTP browser IP address	http://viewds.com/http/subject/address	string

Action category

This attribute is in the XACML category:

`urn:oasis:names:tc:xacml:3.0:attribute-category:action`

There is one attribute that identifies the action being attempted by a subject on a resource.

Value	XACML attribute identifier	XACML data type
HTTP request method	urn:oasis:names:tc:xacml:1.0:action: action-id	string

Environment category

These attributes are in the XACML category:

`http://viewds.com/http/environment/redirect-uri`

Value	XACML attribute identifier	XACML data type
Redirection page's query string	http://viewds.com/http/environment/ redirect-query	string
Redirection page's URL	http://viewds.com/http/environment/ redirect-uri	anyURI

Resource category

These attributes are in the XACML category:

`urn:oasis:names:tc:xacml:3.0:attribute-category:resource`

The PEP includes the following values in an authorization decision request to identify the resource (the site, page or application that the subject is attempting to access).

Value	XACML attribute identifier	XACML data type
URL host name	http://viewds.com/http/resource/hostname	string
URL	urn:oasis:names:tc:xacml:1.0:resource: resource-id	anyURI
File/resource referenced by URL	urn:oasis:names:tc:xacml:1.0:resource: resource-id	string
URL scheme	http://viewds.com/http/resource/scheme	integer
URL port number	http://viewds.com/http/resource/port	string
URL path information	http://viewds.com/http/resource/path	string
URL query string	http://viewds.com/http/resource/query	string
URL fragment	http://viewds.com/http/resource/fragment	string

Requesting-machine category

These attributes are in the XACML category:

`urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

Value	XACML attribute identifier	XACML data type
HTTP server host name	http://viewds.com/http/subject/hostname	string
HTTP server IP address	http://viewds.com/http/subject/address	string

XACML attributes provided by the ViewDS PEP

This topic describes the XACML attributes that the ViewDS Policy Enforcement Point (PEP) includes in an authorization decision request.

These attributes can be declared in an XACML Access Control Domain and then used within a policy to identify the subject, resource and action, for example.

The ViewDS PEP generates authorization decision requests that include the following XACML attribute categories:

- Action category
- Access-subject category
- Resource category

Action category

The attribute is in the XACML category:

`urn:oasis:names:tc:xacml:3.0:attribute-category:action`

The attribute identifies the action being attempted by a subject (directory user) on a resource. A resource can be one of the following:

- directory entry
- attribute type
- attribute value

Directory entry

Possible values	XACML attribute identifier	XACML data type
ReadEntry BrowseEntry AddEntry RemoveEntry ModifyEntry RenameEntry ExportEntry ImportEntry ReturnDN DiscloseEntryOnError AssertTrust	urn:oasis:names:tc: xacml:1.0: action:action-id	string

Attribute type

Possible values	XACML attribute identifier	XACML data type
ReadEntry CompareType AddType RemoveType FilterMatchType DiscloseTypeOnError	urn:oasis:names:tc: xacml:1.0: action:action-id	string

Attribute value

Possible values	XACML attribute identifier	XACML data type
ReadValue CompareValue AddValue RemoveValue FilterMatchValue DiscloseTypeOnError	urn:oasis:names:tc: xacml:1.0: action:action-id	string

Access-subject category

These attributes are in the XACML category:

`urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

The PEP includes the following values in an authorization decision request.

Value	XACML attribute identifier	XACML data type
The authenticated user's Directory Name (DN), which the PEP obtains from the user's authentication information.	urn:oasis:names:tc:xacml: 1.0:subject:subject-id	X500Name
The `viewDSUserName` attribute in the subject's directory entry.		string
The attribute identified by the `RFC822 Name Attribute` (see the XACML Config tab in the ViewDS Management Agent).		rfc822Name

Resource category

These attributes are in the XACML category:

`urn:oasis:names:tc:xacml:3.0:attribute-category:resource`

The PEP includes the following value in an authorization decision request.

Value	XACML attribute identifier	XACMLdata type
The Directory Name (DN) of the resource.	urn:oasis:names:tc:xacml: 1.0:resource:resource-id	X500Name

Operational attributes

This appendix describes the following operational attributes associated with Access Sentinel:

- viewDSXACMLSubtreePolicy
- viewDSXACMLEntryPolicy
- viewDSXACMLAttributePresentation
- viewDSXACMLPolicyVersion
- viewDSXACMLNamedExpression
- viewDSXACMLActivePolicy
- viewDSXACMLConfiguration

For information about manipulating operational attributes using the ViewDS Stream DUA tool, see the *ViewDS Directory Server: Technical Reference Guide*.

viewDSXACMLSubtreePolicy

```
viewDSXACMLSubtreePolicy ATTRIBUTE ::= {
  WITH SYNTAX XACMLPolicy
  EQUALITY MATCHING RULE viewDSXACMLPolicyMatch
  SINGLE VALUE TRUE
  USAGE directoryOperation
  ID id-viewds-aca-XACMLSubtreePolicy
}
```

The attribute is stored in an object class, which is a sub-entry located below the administrative point.

```
viewDSXACMLSubtreePolicySubentry OBJECT-CLASS ::= {
  KIND auxiliary
  MUST CONTAIN { viewDSXACMLSubtreePolicy }
  ID id-viewds-sc-XACMLSubtreePolicySubentry
}
```

The viewDSXACMLSubtreePolicy attribute is automatically indexed for the viewDSXACMLPolicyMatch matching rule.

viewDSXACMLEntryPolicy

This operational attribute stores an XACML policy that applies to an Access Control Domain whose administrative point is a single entry.

```
viewDSXACMLEntryPolicy ATTRIBUTE ::= {
  WITH SYNTAX  XACMLPolicy
  EQUALITY MATCHING RULE  viewDSXACMLPolicyMatch
  SINGLE VALUE  TRUE
  USAGE  directoryOperation
  ID  id-viewds-aca-XACMLSubtreePolicy
}
```

The attribute is stored in an object class, which is a subentry located below the administrative point.

```
viewDSXACMLEntryPolicySubentry OBJECT-CLASS ::= {
  KIND  auxiliary
  MUST CONTAIN  { viewDSXACMLEntryPolicy }
  ID  id-viewds-sc-XACMLEntryPolicySubentry
}
```

The viewDSXACMLEntryPolicy attribute is automatically indexed for the viewDSXACMLPolicyMatch matching rule.

viewDSXACMLAttributePresentation

This operational attribute describes a mapping between a display name in the PAP interface and an XACML triplet. The XACML triplet comprises a category identifier, an attribute identifier and a data-type identifier. (A directory attribute type can also be associated with the triplet.)

```
viewDSXACMLAttributePresentation ATTRIBUTE ::= {
  WITH SYNTAX  XACMLAttributePresentation
  EQUALITY MATCHING RULE  viewDSXACMLAttributePresentationMatch
  USAGE  directoryOperation
  ID  id-viewds-aca-XACMLAttributePresentation
}
XACMLAttributePresentation ::= SEQUENCE {
  displayName     [0] UnboundedDirectoryString,
  category        [1] AnyURI,
  attribute       [2] XACMLAttributeIdentifier,
  dataType        [3] AnyURI,
  type            [4] AttributeType OPTIONAL,
  normalized      [5] BOOLEAN DEFAULT TRUE
  mustBePresent   [6] BOOLEAN DEFAULT FALSE,
```

```
  issuerAttribute [7] BOOLEAN DEFAULT FALSE
  obsolete        [8] BOOLEAN DEFAULT FALSE
  permittedValues [9] SEQUENCE OF UnboundedDirectoryString OPTIONAL
}
XACMLAttributeIdentifier ::= CHOICE {
  identifier  [0] AnyURI
  -- or an XPath expression in future
}
viewDSXACMLAttributePresentationMatch MATCHING-RULE ::= {
  SYNTAX  XACMLAttributeAssertion
  ID  id-viewds-mr-XACMLAttributePresentationMatch
}
XACMLAttributeAssertion ::= SEQUENCE {
  category  [0] AnyURI,
  attribute [1] XACMLAttributeIdentifier,
  dataType  [2] AnyURI
}
```

The normalized field specifies whether the PAP interface should apply stringprep normalization to the values of this attribute appearing in the conditions of rules. The issuerAttribute field indicates whether values of an attribute can be used to identify a policy's issuer. The permittedValues field contains a list of permitted values for an XACML attribute.

viewDSXACMLPolicyVersion

This operational attribute identifies the version and current state of an XACML policy. When a PAP user creates a new version of a policy, viewDSXACMLPolicyVersion is added to the access control administrative point.

```
viewDSXACMLPolicyVersion ATTRIBUTE ::= {
  WITH SYNTAX XACMLPolicyVersion
  EQUALITY MATCHING RULE viewDSXACMLPolicyVersionMatch
  USAGE directoryOperation
  ID id-viewds-aca-XACMLPolicyVersion
}
XACMLPolicyVersion ::= SEQUENCE {
  Identifier [0] XACMLVersion,
  issuer     [1] XACMLIssuer OPTIONAL,
  locked     [2] BOOLEAN DEFAULT FALSE,
  base       [3] XACMLVersion OPTIONAL
}
  viewDSXACMLPolicyVersionMatch MATCHING-RULE ::= {
  SYNTAX XACMLPolicyVersionAssertion,
  ID id-viewds-mr-XACMLPolicyVersionMatch
```

```
}
XACMLPolicyVersionAssertion ::= SEQUENCE {
  identifier [0] XACMLVersion,
  issuer     [1] XACMLIssuer OPTIONAL
}
```

The version field contains a single value to identify the version number of the policy. Version numbers starting with zero (0.1, 0.2, etc) are reserved for old policies that need to be archived and managed outside the PAP interface. The viewDSXACMLPolicy Version Match matching rule uses an integer match on the version field, and requires it to correspond to the assertion value exactly.

The base field identifies the version from which the current policy was created. If the field is undeclared, this indicates that the current policy is not based on an existing version.

The locked field indicates whether the version of policy should be made available for editing by the PAP user. The values of the viewDSXACMLPolicyVersion attribute are never modified or deleted when the locked field is true.

The viewDSXACMLPolicyVersionMatch will match if the issuer is not present in either value or is present in both.

viewDSXACMLNamedExpression

This operational attribute holds one or more named expressions that can be used by the PAP user when constructing conditions in an XACML rule.

```
viewDSXACMLNamedExpression ATTRIBUTE ::= {
  WITH SYNTAX  XACMLNamedExpression
  EQUALITY MATCHING RULE  viewDSXACMLNamedExpressionMatch
  SINGLE VALUE  TRUE
  USAGE  directoryOperation
  ID  id-viewds-aca-XACMLNamedExpression
}
XACMLNamedExpression ::= SEQUENCE {
  identifier       [0] UTF8String,
  version          [1] XACMLVersion,
  issuer           [1] XACMLIssuer OPTIONAL,
  descriptiveName  [2] UTF8String,
  description      [3] UTF8String OPTIONAL,
  definition       [4] [RXER:TYPE-REF {
    namespace-name "http://viewds.com/SchemaGlue",
    local-name "XACMLExpressionContainer" }] Markup
}
XACMLIssuer ::= [RXER:TYPE-REF {
  namespace-name “http://viewds.com/SchemaGlue”,
```

```
    local-name "XACMLPolicyIssuerContainer" }] Markup
}
  viewDSXACMLNamedExpressionMatch MATCHING-RULE ::= {
  SYNTAX  UTF8String
  id-viewds-mr-XACMLNamedExpressionMatch
}
  viewDSXACMLEmbeddedExpressionMatch MATCHING-RULE ::= {
  SYNTAX  UTF8String
  id-viewds-mr-XACMLEmbeddedExpressionMatch
}
```

viewDSXACMLActivePolicy

This operational attribute identifies the active version of a specific policy created by a specific issuer. (The combination of version number and issuer uniquely identifies each policy.) If the issuer is unspecified then the attribute identifies the active version of the trusted policy.

```
viewDSXACMLActivePolicy ATTRIBUTE ::= {
  WITH SYNTAX XACMLActivePolicy
  EQUALITY MATCHING RULE  viewDSXACMLActivePolicyMatch
  USAGE directoryOperation
  ID id-viewds-aca-XACMLActivePolicy
}
XACMLActivePolicy ::= SEQUENCE {
  version [0] XACMLVersion,
  issuer  [1] XACMLIssuer OPTIONAL
}
viewDSXACMLActivePolicyMatch MATCHING-RULE ::= {
  SYNTAX XACMLActivePolicyAssertion
  id-viewds-mr-XACMLActivePolicyMatch
}
XACMLActivePolicyAssertion ::= SEQUENCE {
  issuer [0] XACMLIssuer OPTIONAL
}
```

viewDSXACMLConfiguration

This operational attribute configures various aspects of the Policy Decision Point (PDP) and is stored in the directory's root entry. The attribute takes a single value with the syntax described by this ASN.1 type definition:

```
XACMLConfiguration ::= SEQUENCE {
  combining-algorithm  [0] AnyURI DEFAULT
  "urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides",
  default-version      [1] UTF8String (PATTERN "(\d+\.)*\d+") OPTIONAL,
  rfc822Name-attribute [2] AttributeType OPTIONAL,
  user-base-object     [3] DistinguishedName OPTIONAL,
  user-attributes      [4] SET OF AttributeType OPTIONAL,
  policy-base-object   [5] DistinguishedName OPTIONAL,
  allowed-origins      [6] SEQUENCE OF UTF8STRING OPTIONAL
}
viewDSXACMLConfiguration ATTRIBUTE ::=
  WITH SYNTAX XACMLConfiguration
  SINGLE VALUE TRUE
  USAGE dSAOperation
  ID id-viewds-aca-XACMLConfiguration
}
```

The attribute's fields are described below.

combining-algorithm

When the Policy Decision Point (PDP) evaluates an authorization decision request, it finds the applicable XACML policy sets and combines them according to the combining algorithm. This only applies to the policy sets declared in the viewDSXACMLPolicySet attribute. The values of viewDSXACMLPolicy and viewDSSecondaryXACMLPolicySet are only included if referenced by a policy defined in viewDSXACMLPolicySet. If the combining-algorithm field is absent, then the default deny overrides is applied. Plausible values are:

```
"urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides"
"urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides"
"urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit"
"urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny"
```

For further information see the XACML 3.0 specification.

default-version

XACML policies and policy sets can be versioned. By default, when there are multiple policies or policy sets with the same identifier, the Policy Decision Point (PDP) uses the one with the

highest version number. Alternatively, if the default-version field is defined, the Policy Decision Point (PDP) uses the policy or policy set with the highest version number that is less than or equal to the field's value.

rfc822Name-attribute

If subject attributes are not provided in an authorization decision request, the Policy Decision Point (PDP) will attempt to look them up in the Policy Information Point (the ViewDS directory). For this to occur the request must include the following XACML attribute:

```
urn:oasis:names:tc:xacml:1.0:subject:subject-id
```

If the data type of the subject-id is a:

- String – the Policy Decision Point looks for a directory entry whose viewDSUserName attribute equals the string value specified by subject-id.
- x500Name – the Policy Decision Point looks for a directory entry whose LDAP Distinguished Name equals the specified X500 name specified by subject-id.
- rfc822Name – the Policy Decision Point looks for a directory entry that has a value of the attribute type identified by the rfc822Name-attribute that is equal to the value specified by subject-id.

user-base-object

The root of the subtree in the directory that the Policy Decision Point (PDP) will search in order to find a user entry. (The directory acts as a Policy Information Point by storing information that can influence in an access decision.)

user-attributes

These are user attributes that the Policy Decision Point (PDP) will need to access when evaluating authorization requests.

policy-base-object

The root of the subtree in the directory that the Policy Decision Point (PDP) will search in order to find a policy or policy set.

allowed-origins

Defines a cross-origin resource sharing (CORS) policy that specifies from which origins the Policy Decision Point (PDP) will accept requests.

The field is a SEQUENCE OF UTF8String where each string is a regular expression conforming to XML Schema (see https://www.w3.org/TR/xmlschema-2/#regexs).

Example

Here is an example of a Stream DUA operation to add a value of the viewDSXACMLConfiguration attribute:

```
modify {}
  with changes {
   add attribute viewDSXACMLConfiguration {
     combining-algorithm "urn:oasis:names:tc:xacml:3.0:" +
     "policy-combining-algorithm:deny-unless-permit",
     default-version "3.1",
     rfc822Name-attribute { 0 9 2342 19200300 100 1 3
   }
  }
};
```