ViewDS Access Sentinel Installation and Reference Guide

Published: 2015
Version: 7.4(2)

ViewDS Access Sentinel Installation and Reference Guide

April 2015

Document Lifetime
ViewDS may occasionally update online documentation between software releases. Consequently, this PDF may not contain the most up-to-date information. Refer to the online documentation at www.viewds.com/resources/documentation.html for the most current information.

About this guide

This guide provides an introduction to Access Sentinel. It also describes the ViewDS implementation of XACML, how to install ViewDS 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 external to ViewDS.

Before using this guide, you should first read the system overview in the ViewDS Installation and Operations Guide.

Related documents

The other documents in the ViewDS document set are:

- ViewDS Installation and Operation Guide
- ViewDS Access Proxy Installation Guide
- ViewDS Application Integration Kit for .NET
- ViewDS Application Integration Kit for Java
- ViewDS Technical Reference Guide: Directory System Agent
- ViewDS Technical Reference Guide: User Interfaces
- ViewDS Management Agent In-application Help

How this guide is organized

This guide contains the following:

Section 1: About this guide

Provides an overview of this guide.

Section 2: About ViewDS Access Sentinel

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

Section 3: Installation and Configuration

Provides the instructions to install and configure Access Sentinel.

Section 4: Defining XACML policy

Provides information about Access Sentinel's implementation of XACML, along with a tutorial that steps through writing and applying XACML policy.

Appendix A: XACML attributes provided by a PEP

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

Appendix B: 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?
- Why use XACML access controls?
- Brief introduction to XACML
- Access Sentinel architecture

What is Access Sentinel?

The ViewDS core product includes an XACML framework. It allows you to apply an XACML Access Control scheme by defining XACML policy to control access to a ViewDS directory.

ViewDS Access Sentinel is an extension of the XACML framework and allows you to apply XACML policy to applications external to ViewDS.

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:

- You can impose consistent access-control policies across 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
- an access-control decision language

The policy language is used to specify 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 figure below illustrates a simplified XACML implementation.

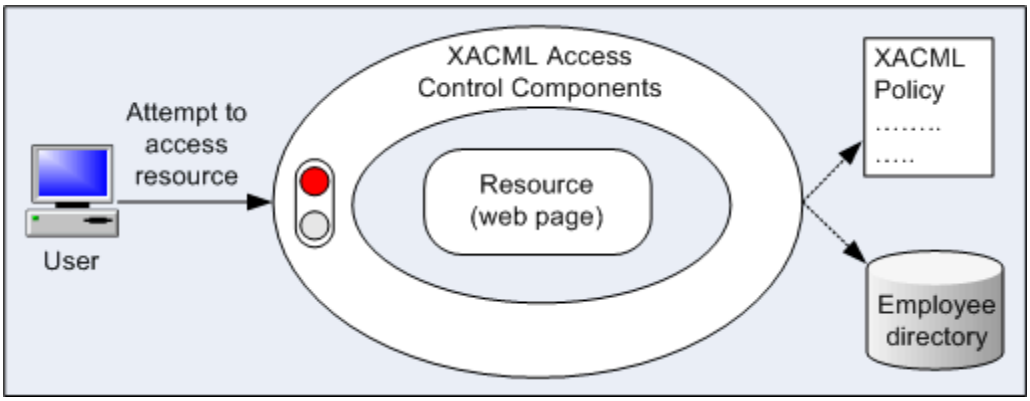


Figure 1: Attempt to access resource

In this example, a user attempts to view a web page protected by XACML access-control. 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).

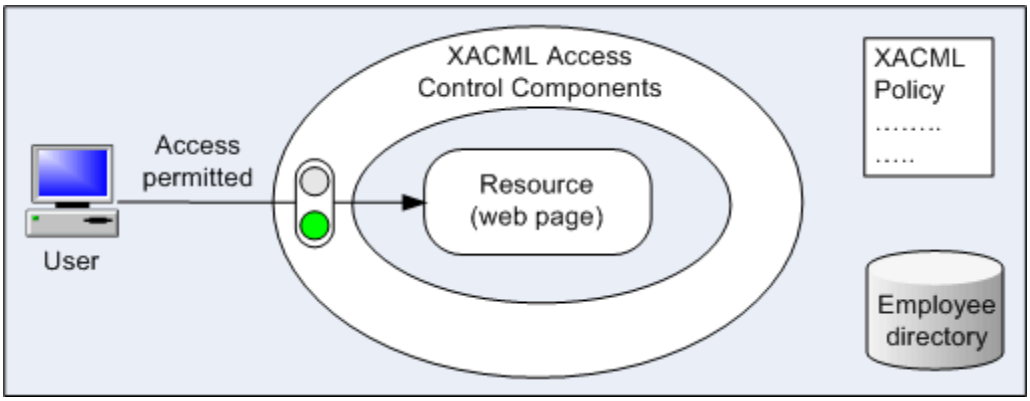


Figure 2: Permit access to resource

Components of XACML access control

An XACML access control implementation has four main logical components:

- Policy Enforcement Point (PEP) – protects a resource from unauthorized actions.
- Policy Decision Point (PDP) – makes decisions about whether access should be granted to a protected resource.

- Policy Administration Point (PAP) – allows policies to be created, managed 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.

These components are illustrated below.

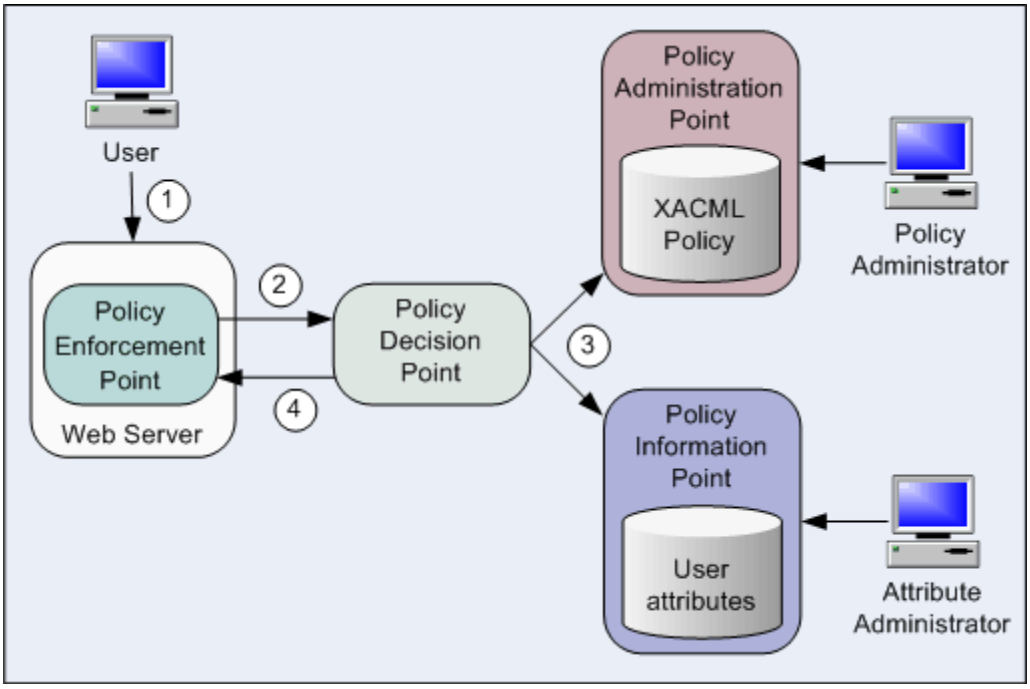


Figure 3: Example XACML access control components

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

The numbered steps shown above 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 a set of 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 attrib-

utes in the Policy Information Point (PIP). Among other things, the information in the PIP allows the PDP to identify the user attempting to access the resource.

4. The 'authorization decision response' is returned 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 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 is the approach adopted by ViewDS and is discussed in the section Access Sentinel architecture.

Access Sentinel architecture

Access Sentinel extends the XACML framework that is installed as part of the core ViewDS product's 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.

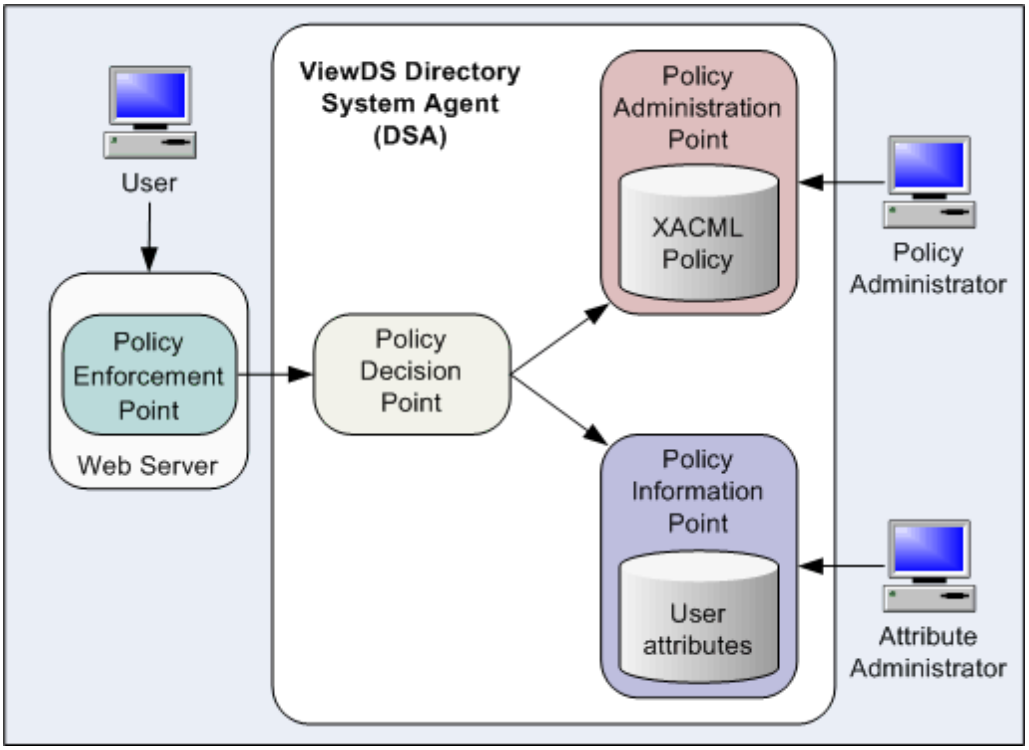


Figure 4: Access Sentinel architecture

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.

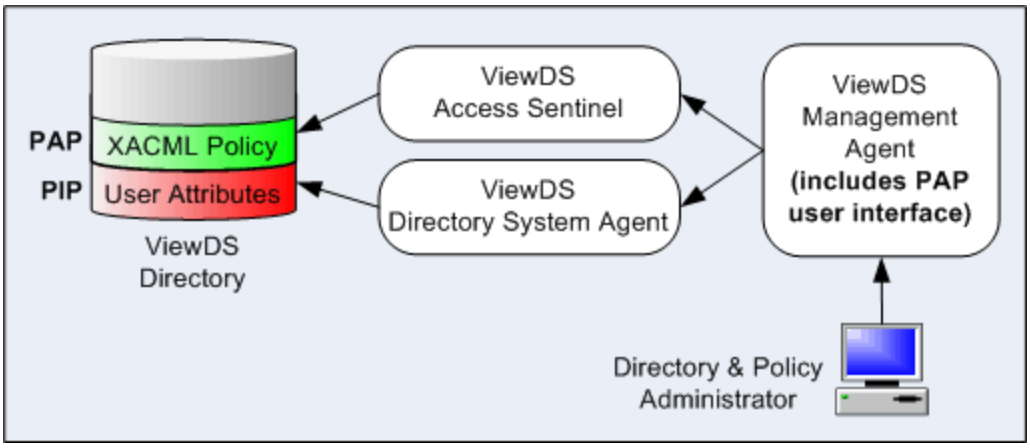


Figure 5: Unified PIP and PAP repository and user interface

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.

Delegation

In ViewDS, XACML policies are normally managed by trusted administrators who authenticate themselves with the ViewDS Management Agent using strong (PKI) credentials. However, trusted administrators can delegate all or part of their policy management authority to another user. This feature facilitates decentralized administration of policies and rules using the Authorization Policy Manager.

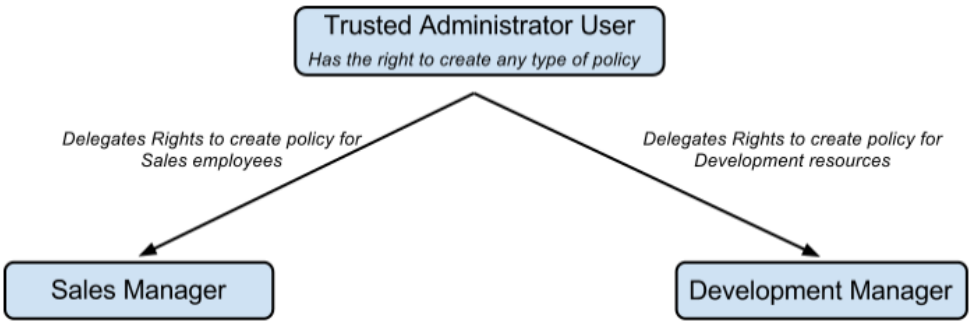


Figure 6: Delegation of authority

The figure above shows a trusted administrator who has delegated the ability to manage certain types of policies to Sales Managers and Development Managers.

Delegation rules are evaluated by the PDP at the same time as it evaluates an authorization request. As a result, delegates are able to create any type of policy, with the PDP determining at the point of authorization whether or not a particular delegate actually has the authority to implement a given policy.

So in the example above, a Sales Manager is able to create policies for both sales employees and development resources. However, since the delegation rules do not allow Sales Managers to implement development resource policies, the PDP will ignore the new development resource policy.

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
- AIK: Java / .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.

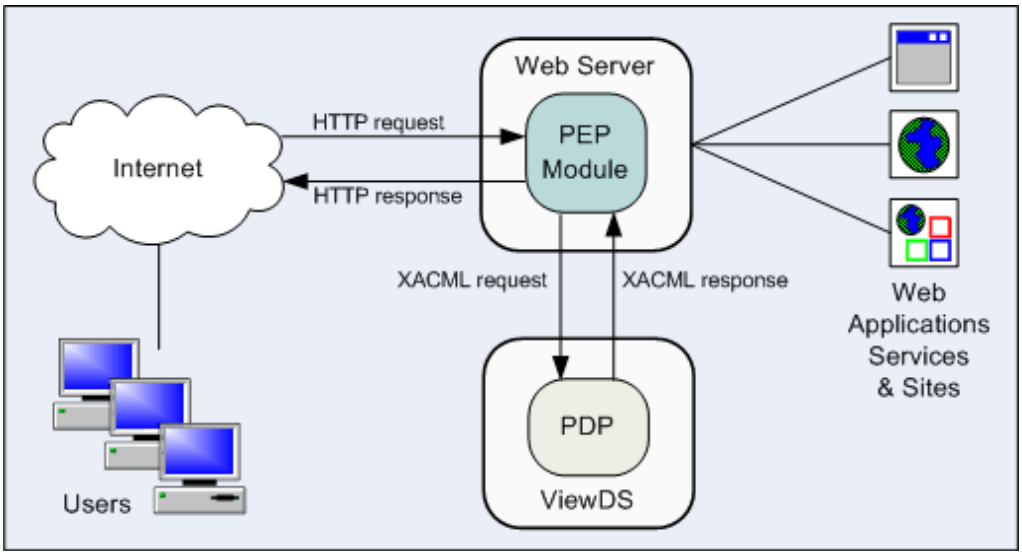


Figure 7: IIS or Apache PEP module

Application Integration Kits

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.

Installation and configuration

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

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

Installing ViewDS Access Sentinel

To install ViewDS Access Sentinel:

1. If ViewDS is not currently installed, perform the tasks in Installing ViewDS in the ViewDS Installation and Operations Guide.

   > An Access Sentinel licence is required.

   If ViewDS is already installed, add the Access Sentinel licence to the DSA's configuration – see the ViewDS Management Agent help topic, Import licence information.

2. Read and perform the task described in XACML configuration parameters.
3. Optionally, perform the task Installing the Authorization Policy Manager.
4. Perform one of the following tasks:
   - Deploying the IIS PEP
   - Deploying the Apache PEP

XACML configuration parameters

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

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

- User attributes
- Resource attributes
- Policy base object

Combining algorithm

Access Sentinel can evaluate policies from different sources: native ViewDS XACML policy (defined using the VMA 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, then the results are always 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 the Combining Algorithm specified here. Four combining algorithms are available:

- deny overrides – if any nested item (a rule, policy or policy set) evaluates to deny, then the container (a 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.
- permit overrides – if any nested item evaluates to permit, then the container evaluates to permit; otherwise, if any item evaluates to deny, then the container evaluates to deny; otherwise, the container evaluates to not-applicable.
- deny unless permit – if any nested item evaluates to permit, then the container evaluates to permit; otherwise, the container evaluates to 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.

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, then 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 VMA or the Authorization Policy Manager.

RFC822 name 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 (PIP - the ViewDS server). 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:

- 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.

User base object

This is the root of the directory subtree in the Policy Information Point (PIP) that the Policy Decision Point (PDP) will search in order to find a user's entry.

User attributes

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

Resource attributes

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

Policy base object

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

Setting the XACML configuration parameters

1. From 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. Complete the boxes in the **XACML Config** tab as required.
5. At the bottom of the tab, click the **Set XACML Configuration** button.

Installing the Authorization Policy Manager

The Authorization Policy Manager is a stand-alone PAP that can be installed on any platform.

To install the application:

1. Install Java SE Runtime (32-bit).
2. From the Access Sentinel distribution media, unzip the file PAPui.zip.
3. In the extracted folder, double-click PAPui.jar. The Authorization Policy Manager starts.

The Authorization Policy Manager requires you to authenticate yourself before you can connect to Access Sentinel. You can do this by providing a username and password or by using a certificate.

Certificate based authentication

To install your user certificate:

1. Add your user certificate to the ViewDS trusted directory. For example, in Windows the trusted directory is as follows:

   ```
   %VFHOME%\setup\trusted
   ```

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

   See Installing credentials in the ViewDS Installation and Operations Guide for further details.

2. Import your PKI credentials into the Authorization Policy Manager by performing the following steps:
   a. Click **Tools** on the menu bar and select **Keystore** from the dropdown list.
   b. Create a default Java keystore for the Authorization Policy Manager. We recommend creating a keystore with a password.
   c. In the **Key Store** dialog click **Import** and follow the onscreen prompts to import your PKCS#12 (.p12) file into the keystore you just created.

Create a connection

To connect to Access Sentinel:

1. From the menu bar, click **File** followed by **New Session**. The New Session window is displayed.
2. In the **Name** box, enter a name for your ViewDS DSA. this will appear in the left pane of the interface.
3. In the **Host** box, enter the address of your ViewDS DSA. For example, if the Authorization Policy Manager is on the same host as the DSA, enter localhost.
4. In the **Port** box, enter the port number to connect to on the DSA (by default 3000).
5. Choose the authentication method you want to use: **Simple** (username and password) or **Certificate** (see Certificate based authentication).
   a. To use **Simple** authentication provide your **Username** and **Password**, then click **Save**.

If you use **Simple** authentication, then you will be required to provide your password every time you start a new Authorization Policy Manager session. To do this click **Connect** in the main window and type your **Password** into the box provided.

b. To use **Certificate** authentication click the **Certificate** tab and provide your key store password if required. Choose the appropriate **Key Alias** from the dropdown list and provide the corresponding **Password**, then click **Save**.

Getting started

These steps will introduce you to the interface.

1. Right-click Deltawing and then click **Add XACML Access Control Domain**. The **Policies** and **Attributes** tabs are displayed in the right pane.
2. Click the **New** button. The **New XACML Policy** window is displayed.
3. Click **OK** to accept the defaults. A new policy is listed in the **Policies** tab and the **Rules** and **Named Expressions** tabs are added to the right pane.

   The interface is now identical to the **XACML AC** tab in the ViewDS Management Agent – both allow you to perform exactly the same tasks.
4. To remove the XACML Access Control Domain, right-click Deltawing and then click **Remove XACML Access Control Domain**.

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

Click on the links above to see a description of each task.

Enabling .NET extensibility for IIS

To enable .NET extensibility for IIS on Windows 7 or Windows Server 2008:

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** checkbox.

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 bin folder for the site. You should 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 ISS Manager on Windows 7:
    a. From **Information Services (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, click **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. For example, under Windows 7:
    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, and then click the checkbox to Allow 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 in the pepConfig.txt file as required – see IIS PEP configuration-file parameters.

IIS PEP configuration-file parameters

The IIS PEP has a configuration file 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
XACMLTrace	This parameter is required to be on as part of enabling tracing (see Tracing decision making). With tracing enabled, the PEP sends authorization decision requests that enable tracing of the policies evaluated by the PDP in order to generate an authorization decision response. The PDP writes tracing information to its query log; and the PEP writes tracing information in the authorization decision response to its log folder (identified by the LogPath parameter). Default: off
`LogSwitch`	When this parameter is on, the PEP logs all authorization decision requests and responses exchanged with the PDP to the log folder (identified by the LogPath parameter).
`LogPath`	The location of the PEP's log files. For example: c:\peplog
`NotApplicable`	The PEP's action if it receives 'not applicable' in an authorization decision response from the PDP. If the parameter is set to allow, the user will be granted access to the resources they are attempting to access; if it is set to intercept, they will be denied access. Advised: intercept
`Indeterminate`	The PEP's action if it receives 'indeterminate' in an authorization decision response from the PDP. If the parameter is set to allow, the user will be granted access to the

	resources they are attempting to access; if it is set to intercept, they will be denied access. Advised: intercept
`NoResponse`	The PEP's action if receives no response to an authorization decision request. If the parameter is set to allow, the user will be granted access to the resources they are attempting to access; if it is set to intercept, they will be denied access. Advised: intercept

The following is an example configuration file:

```
XACMLHost localhostXACMLPort 3009XACMLTrace offLogPath c:\peplogNotAp-
plicable interceptIndeterminate interceptNoResponse intercept
```

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 the **Save** button.

Test the deployment

1. Test your deployment by attempting to access the protected website. You should be denied access, which is the default behaviour in the absence of XACML policy.
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

Click on the links above to see a description of each task.

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 (under 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 in the next subsection below.

Apache PEP configuration parameters

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

`XACMLHost`	The host name of the ViewDS server (which includes the PDP).
`XACMLPort`	The soapAddress on the ViewDS server (by default, 3009) where the PDP listens for authorization decision requests (see Modifying the SOAP address).
`XACMLTrace`	Optional and determines whether the PEP's authorization decision requests will switch on decision tracing in the PDP. (The 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 permit is specific to Access Sentinel.
`XACML Authoritative`	Optional and determines whether this module is the authoritative author-

isation 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 the **Save** button.

Test the deployment

1. Test your deployment by attempting to access the protected website. You should be denied access, which is the default behaviour in the absence of XACML policy.
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 click the **Set** button.

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.)

Enable tracing

To enable tracing:

1. Set the PEP's configuration-file parameter XACMLTrace to on (see IIS PEP configuration-file 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 the **Set** button.
3. Define an XACML attribute with the following settings in the PAP's **Attributes** tab:
    a. User Friendly Name equals tracing (for example)
    b. XACML Attribute Category equals urn:oasis:names:tc:xacml:3.0: attribute-category:action
    c. XACML Attribute Identifier equals urn:oasis:names:tc:xacml:1.0: action:action-id
    d. XACML Data Type equals anyURI
4. Create a new permit rule within the policy. The rule's condition should be that the above XACML attribute is equal to http://viewds.com/xacml/environment/trace.

Defining XACML policy

XACML policy

This section provides the background information you will need to write XACML policy. We'll start by looking at how XACML can be used to protect web pages (see Figure 8: Access Sentinel components below).

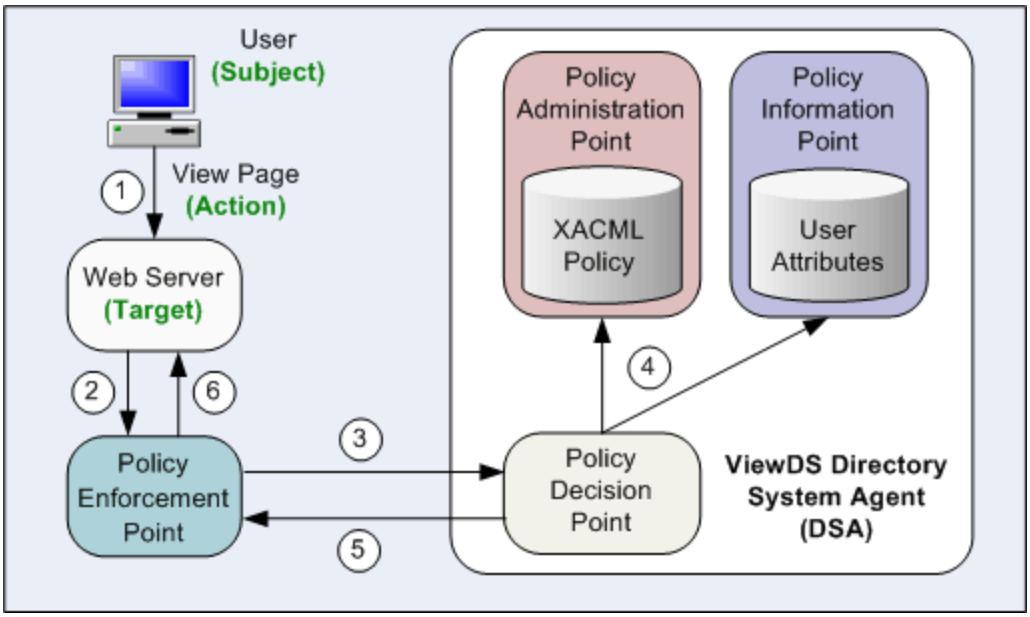


Figure 8: Access Sentinel components

The steps shown in Figure 8: Access Sentinel components are as follows:

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:

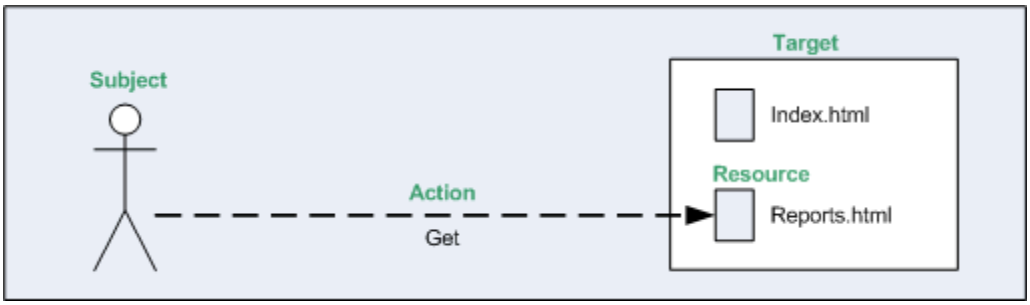


Figure 9: XACML terminology

XACML policy components

The Access Sentinel implementation of an XACML policy comprises:

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

Each is discussed below.

XACML Access Control Domain

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

In the ViewDS implementation of XACML, 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, Access Sentinel 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.

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 – which identify the subject attempting to access a particular resource.
- Resource – which identify the resource the subject is attempting to access.
- Action – which identify the action the subject is attempting to perform on the resource (for example, read, modify).
- Environment – which identify environmental factors such as the 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 ViewDS PEPs, 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.

Access Sentinel allows you to declare two different types of attributes: attribute designators and attribute 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 XACML 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 such XML documents using XPath queries.

XPath is a language, based on a tree representation of XML documents, which provides the ability to navigate around 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 be configured with 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.

Access Sentinel currently supports:

- 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

However, attribute selectors are not supported by the following as they do not make use of XML documents within authorization decision requests:

- the ViewDS XACML framework
- the HTTP PEPS (IIS and Apache)

Rules

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 (entry), or to a target resource and all its subordinates (subtree).

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 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'.

Attribute-based versus role-based access control policies

Access Sentinel supports both attribute-based access control (ABAC) and role-based access control (RBAC) policies.

In ABAC attributes associated with for example 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) in order to establish if access should be permitted or denied.

Like ABAC, RBAC uses attributes to construct conditions however a separate condition that identifies a subject's role is also included.

For example, an ABAC policy may look like this:

```
Permit if the following condition is met:
 action = read AND
 resource = document-xyz AND
 subject's title = 'Sales Executive' AND
 subject's age > 18
```

An equivalent RBAC policy that separates attribute conditions and role conditions may look like this:

```
Permit if the following condition is met:
 action = read AND
 resource = document-xyz AND
 subject's age > 18
 AND the following role condition is met:
 subject's role = Sales Executive
```

Additionally, RBAC makes use of a role hierarchy for permission inheritance. This means that access rights for a given user are evaluated based on their allocated role and any permissions they inherit from subordinate roles within the role hierarchy.

Only permit rules are inherited.

For example, consider the situation in which the Sales Executive role has a subordinate role Employee (Figure 10: Role hierarchy inheritance). Using RBAC, a Sales Executive will be evaluated using policies that apply to their role directly as well as any permit rules for the junior role of Employee.

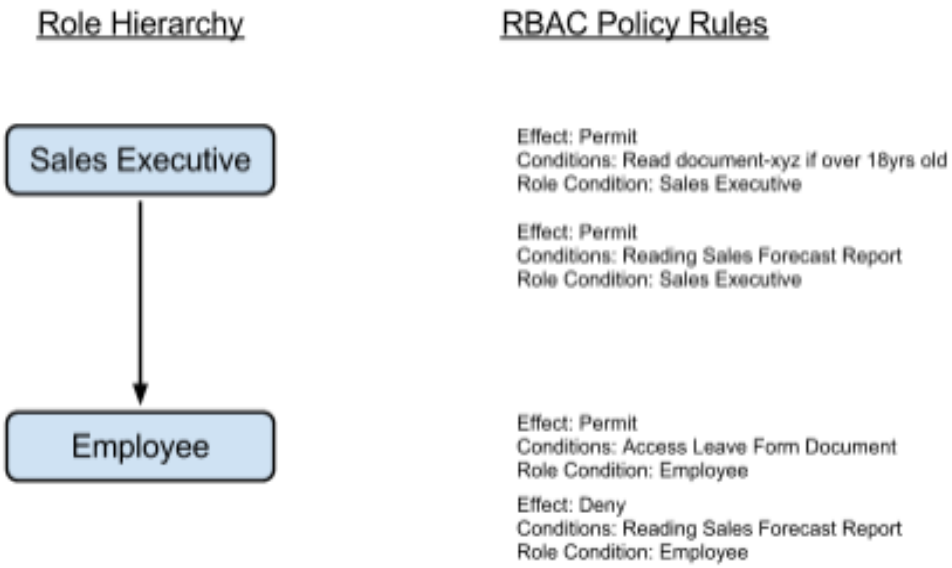


Figure 10: Role hierarchy inheritance

This means that, in the example illustrated above, a Sales Executive who is over 18 years old would be able to read document-xyz, read the Sales Forecast Report and (due to role hierarchy inheritance) have access to the Leave Form Document.

Role management

To facilitate RBAC, the ViewDS XACML framework allows you to define and manage discrete roles and role hierarchies for a given access control domain using the Authorization Policy Manager (and the ViewDS Management Agent).

Once defined, these roles can be used as static or dynamic role values and included in XACML access control policies (including ABAC policies).

Static roles are obtained from directory entries or XACML requests. Whereas, dynamic roles are determined by performing some sort of run-time evaluation.

Role enablement

Role enablement extends the ViewDS XACML framework to support dynamic roles. The Authorization Policy Manager (and the ViewDS Management Agent) can be used to define role enablement rules in the form of XACML policies. These rules harness the power of XACML to determine user roles dynamically. For example:

User role = 'Acme-Employee' if email address ends with '@acme.com'

User role = 'Acme-External-Contractor' if email address ends with @third-party-contractors.acme.com'

Obligations and advice

Obligations and advice are features of XACML 3.0 that have been implemented in Access Sentinel 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.

> 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.

HTTP PEP tutorial

This tutorial takes you through the steps to define and apply an XACML policy to web pages hosted by either an Apache or IIS web server.

For full details of the XACML categories and types of the attributes provided by each PEP, see Lock the policy.

> It is the application – in this case the web server - which defines the attributes it uses, rather than that being determined by, for example, the policy writer.

The tutorial includes the following:

- Overview
- Set the policy base object
- Create tutorial files and configure the web server
- Declare XACML attributes
- Create a policy
- Define the first rule
- Define the second rule
- Define the third rule
- Activate the policy
- Test the policy
- Lock the policy

> Before starting the tutorial, read XACML policy.

Overview

This tutorial illustrates how to create a policy 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

Figure 11: Policy requirement illustrates the third requirement.

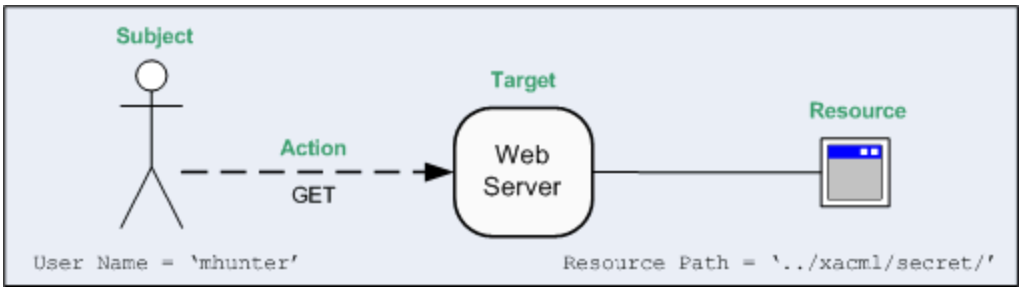


Figure 11: Policy requirement

When a user (subject) attempts to access a webpage (resource), the 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.

Attributes

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

User friendly name	**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

An XACML attribute's category corresponds to its purpose, as illustrated in Figure 11: Policy requirement.

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 and condition for each rule in this tutorial are shown below.

Rule 1:

Effect: Permit (if the condition is true)

Condition:URL Path contains 'index.html'

Rule 2:

Effect: Permit (if the condition is true)

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

Rule 3:

Effect: Permit (if the condition is true)

Condition: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 the Browse button 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 create the tutorial 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):
    1. /xacml/index.html
    2. /xacml/restricted/index.html
    3. /xacml/restricted/restricted.html
    4. /xacml/secret/index.html
    5. /xacml/secret/secret.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 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:

1. In the ViewDS Management Agent, click **Server View**.
2. In the left pane, click your ViewDS server. The Status tab displays the status of your ViewDS server. Ensure that the ViewDS Management Agent is connected to your ViewDS server, and that your ViewDS server is running.
3. In the bottom left pane, click **Global DIT View**.
4. Press **F5** to refresh the screen.
5. In the left pane, expand the **Deltawing** entry at the top of the Directory Information Tree (DIT).
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.

Declare XACML attributes

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

1. In the right pane, click the **XACML AC** tab.
2. Within the XACML AC tab, click the **Attributes** tab.
3. At the bottom of the right pane, click the **New button**. The XACML Attribute window is displayed.
4. In the **Label** box, enter *URL Path*.
5. In the **Category** box, click **urn:oasis:names:tc:xacml:3.0: attribute-category:action**. The Identifier box defaults to urn:oasis:names:tc:xacml:1.0: action:action-id, and the Data Type box defaults to string.
6. In the **Identifier** box, delete the default value and enter the following:

   http://viewds.com/http/resource/path
7. Click **Save**. The XACML attribute is added to the Attributes tab.
8. Repeat the above steps to declare the following XACML attribute:

User Friendly Name	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

Create a policy

To create the policy:

1. In the XACML AC tab, click **Policy Versions**.
2. In the right pane, click **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 version number and its status is displayed next to the Version Management button.

> 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 define the first rule:

1. 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.
2. In the Label box, enter Access to index.html.
3. Optionally, in the Description box, enter a longer description of the rule.
4. Click the Edit button. The XACML Expression window is displayed. This is described in more detail below.

XACML Expression window

Figure 12: XACML Expression window shows the XACML Expression window, which allows you to define rule conditions.

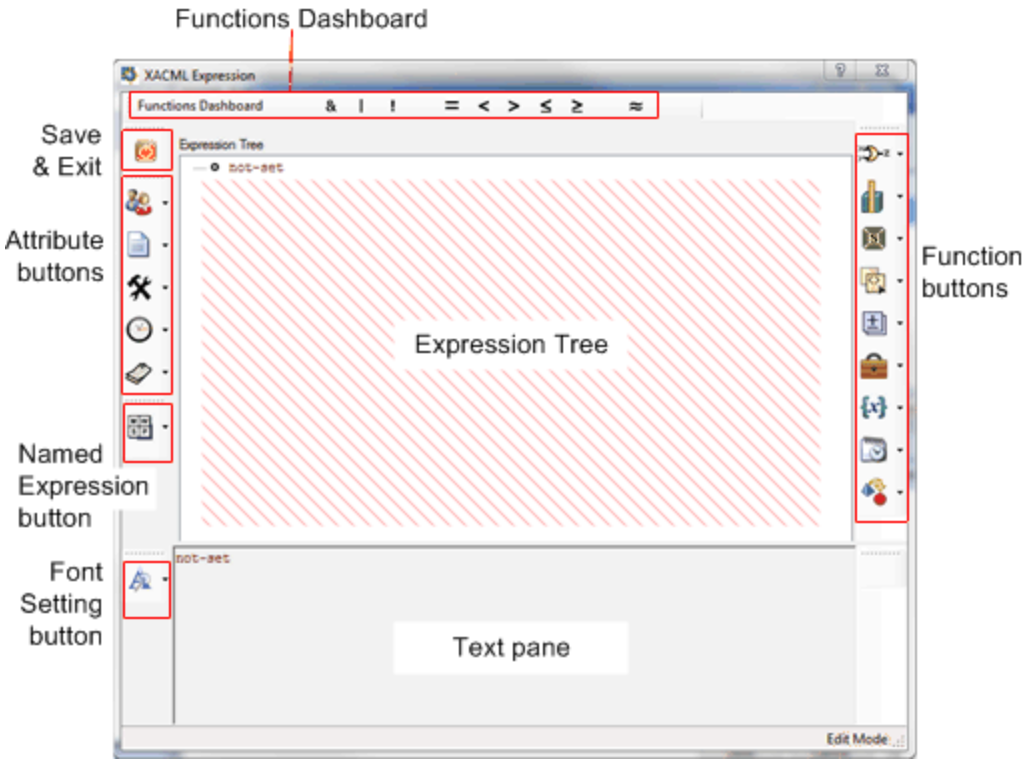


Figure 12: XACML Expression window

The window has the following areas:

- Expression Tree – the window's main work area and allows you to build the expressions in a rule's condition in a tree format.
- Text Pane – shows the contents of the Expression Tree in a plain text format.

The window also has the following buttons:

- Functions Dashboard – allows 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 – allows you save the Expression Tree and exit the XACML Expression window.
- Attribute 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 and environment attributes.
- Font Setting button – allows you to change the font for the attributes, values, functions and named expression displayed in the text pane.
- Named Expression button –allows you to add a named expression to the Expression Tree.
- Function buttons– allow you to add a function to the Expression Tree. There are eight function categories: Boolean, Relational, String, Arithmetic, Bag, Set, Date and Time, and Conversion.

The interface provides descriptions of individual functions through pop-up 'tool tips'.

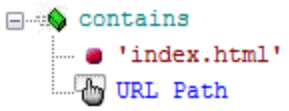
A condition comprises expressions

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'

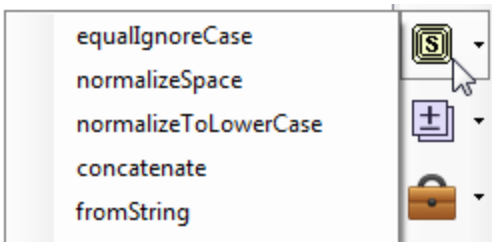
Every expression has a function and XACML attributes. The function is contains and the XACML attribute is Resource Path, and is represented in the expression tree as follows:



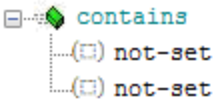
Defining the condition

To define the condition:

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

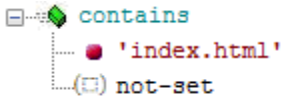


2. Drag and drop the contains function onto the not-set node in the expression tree. The contains function is added to the tree with two not-set nodes below it.

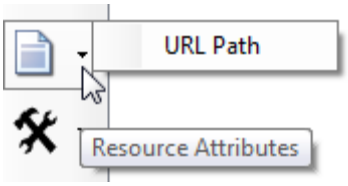


3. Double-click the first not-set node. The String Editor window is displayed.

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

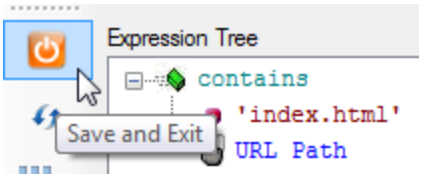


5. Click the Resource Attributes button.



6. Drag and drop URL Path onto the remaining not-set node.

7. Click the Save and Exit button. The XACML Expression window closes and the XACML Rule window is displayed.



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

Define the second rule

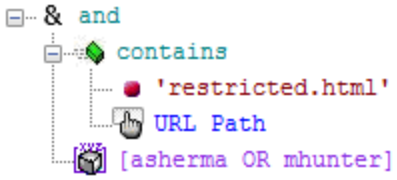
The second rule's condition is as follows:

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.

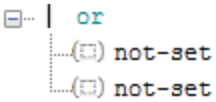
The first expression and the named expression will be tied together by a Boolean 'and' function to form the second rule.



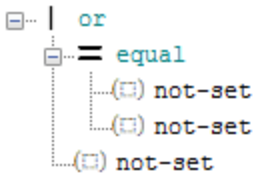
Defining the named expression

To define the named expression:

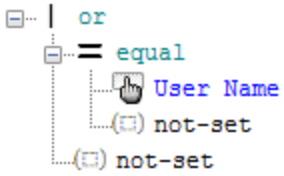
1. In the right pane, click the Policy Versions tab.
2. In the first filter box, click Named Expressions. The named expressions are listed in the summary area of the tab.
3. Click the New icon. The XACML Named Expression window is displayed.
4. In the Name box, enter asherma OR mhunter.
5. Click the Edit button. The XACML Expression window is displayed.
6. 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.



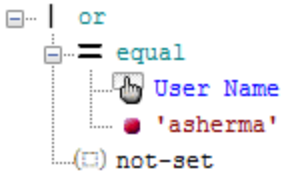
7. Drag and drop the = equal function from the Functions Dashboard onto the first not-set node. The function is added to the expression tree with two empty nodes below it.



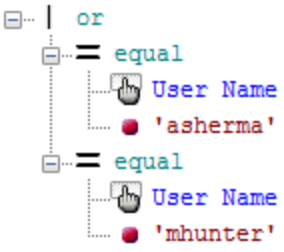
8. Click the Subject Attributes button, then drag and drop User Name onto the first not-set node below the = equal function.



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



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

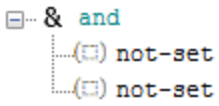


12. Click the Save and Exit button.
13. Click Save.

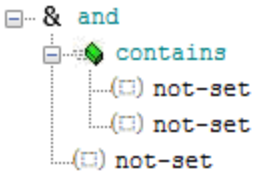
Defining the second rule

To define the second rule:

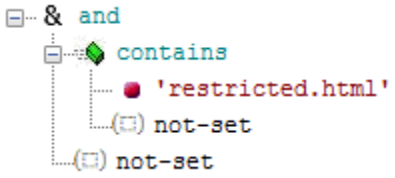
1. With ABAC Rules and Access selected in the filter boxes, click the New icon. The XACML Rule window is displayed.
2. In the Label box, enter Access to restricted.html.
3. Click the Edit button. The XACML Expression window is displayed.
4. Drag and drop the & function from the Functions Dashboard onto 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.



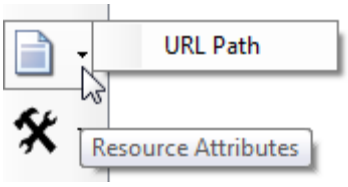
5. Click the String Functions button. A list of functions is displayed.
6. Drag and drop the contains function onto the first not-set node in the expression tree. The function is added to the tree with two not-set nodes below it.



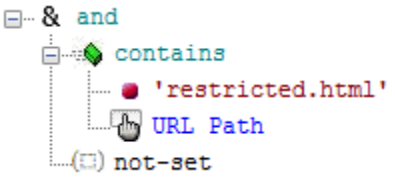
7. Double-click the first not-set node. The String Editor window is displayed.
8. In the Value box, enter restricted.html and click OK.



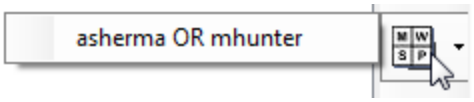
9. Click the Resource Attributes button.



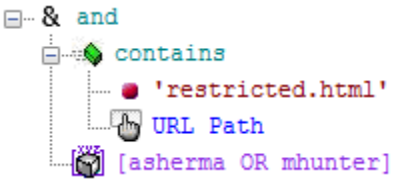
10. Drag and drop URL Path onto the not-set node below restricted.html.



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.

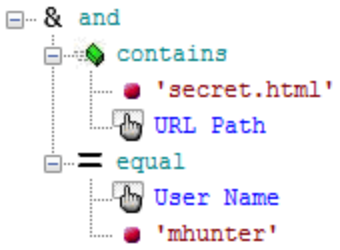
Define the third rule

The third rule's condition is as follows:

URL Path contains 'secret.html' AND

User Name = 'mhunter'

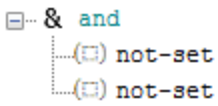
It is defined in the expression tree as follows:



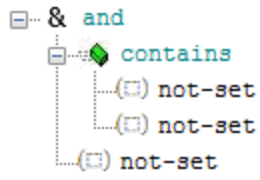
To define the rule:

1. With ABAC Rules and Access selected in the filter boxes, click the New icon. The XACML Rule window is displayed.

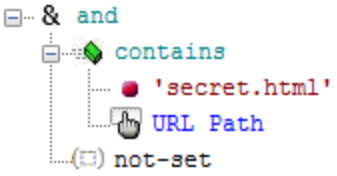
2. In the Label box, enter Access to secret.html.
3. Click the Edit button. The XACML Expression window is displayed.
4. Drag and drop the & function from the Functions Dashboard onto the not-set node at the top of the Expression Tree. The function is added to the expression tree with two empty nodes below it.



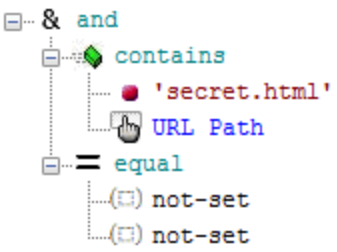
5. Click the String Functions button.
6. Drag and drop the contains function onto the first not-set node in the expression tree. The function is added to the tree with two not-set nodes below it.



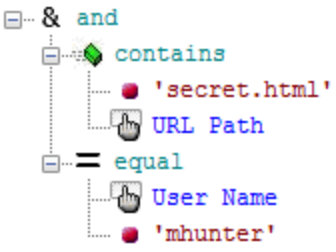
7. Double-click the first not-set node. The String Editor window is displayed.
8. In the Value box, enter secret.html and click OK.
9. Click the Resource Attributes button.
10. Drag and drop URL Path onto the not-set node below secret.html.



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 the Save button.

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. Click the **Policy Versions** tab.
2. Click the **Version Management** button followed by **Activate**. A warning is displayed.
3. 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’

To see a trace of the authorization decision request and response, see Tracing decision making.

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. Click the **Policy Versions** tab.
2. Click the **Version Management** button followed by **Lock**. A warning is displayed.
3. Click **Yes**. The policy’s Status is now *active,locked*.

XACML attributes provided by a PEP

This appendix describes the values supplied by each Policy Enforcement Point (PEP).

The Apache and IIS PEPs generate authorization decision requests that include values in the following XACML attribute categories:

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

The attributes are included in an authorization decision request if the corresponding information is available in the HTTP server request context.

To use a value supplied by a PEP in a policy, the XACML Access Control Domain must include an XACML attribute definition with the appropriate combination of XACML attribute category, identifier and data type. Each combination corresponds to a value shown in the following tables.

Access-subject category

These values 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 category	XACML attribute identifier	XACML data type
HTTP authenticated user identifier	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	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

Resource category

These values 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 category	XACML attribute identifier	XACML data type
URL host name	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	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	string
URL port number		http://viewds.com/http/resource/port	integer
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

Action category

This value 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 category	XACML attribute identifier	XACML data type
HTTP request method	urn:oasis:names:tc:xacml:3.0:attribute-category:action	urn:oasis:names:tc:xacml:1.0:action:action-id	string

Environment category

These values are in the XACML category: http://viewds.com/http/environment/redirect-uri

Value	XACML attribute category	XACML attribute identifier	XACM-L data type
Redir-ection page's query string	http://viewd-s.com/http/environment/redirect-uri	http://viewd-s.com/http/environment/redirect-query	string
Redir-ection page's URL		http://viewd-s.com/http/environment/redirect-uri	anyURI

Requesting-machine category

These values are in the XACML category: urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine

Value	XACML attribute category	XACML attribute identifier	XACML data

			type
HTTP server host name	urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	http://viewds.com/http/subject/hostname	string
HTTP server IP address		http://viewds.com/http/subject/address	string

Operational attributes

This appendix describes the 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 Technical Reference Guide: Directory System Agent.

viewDSXACMLSubtreePolicy

This operational attribute stores an XACML policy that applies to an Access Control Domain whose administrative point is at the top of a subtree. The policy applies to the entire subtree.

```
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  id-viewds-mr-XACMLNamedExpressionMatch
 }
 viewDSXACMLEmbeddedExpressionMatch MATCHING-RULE ::= {
 SYNTAX  UTF8String
 ID  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  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-over-
rides",
 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
 }
 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-over-
rides"
 "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.

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 }
 }
 } ;
```