AuthZForce Documentation

Release 4.4

Cyril Dangerville

Sep 27, 2017

Contents

CHAPTER 1

Introduction

AuthZForce is the reference implementation of the Authorization PDP Generic Enabler (formerly called Access Control GE). Indeed, as mandated by the GE specification, this implementation provides an API to get authorization decisions based on authorization policies, and authorization requests from PEPs. The API follows the REST architecture style, and complies with XACML v3.0. XACML (eXtensible Access Control Markup Language) is a OASIS standard for authorization policy format and evaluation logic, as well as for the authorization decision request/response format. The PDP (Policy Decision Point) and the PEP (Policy Enforcement Point) terms are defined in the XACML standard. This GEri plays the role of a PDP.

To fulfill the XACML architecture, you may need a PEP (Policy Enforcement Point) to protect your application, which is not provided here. For REST APIs, we can use the PEP Proxy (Wilma) available in the FIWARE catalogue.

Contents:

AuthZForce - Installation and Administration Guide

This guide provides the procedure to install the AuthZForce server, including system requirements and troubleshooting instructions.

System Requirements

- CPU frequency: 2.6 GHz min
- CPU architecture: i686/x86_64
- RAM: 4GB min
- Disk space: 10 GB min
- Operating System: Ubuntu 14.04 LTS
- **Java environment (automatically installed with the Ubuntu package):**
  - JDK 7;

- Tomcat 7.x.

Installation

Minimal

1. Download the latest binary (Ubuntu package with `.deb` extension) release of AuthZForce from the FIWARE catalogue, in the Downloads section. You get a file called `authzforce_4.2.0-fiware_all.deb`.
2. Copy this file to the host where you want to install the software.
3. **On the host, from the directory where you copied this file, run the following commands:**

```
$ sudo aptitude install gdebi curl
$ sudo gdebi authzforce_4.2.0-fiware_all.deb
```

4. At the end, you will see a message giving optional instructions to go through. Please follow them as necessary.

Advanced

The previous section gave you minimal installation steps to get started testing the features of the GE API. This may be enough for testing purposes, but barely for production. If you are targeting a production environment, you have to carry out extra installation and configuration steps to address non-functional aspects: security (including availability), performance, etc. The *Appendix* also gives some recommendations on what you should do.

Administration

Tomcat

For configuring and managing Tomcat, please refer to the official user guide.

Authzforce webapp

The Authzforce webapp configuration directory is located here: `/opt/authzforce/conf`.

In particular, the file `logback.xml` configures the logging for the webapp (independently from Tomcat). By default, Authzforce-specific logs go to `/var/log/tomcat7/authzforce/error.log`.

Restart Tomcat to apply any configuration change: `$ sudo service tomcat7 restart`

Policy Domain Administration

The Concept of Policy Domain

The application is multi-tenant, i.e. it allows users or organizations to work on authorization policies in complete isolation from each other. In this document, we use the term *domain* instead of *tenant*. In this context, a policy domain consists of:

- Various metadata about the domain: ID, name, description;
- The root XACML <PolicySet>;
- Optional <PolicySet>s that may be referenced in <Policy(Set)Reference>s by the aforementioned root <PolicySet>;

- Attribute Finders configuration: attribute finders resolve attributes from other sources than the PEP's or any other client's XACML <Request>.

The reasons for creating different domains:

- Users or organizations do not want others to access their data, or even be impacted by others working on the same application.
- The same user or organization may want to work on different domains for different use cases; e.g. work with one policyset for production environment, another for testing, another for a specific use case project, etc.

Domain Creation

You create a domain by doing a HTTP POST request with XML payload to URL: `http://${SERVER_NAME}:${PORT}/authzforce/domains`. Replace `${SERVER_NAME}` and `${PORT}` with your server hostname and port for HTTP. You can do it with `curl` tool:

```
$ curl --verbose --trace-ascii - --request POST \
--header "Content-Type: application/xml;charset=UTF-8" \
--data '<?xml version="1.0" encoding="UTF-8"?><taz:properties xmlns:taz="http://
↪thalesgroup.com/authz/model/3.0/resource"> <name>MyDomain</name><description>This␣
↪is my domain.</description></taz:properties>' \
--header "Accept: application/xml" http://${SERVER_NAME}:${PORT}/authzforce/domains
...
> POST /authzforce/domains HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.
↪3.4 libidn/1.23 librtmp/2.3
> Host: az.testbed.fi-ware.eu
> Content-Type: application/xml;charset=UTF-8
> Accept: application/xml
> Content-Length: 227
>
...
< HTTP/1.1 200 OK
< Server: Authorization System
< Date: Mon, 04 Aug 2014 13:00:12 GMT
< Content-Type: application/xml
< Transfer-Encoding: chunked
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><link xmlns="http://www.w3.org/
↪2005/Atom" rel="item" href="0ae7f48f-1f13-11e3-a300-eb6797612f3f"/>
```

WARNING: Mind the leading and trailing single quotes for the `--data` argument. Do not use double quotes instead of these single quotes, otherwise curl will remove the double quotes in the XML payload itself, and send invalid XML which will be rejected by the server. The `--trace-ascii -` argument (the last dash here means *stdout*) is indeed a way to check the actual request body sent by `curl`. So use it only if you need to dump the outgoing (and incoming) data, in particular the request body, on *stdout*.

The `href` value in the response above gives you the domain ID (in the form of a UUID), that you will now use for assigning user roles on the domain.

Domain Removal

You remove a domain by doing a HTTP DELETE request with XML payload to URL: `http://${SERVER_NAME}:${PORT}/authzforce/domains/{domain_ID}`.

For example with `curl` tool:

```
$ curl --verbose --request DELETE --header "Content-Type: application/xml;charset=UTF-
↪8" \
    --header "Accept: application/xml" http://${SERVER_NAME}:${PORT}/authzforce/
↪domains/0ae7f48f-1f13-11e3-a300-eb6797612f3f
```

Policy administration is part of the Authorization Server API, addressed more extensively in the *Programmer Guide*.

Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that the installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

End to End testing

To check the proper deployment and operation of the Authorization Server, perform the following steps:

1. Get the list of policy administration domains by doing the following HTTP request, replacing `${host}` with the server hostname, and `${port}` with the HTTP port of the server, for example with `curl` tool:

   ```
   $ curl --verbose --show-error --write-out '\n' --request GET http://${host}:$
   ↪{port}/authzforce/domains
   ```

2. Check the response which should have the following headers and body (there may be more headers which do not require checking here):

   ```
   Status Code: 200 OK
   Content-Type: application/xml
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <ns2:resources xmlns:ns2="http://thalesgroup.com/authzforce/model" xmlns:ns3=
   ↪"http://www.w3.org/2005/Atom">
      ... list of links to policy domains omitted here...
   </ns2:resources>
   ```

You can check the exact body format in the representation element of response code 200 for method `getDomains`, and all other API resources and operations in general, in the WADL (Web Application Description Language) document available at the following URL:

```
http://${host}:${port}/authzforce/?_wadl
```

List of Running Processes

- One or more `java` processes for Tomcat.

Network interfaces Up & Open

- TCP 22;
- TCP 8080.

The port 8080 can be replaced by any other available port by any other port Tomcat is listening to for HTTP connections to the webapp.

Databases

None.

Diagnosis Procedures

1. Perform the test described in *End to End testing*.
2. If you get a Connection Refused/error, check whether Tomcat is started:

```
$ sudo service tomcat7 status
```

3. If status stopped, start Tomcat:

```
$ sudo service tomcat7
```

4. If Tomcat fails to start, check for any Tomcat high-level error in Tomcat log directory: `/var/log/tomcat7`
5. If Tomcat is successfully started (no error in server logs), perform the test described in *End to End testing* again.
6. **If you still get a Connection Refused/error, check whether Tomcat is not listening on a different port:** `$ sudo netstat -lataupen|grep java`
7. If you still get a connection refused/error, especially if you are connecting remotely, check whether you are able to connect locally, then check the network link, i.e. whether any network filtering is in place on the host or on the access network, or other network issue: network interface status, DNS/IP adress resolution, routing, etc.
8. **If you get an error `404 Not Found`, make sure the webapp is deployed and enabled in Tomcat. Check for any webapp d** `/var/log/tomcat7/authzforce/error.log`.

Resource availability

To have a healthy enabler, the resource requirements listed in *System Requirements* must be satisfied, in particular:

- Minimum RAM: 4GB;
- Minimum CPU: 2.6 GHz;
- Minimum Disk space: 10 GB.

Remote Service Access

None.

Resource consumption

The resource consumption strongly depends on the number of concurrent clients and requests per client, the number of policy domains (a.k.a. tenants in this context) managed by the Authorization Server, and the complexity of the policies defined by administrators of each domain.

The memory consumption shall remain under 80% of allocated RAM. See *System Requirements* for the minimum required RAM.

The CPU usage shall remain under 80% of allocated CPU. See *System Requirements* for the minimum required CPU.

As for disk usage, at any time, there should be 1GB free space left on the disk.

I/O flows

- HTTPS flows with possibly large XML payloads to port 8080;
- HTTP flow to port 8080.

The port 8080 can be replaced by any other port Tomcat is listening to for HTTP connections to the webapp.

Appendix

Security setup for production

You have to secure the environment of the application server and the server itself. Securing the environment of a server in general will not be addressed here because it is a large subject for which you can find a lot of public documentation. You will learn about perimeter security, network and transport-level security (firewall, IDS/IPS...), OS security, application-level security (Web Application Firewall), etc. For instance, the ''NIST Guide to General Server Security" (SP 800-123) is a good start.

Server Security Setup

For more Tomcat-specific security guidelines, please read Tomcat 7 Security considerations.

For security of communications (confidentiality, integrity, client/server authentication), it is also recommended to enable SSL/TLS with PKI certificates. The first step to set up this is to have your Certification Authority (PKI) issue a server certificate for your AuthZForce instance. You can also issue certificates for clients if you want to require client certificate authentication to access the AuthZForce server/API. If you don't have such a CA at hand, you can create your own (a basic one) with instructions given in the next section.

Certificate Authority Setup

If you have a CA already, you can skip this section. So this section is about creating a local Certificate Authority (CA) for issuing certificates of the Authorization Server and clients, for authentication, integrity and confidentiality purposes. **This procedure requires using a JDK 1.7 or later.** (For the sake of simplicity, we do not use a subordinate CA, although you should for production, see keytool command example, use the `pathlen` parameter to restrict number of subordinate CA, `pathlen=0` means no subordinate.)

1. Generate the CA keypair and certificate on the platform where the Authorization Server is to be deployed (change the validity argument to your security requirements, example here is 365 days):

```
$ keytool -genkeypair -keystore taz-ca-keystore.jks -alias taz-ca -dname
↪"CN=Thales AuthzForce CA, O=FIWARE" \
    -keyalg RSA -keysize 2048 -validity 365 -ext bc:c="ca:true,pathlen:0"
```

2. Export the CA certificate to PEM format for easier distribution to clients:

```
$ keytool -keystore taz-ca-keystore.jks -alias taz-ca -exportcert -rfc > taz-ca-
↪cert.pem
```

Server SSL Certificate Setup

For Tomcat 7, refer to the Tomcat 7 SSL/TLS Configuration HOW-TO.

User and Role Management Setup

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on what parts of API, we need to have access to user identity and attributes and assign proper roles to them. These user and role management features are no longer supported by the AuthZForce server itself, but should be delegated to the Identity Management GE.

Domain Role Assignment

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on what parts of API, we need to have access to user identity and attributes and assign proper roles to them. These user role assignment features are no longer supported by the AuthZForce server itself, but should be delegated to the Identity Management GE.

Performance Tuning

For Tomcat and JVM tuning, we strongly recommend reading and applying - when relevant - the guidelines from the following links:

- Performance tuning best practices for VMware Apache Tomcat;
- How to optimize tomcat performance in production;
- Apache Tomcat Tuning Guide for REST/HTTP APIs.

Last but not least, consider tuning the OS, hardware, network, using load-balancing, high-availability solutions, and so on.

AuthZForce - User and Programmers Guide

AuthZForce is the reference implementation of the Authorization PDP GE. In this regard, it provides an API to manage XACML-based access control policies and provide authorization decisions based on such policies and the context of a given access request. This guide explains how to use the API.

Background and Detail

This User and Programmers Guide relates to the reference implementation of the Authorization PDP GE which is part of FIWARE Security Architecture. Please find more information about this Generic Enabler in the following Open Specification.

User Guide

Since the Authorization PDP is a Generic Enabler which provides backend functionality to other applications (e.g. Generic Enablers or end user facing applications) and security administrators, we do not distinguish between the User and Programmers Guide. Please refer to the Programmers Guide section for more information.

Programmer Guide

AuthZForce provides the following APIs:

- PDP API (PDP = Policy Decision Point in the XACML terminology): provides an API for getting authorization decisions computed by a XACML-compliant access control engine;
- PAP API (PAP = Policy Administration Point in XACML terminology): provides API for managing XACML policies to be handled by the Authorization Service PDP.

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available on the FIWARE catalogue in the package `FIWARE-AuthorizationPDP-REST-API-Model-XXX-src.zip` where `XXX` is the current version.

XACML is the main international OASIS standard for access control language and request-response formats, that addresses most use cases of access control. AuthZForce supports the full core XACML 3.0 language; therefore it allows to enforce very generic and complex access control policies.

Attribute-Based Access Control

AuthZForce provides Attribute-Based Access Control. To understand what is meant by "attribute" in the context of access control, below is the list of categories of attributes identified by the XACML standard:

- Subject attributes: the subject is an actor (human, program, device, etc.) requesting access to a resource; attributes may be user ID, Organization, Role, Clearance, etc.
- Resource attributes: the resource is a passive entity (from the access control perspective) on which subject requests to act upon (e.g. data but also human, device, application, etc.); resource attributes may be resource ID, URL, classification, etc.
- Action attributes: the action is the action that the subject requests to perform on the resource (e.g. create, read, delete); attributes may be action ID, parameter A, parameter B, etc.
- Environment attributes: anything else, e.g. current time, CPU load of the PEP/PDP, global threat level, etc.

Policy Administration API

The PAP is used by policy administrators to manage the policy repository from which the PDP loads the enforced policies. The PAP supports multi-tenancy in the form of generic administration domains that are separate from each other. Each policy administator (except the Superadmin) is in fact a domain administrator, insofar as he is allowed to manage the policy for one or more specific domains. Domains are typically used to support isolation of tenants (one domain per tenant).

Policy Management

The PAP provides a RESTful API for creating/updating policies for a specific domain, i.e. the top-level a.k.a. root XACML PolicySet of the domain. HTTP requests to this API must be formatted as follows:

- Method: PUT
- Path: /domains/{domainId}/pap/policySet
- **Headers:**
  - Content-Type: application/xml
  - Accept: application/xml

- Body: XACML PolicySet as defined in the XACML 3.0 schema.

Example of request given below:

```
PUT /domains/3b39dad9-1380-4c5b-8662-50cac998c644/pap/policySet
HTTP/1.1
Host: 127.0.0.1:8080
Accept: application/xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/xml; charset=UTF-8
Content-Length: 2631

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicySetId="P1"
    Version="1.0" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
↪algorithm:deny-unless-permit">
    <Description>Sample PolicySet</Description>
    <Target />
    <Policy PolicyId="MissionManagementApp" Version="1.0"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
↪algorithm:deny-unless-permit">
        <Description>Policy for MissionManagementApp</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
↪">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
↪#string">MissionManagementApp</AttributeValue>
                        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.
↪0:attribute-category:resource"
                            AttributeId="urn:oasis:names:tc:xacml:1.
↪0:resource:resource-id" DataType="http://www.w3.org /2001/XMLSchema#string"
                            MustBePresent="true" />
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Rule RuleId="MissionManager_role_can_manage_team" Effect="Permit">
            <Description>Only MissionManager role authorized to manage the mission team
↪</Description>
            <Target>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
↪#string">Team</AttributeValue>
                            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.
↪0:attribute-category:resource"
                                AttributeId="urn:thales:xacml:2.0:resource:sub-
↪resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"
                                MustBePresent="true" />
                        </Match>
                    </AllOf>
                </AnyOf>
                <AnyOf>
                    <AllOf>
```

```
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
↪#string">manage</AttributeValue>
                            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.
↪0:attribute-category:action"
                                AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"
                                MustBePresent="true" />
                        </Match>
                    </AllOf>
                </AnyOf>
            </Target>
            <Condition>
                <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
                    <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal" />
                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪MissionManager</AttributeValue>
                    <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.
↪0:subject:role"
                        DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="false"
                        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
↪subject" />
                </Apply>
            </Condition>
        </Rule>
    </Policy>
</PolicySet>
```

The HTTP response status is 200 if the policy has been successfully created/updated. It is not possible to delete a policy as a minimal policy must always be in place. If you want a *Permit All* (resp. *Deny All*), you have to update with such a policy: Target All, no condition, effect is Permit (resp. Deny).

Response (body is the PolicySet uploaded in the request):

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 2631
Date: Mon, 03 Dec 2014 10:12:43 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet ...
...content omitted...
</PolicySet>
```

Re-usable Policies (e.g. for Hierarchical RBAC)

The PAP provides a RESTful API for creating/updating <PolicySet>s that can be referred to from the root <PolicySet> for inclusion. This allows to include/reuse a given <PolicySet>s from multiple points of the domain's <PolicySet>, by means of XACML <PolicySetIdReference>s. One major application of this is Hierarchical RBAC. You can refer to the ''Core and hierarchical role based access control (RBAC) profile of XACML v3.0" specification for how to achieve Hierarchical RBAC with <PolicySetIdReference>s. HTTP requests to this API must be formatted as follows:

- Method: PUT
- Path: /domains/{domainId}/pap/refPolicySets
- **Headers:**
  - Content-Type: application/xml
  - Accept: application/xml
- Body: 0 or more XACML <PolicySet>s in a <policySets> element from XML namespace `http://thalesgroup.com/authz/model`[1].

Example of request given below:

```
PUT /domains/3b39dad9-1380-4c5b-8662-50cac998c644/pap/refPolicySets
HTTP/1.1
Host: 127.0.0.1:8080
Accept: application/xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/xml; charset=UTF-8
Content-Length: 2631

<?xml version="1.0" encoding="UTF-8"?>
<az:policySets xmlns:az="http://thalesgroup.com/authz/model/3.0" xmlns=
↪"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
   <PolicySet PolicySetId="PPS:Employee" Version="1.0"
       PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
↪algorithm:deny-unless-permit">
       <Description>Permissions specific to the Employee role</Description>
       <Target />
       <Policy PolicyId="PP:Employee" Version="1.0"
           RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
↪algorithm:deny-unless-permit">
           <Target />
           <Rule RuleId="Permission_to_create_issue_ticket" Effect="Permit">
               <Target>
                   <AnyOf>
                       <AllOf>
                           <Match MatchId="urn:oasis:names:tc:xacml:1.
↪0:function:string-equal">
                               <AttributeValue DataType="http://www.w3.org/2001/
↪XMLSchema#string">https://acme.com/ticketmanagementservice/tickets</AttributeValue>
                               <AttributeDesignator Category=
↪"urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                                   AttributeId="urn:oasis:names:tc:xacml:1.
↪0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"
                                   MustBePresent="true" />
                           </Match>
                       </AllOf>
                   </AnyOf>
                   <AnyOf>
                       <AllOf>
                           <Match MatchId="urn:oasis:names:tc:xacml:1.
↪0:function:string-equal">
                               <AttributeValue DataType="http://www.w3.org/2001/
↪XMLSchema#string">POST</AttributeValue>
                               <AttributeDesignator Category=
↪"urn:oasis:names:tc:xacml:3.0:attribute-category:action" AttributeId=
↪"urn:oasis:names:tc:xacml:1.0:action:action-id"
```

[1] This is not a browsable URL, only an XML namespace URI.

```
                                        DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="true" />
                            </Match>
                        </AllOf>
                    </AnyOf>
                </Target>
            </Rule>
        </Policy>
    </PolicySet>
    <!-- <PolicySet PolicySetId="PPS:Technician" PolicyCombiningAlgId=
↪"urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides">
        ...content omitted... </PolicySet> ... -->
</az:policySets>
```

The HTTP response status is 200 if the policy has been successfully created/updated. Response (body is the PolicySet uploaded in the request):

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 2631
Date: Mon, 03 Dec 2014 10:12:43 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<az:policySets ...
...content omitted (same as request body)...
</az:policySets>
```

AFTER uploading the ''policySets" above, the PolicySet ''PPS:Employee" becomes available for use in <PolicySetIdReference>s within any root <PolicySet> you upload from now on, with the API feature of the previous section of this guide. For example, now you can use such a root policySet (bare the <PolicySetIdReference> in particular):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://
↪www.w3.org/2001/XMLSchema-instance"
    PolicySetId="root:policyset" Version="1.0"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-
↪unless-permit">
    <Description>Root PolicySet</Description>
    <Target />
    <PolicySet PolicySetId="RPS:Employee" Version="1.0"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
↪algorithm:deny-unless-permit">
        <Description>
            Employee Role PolicySet
        </Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
↪">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
↪#string">Employee</AttributeValue>
                        <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.
↪0:subject-category:access-subject" AttributeId="urn:oasis:names:tc:xacml:2.
↪0:subject:role"
                            DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="true" />
```

```
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
    </PolicySet>
    <PolicySet PolicySetId="RPS:Manager" Version="1.0"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
↪algorithm:deny-unless-permit">
        <Description>
            Manager Role PolicySet
        </Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
↪">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
↪#string">Manager</AttributeValue>
                        <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.
↪0:subject-category:access-subject" AttributeId="urn:oasis:names:tc:xacml:2.
↪0:subject:role"
                            DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="true" />
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Policy PolicyId="PP1:Manager" Version="1.0"
            RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
↪algorithm:deny-unless-permit">
            <Description>Permissions specific to Manager Role</Description>
            <Target />
            <Rule RuleId="Permission_to_create_new_project" Effect="Permit">
                <Target>
                    <AnyOf>
                        <AllOf>
                            <Match MatchId="urn:oasis:names:tc:xacml:1.
↪0:function:string-equal">
                                <AttributeValue DataType="http://www.w3.org/2001/
↪XMLSchema#string">https://acme.com/ticketmanagementservice/projects</AttributeValue>
                                <AttributeDesignator Category=
↪"urn:oasis:names:tc:xacml:3.0:attribute-category:resource" AttributeId=
↪"urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                                    DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="true" />
                            </Match>
                        </AllOf>
                    </AnyOf>
                    <AnyOf>
                        <AllOf>
                            <Match MatchId="urn:oasis:names:tc:xacml:1.
↪0:function:string-equal">
                                <AttributeValue DataType="http://www.w3.org/2001/
↪XMLSchema#string">POST</AttributeValue>
                                <AttributeDesignator Category=
↪"urn:oasis:names:tc:xacml:3.0:attribute-category:action" AttributeId=
↪"urn:oasis:names:tc:xacml:1.0:action:action-id"
```

```
                                    DataType="http://www.w3.org/2001/XMLSchema#string"␣
↪MustBePresent="true"/>
                            </Match>
                        </AllOf>
                    </AnyOf>
                </Target>
            </Rule>
        </Policy>
        <!-- This role is senior to the Employee role, therefore includes the Employee␣
↪role Permission PolicySet -->
        <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
    </PolicySet>
</PolicySet>
```

Policy Decision API

The PDP API returns an authorization decision based on the currently enforced policy, access control attributes provided in the request and possibly other attributes resolved by the PDP itself. The Authorization decision is typically Permit or Deny. The PDP is able to resolve extra attributes not provided directly in the request, such as the current date/time (environment attribute).

The PDP provides an HTTP RESTful API for requesting authorization decisions. The HTTP request must be formatted as follows:

- Method: POST
- Path: /domains/{domainId}/pdp
- **Headers:**
  - Content-Type: application/xml
  - Accept: application/xml
- Body: XACML Request as defined in the XACML 3.0 schema.

The HTTP response body is a XACML Response as defined in the XACML 3.0 schema.

Example of request given below:

```
POST /domains/3b39dad9-1380-4c5b-8662-50cac998c644/pdp
HTTP/1.1
Host: 127.0.0.1:8080
Accept: application/xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/xml; charset=UTF-8
Content-Length: 954

<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Request xmlns='urn:oasis:names:tc:xacml:3.0:core:schema:wd-17' CombinedDecision=
↪"false"
    ReturnPolicyIdList="false">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
↪">
        <Attribute AttributeId='urn:oasis:names:tc:xacml:1.0:subject:subject-id'
            IncludeInResult="false">
            <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>joe</
↪AttributeValue>
```

```
      </Attribute>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"␣
↪IncludeInResult="false">
          <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>Manager
↪</AttributeValue>
      </Attribute>
   </Attributes>
      <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource
↪">
          <Attribute AttributeId='urn:oasis:names:tc:xacml:1.0:resource:resource-id'
              IncludeInResult="false">
              <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>
↪MissionManagementApp</AttributeValue>
          </Attribute>
          <Attribute AttributeId='urn:thales:xacml:2.0:resource:sub-resource-id'␣
↪IncludeInResult="false">
              <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>Team
↪</AttributeValue>
          </Attribute>
      </Attributes>
      <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
          <Attribute AttributeId='urn:oasis:names:tc:xacml:1.0:action:action-id'
              IncludeInResult="false">
              <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>
↪manage</AttributeValue>
          </Attribute>
      </Attributes>
      <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
↪category:environment" />
</Request>
```

Response:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 355
Date: Mon, 03 Dec 2014 14:06:26 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
   <Result>
      <Decision>Permit</Decision>
      <Status>
          <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok" />
      </Status>
   </Result>
</Response>
```

Integration with the IdM GE (e.g. for OAuth)

The easy way to integrate with IdM is to delegate the integration to the PEP up-front, i.e. we assume the PEP got all the required IdM-related info and forwards it to the Authorization PDP in the XACML request; the PEP Proxy by UPM can provide such a feature.

Software Libraries for clients of AuthZForce or other Authorization PDP GEis

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available on the FIWARE catalogue in the package `FIWARE-AuthorizationPDP-REST-API-Model-XXX-src.zip`, where `XXX` is the current version. Therefore, you can use any WADL-supporting REST framework for clients; for instance in Java: Jersey, Apache CXF. From that, you can use WADL-to-code generators to generate your client code. For example in Java, 'wadl2java' tools allow to generate code for JAX-RS compatible frameworks such as Apache CXF and Jersey. Actually, we can provide a CXF-based Java library created with this tool to facilitate the development of clients.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search