Global Privacy Policy Technical Framework (GPPTF) Implementation Guide

FINAL DRAFT 2012-05-14

Table of Contents

1 Introduction

The purpose of this Implementation Guide is to build on the Global Privacy Policy Technical Framework (GPPTF)[1] Implementation Primer[2] and provide detailed instruction on implementing externalized[3] access control and privacy policy services. The GPPTF Implementation Primer provides high-level technical guidance on steps for analyzing privacy policies and the software components needed to implement an externalized access control and privacy policy service. This guide builds upon the GPPTF Implementation Primer by providing a "hands-on" technical tutorial for implementing such services in an enterprise. The tutorial references are supplemented by a downloadable, executable sample information-sharing application. The downloadable software application provides a simulated environment in which to study and practice various skills that are required for the implementation of externalized access control within production-quality applications.

1.1 Scope of This Guide

The scope of this Implementation Guide includes the following topics:

1. The implementation of an externalized authorization and privacy policy service framework, based on GPPTF and the eXtensible Access Control Markup Language (XACML);
2. The development of XACML policies based on source policies, including aggregating multiple policies from multiple levels of authority;
3. The configuration of a XACML policy evaluation engine;
4. The construction of XACML access requests, and the parsing of XACML decision responses;
5. The integration of the policy framework with a Global Federated Identity and Privilege Management (GFIPM) web service;
6. The integration of the policy framework with a National Information Exchange Model (NIEM) Information Exchange Package Description (IEPD); and
7. The handling of policy obligations.

[1] See http://it.ojp.gov/docdownloader.aspx?ddid=1195 for the Global Privacy Policy Technical Framework document.
[2] [Add a URL or other reference to the GPPTF Implementation Primer here.]
[3] The term "externalized" in this context means *external to any specific business application*.

The following items are out-of-scope:

1. The development of source privacy policies[4];
2. The assessment of existing information sharing systems to determine their level of compatibility with an externalized access control and privacy policy service framework[5];
3. Details about developing web service components that conform to the GFIPM suite of standards[6];
4. The creation of a NIEM IPED[7]; and
5. Other security issues, such as network security, physical security, system administration, governance, and management.

1.2 Intended Audience

The intended audience for this guide is anybody who plays a role in the implementation of information sharing computer systems. This includes the following categories of individuals:

1. **IT Directors** who manage and oversee the design, implementation, and operation of IT systems;
2. **Enterprise architects** who develop and enforce business process and IT standards;
3. **Policy analysts** who develop organizational security and privacy policies and multiagency agreements; and
4. **Project managers, application architects, and technologists** who manage, design, implement, or support IT systems.

1.3 Primer on Related Topics and Technologies

This section provides a primer on topics and technologies that are either directly or tangentially related to this Implementation Guide. Pointers and references to additional information are provided where appropriate.

1.3.1 Terms and Definitions Related to Authorization, Access Control and Privacy Policies

Authorization refers to the process of granting privileges to subjects (users and other entities) of a system, and also refers to the privileges themselves. *Access*

[4] Development of source privacy policies is covered in the *Global Privacy and Civil Liberties Policy Development Guide and Implementation Templates* document. See http://www.it.ojp.gov/privacy for more information.
[5] Assessment of existing systems and applications is covered in the *Privacy Policy Automation - Readiness Self-Assessment* document. See [URL or reference] for more information.
[6] See http://gfipm.net/ for more information about GFIPM.
[7] The *IEM Model Package Description Specification* document contains more information about creating NIEM IEPDs. See http://reference.niem.gov/niem/specification/model-package-description/1.0/model-package-description-1.0.pdf for more information.

control is the process of controlling access by subjects to protected resources, and this process takes into account authorizations granted by one or more authorities. The term *electronic access control* is used within the context of computerized information resources. All references to the term *access control* in this guide correspond to electronic access control, unless explicitly stated otherwise. An access control mechanism is a software component that is responsible for mediating access to protected information resources. It is configured with a set of access control rules: statements declaring that a set of entities can or cannot perform a set of actions on a set of resources under a set of conditions. The set of rules implemented by an access control mechanism comprise the *access control policy* for that mechanism. When an entity requests access to a resource, the access control mechanism evaluates its policy, determines whether to grant access, and enforces the decision. This guide covers the development of electronic access control policies based on *source policies*: natural language (i.e., "plain English") expressions of statutes, regulations, and other laws that govern the operation of government and private agencies and organizations in the context of information sharing.

The access control policies covered in this guide use the *attribute-based access control* (ABAC) model, in which access control rules are specified using *attributes* of entities, resources, actions, and environmental conditions. In ABAC, an *attribute* is a quality or feature regarded as an inherent part of somebody or something for access control purposes. For example, there may exist a "citizenship" attribute for users of a system and access to resources may be restricted to users who are citizens of a particular nation.

As stated in the Global Privacy and Civil Liberties Policy Development Guide, the term *privacy* refers to individuals' interests in preventing the inappropriate collection, use, and release of *personally identifiable information (PII)*. PII is defined as "one or more pieces of information that, when considered together or when considered in the context of how it is presented or how it is gathered, are sufficient to specify a unique individual"[8]. These pieces of information can include personal characteristics, a unique set of numbers or characters assigned to a specific individual, descriptions of events or points in time, and descriptions of locations or places.

A *privacy policy* is a set of rules that limit the collection, release, or processing of PII to only those entities that have a legitimate, authorized purpose. The privacy policy concept is a subset of the concept of access control policy (i.e., an access control policy limits access to information in general), and we can therefore use the same tools and frameworks to manage and enforce both types of policies.

1.3.2 Global Reference Architecture and Global Privacy Policy Technical Framework

[8] See http://www.it.ojp.gov/privacy.

The Global Reference Architecture (GRA) is a service-oriented reference architecture for information sharing across geo-political boundaries. The GRA adheres to the principles of service-oriented architecture (SOA) and provides guidance to implementers on how to develop information sharing solutions and applications based on loosely coupled services that are implemented using standard technologies.

The Global Privacy Policy Technical Framework (GPPTF) document explores various approaches and alternatives to resolving technical and interoperability challenges in automating privacy policy enforcement within and between information-sharing enterprises. It defines a high-level architecture for implementing and enforcing privacy policies, and it also provides guidance on authoring electronic privacy policy statements based on applicable source policies.

The GPPTF advocates the implementation of privacy policy statements by using *identity credentials*, *resource content metadata*, *action verbs*, and *environmental conditions* to express which access requests are to be allowed. It also accommodates the implementation of *obligations*: actions that the user or information sharing service must perform to be in compliance with the policy.

The GPPTF uses externalized policy services that ensure that electronic information access requests for information protected by a privacy policy are authenticated, authorized, and audited before access is granted. These policy services rely on two logical components of the framework called a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). The PDP evaluates each access request against a policy, and provides a decision on whether to grant access. The PEP intercepts each access request, retrieves an access decision on the request from the PDP, enforces the decision, and fulfills any policy obligations that apply to the decision. Figure 1 depicts the Privacy Policy Technical Framework.

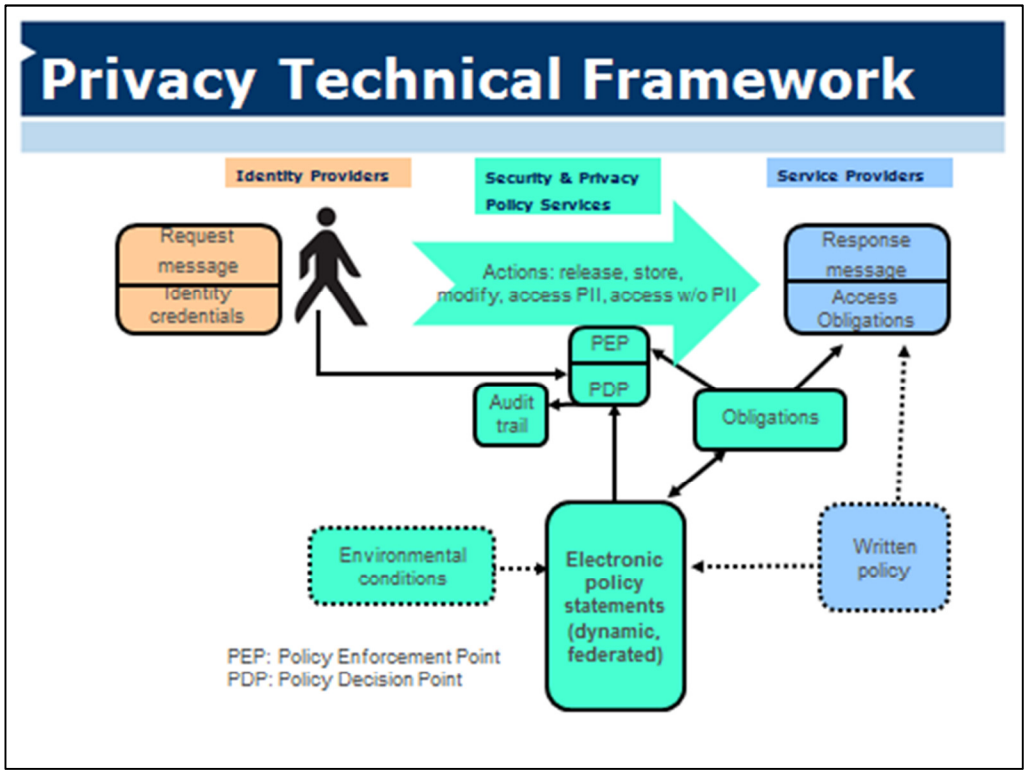

Figure 1: Privacy Policy Technical Framework

The framework presented in this Implementation Guide conforms to the GPPTF.

1.3.3 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML)[9] is an XML-based industry standard that includes: (1) a language for expressing attribute-based access control policies; (2) a normative specification for the evaluation of an access request against a policy; and (3) message formats for expressing access requests and access control decisions.

The basic building blocks of XACML policies and access requests are attributes. In XACML, an attribute is a name-value pair that expresses a characteristic of a *subject* (a user or entity attempting to access a resource), a *resource* (a sensitive data item or service protected by an access control system), an *action* (the type of operation to be performed on a resource by a subject), or the system *environment* (conditions external to the system that affect access control decisions). XACML also supports the specification of policy *obligations*, which may be attached to specific XACML policy statements.

[9] See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml for more information on XACML.

Although XACML provides a standard for expressing attributes, it does ***not*** provide any domain specific dictionary of attributes. Defining domain specific attributes and values is a task left to each community of interest. For example, in the justice community, the Global Federated Identity and Privilege Management (GFIPM) metadata standard defines many relevant characteristics and values. Other communities – human services, education, health care, etc. – are also developing metadata standards that will assist in enforcing their source policies.

1.3.4 Global Federated Identity and Privilege Management (GFIPM)

The Global Federated Identity and Privilege Management (GFIPM) program addresses the security requirements for the foundation of operational inter-enterprise justice federations. The conceptual foundation of GFIPM is the idea of federated identity and privilege management (FIPM), which provides the ability to separate the management of user identities and privileges from the management of the systems and applications in which those identities and privileges are used. Within a federation, identity providers (IDPs) manage user identities and service providers (SPs) manage applications and other information resources. Each system or application in a federation typically has its own set of business requirements and access control policies, and FIPM provides a cost-effective framework that enables these systems to be made available to federated users while still respecting their native requirements.

At the core of the GFIPM concept is the idea of collecting information attributes about users and sharing them with systems and applications in a trusted manner. These attributes serve as a framework for supporting various value-added features, including dynamic provisioning of local user accounts within applications, and federated authorization, in which an application can make access control decisions for users based on the attribute values provided during the attribute sharing process. In GFIPM, attributes capturing this information can be used to convey important facts about a user to the target application, thereby enabling the application to decide whether to permit or deny access without the need for manual intervention by a local security administrator.

One of the technical work products created through the GFIPM program is the GFIPM Metadata Specification[10], which expresses precise syntax and semantics for exchanging attribute data about users, system entities, information resources, information-sharing actions, and environmental conditions within an information-sharing federation. The GFIPM Metadata Specification provides a rich set of attributes with which to express attribute-based access control policies, and is the primary source of attributes for the policies discussed in this guide.

[10] See http://gfipm.net/standards/metadata/2.0/index.html for more information about the GFIPM Metadata Specification.

The GFIPM Web Services System-to-System Profile[11] is a normative technical document containing a set of Service Interaction Profiles (SIPs) that enable secure, interoperable, standards-based SOAP web services communication within a GFIPM federation. It conforms to the GRA and leverages other GFIPM core normative standards, including the GFIPM Metadata Specification and the GFIPM Cryptographic Trust Model[12], to further promote the GFIPM goals of secure, interoperable information sharing across trust domains at low-cost. The sample application that supplements this Implementation Guide conforms to one of the SIPs specified in the GFIPM Web Services System-to-System Profile.

1.3.5 National Information Exchange Model

The National Information Exchange Model (NIEM)[13] is a set of XML-based data objects that can be used to implement standards-based information exchanges. NIEM defines precise syntax and semantics, to enable accurate machine interpretation and processing of complex XML documents across many information-sharing domains. It is the predominant data model used for information exchange within the U.S. justice community. A NIEM Information Exchange Package Documentation (IEPD)[14] provides a mechanism for normatively specifying a NIEM-conformant set of data structures and messages that can be used for specific data exchanges. NIEM IEPDs effectively provide a standard interface between systems at the data payload level.

The NIEM community is currently investigating the addition of domain specific metadata that could supply some data resource attributes for XACML rules.

1.3.6 Illustration and Summary

In a typical justice use case, sworn law enforcement officers are appropriately permitted to review personally identifiable information in criminal intelligence databases for the purpose of criminal investigations. Table 1 shows how the components in the GPPTF map to XACML terminology and to example attribute values that would be relevant to a source policy related to criminal intelligence information exchange.

[11] See http://gfipm.net/standards.html for more information about the GFIPM Web Services System-to-System Profile.
[12] See http://it.ojp.gov/docdownloader.aspx?ddid=1338 for more information about the GFIPM Cryptographic Trust Fabric.
[13] See https://www.niem.gov/ for more information on NIEM.
[14] See http://reference.niem.gov/niem/specification/model-package-description/1.0/model-package-description-1.0.pdf for the Model Package Description Specification which contains the specification for IEPDs.

GPPTF Component	XACML Terminology	Example Attribute Values
Identity Credentials	Subject Attributes	Sworn Law Enforcement Officer Indicator[15]
Content Metadata	Resource Attributes	Criminal Intelligence Data Indicator[16]
Action (Verbs)	Action Attributes	Action Type [17](Read, Delete, etc.)
Environmental Conditions	Environment Attributes	Local Weather Conditions[18]; Homeland Security Threat Level[19]
Obligations	Obligations	The Data Requestor must not further disseminate the requested data; The Data Requestor must delete the requested data within 30 days.

Table 1: Mapping from GPPTF Terms to XACML Terms to GFIPM Metadata

The examples for XACML subject, resource, action, and environment attributes are from the GFIPM Metadata Specification. The GFIPM Metadata Specification does not include obligation metadata; however, the Global Information Sharing Standards Toolkit (GISST) Obligations Task Team is currently working to create standards for defining and handling obligations.

1.4 How to Use This Document

This document is in part a reference manual on a XACML-based, enterprise access control framework that conforms to the GPPTF, and in part a how-to guide on implementing a subset of that framework. Section 2 of this document describes the full XACML-based enterprise access control framework in detail. Section 3 is the step-by-step implementation tutorial. The tutorial covers three major policy implementation objectives:

1. Creating XACML policies from source policies;
2. Implementing and configuring various software components within the enterprise access control framework; and
3. Integrating the framework with a sample information sharing application that is available for download by the reader.

Through a series of "lessons", the tutorial guides the reader through a step-by-step process to complete each of above three tasks on the sample application, to achieve a working prototype. Section 4 provides a "gap analysis" between the prototype

[15] See http://gfipm.net/standards/metadata/2.0/user/SwornLawEnforcementOfficerIndicator.html.
[16] See http://gfipm.net/standards/metadata/2.0/resource/CriminalIntelligenceDataIndicator.html.
[17] See http://gfipm.net/standards/metadata/2.0/action/ActionType.html.
[18] See http://gfipm.net/standards/metadata/2.0/environment/LocalWeatherConditions.html.
[19] See http://gfipm.net/standards/metadata/2.0/environment/HomelandSecurityThreatLevel.html.

developed in Section 3 and the full framework described in Section 2. It also provides various recommendations for improving and extending the prototype to better meet the requirements of the full enterprise framework. Section 0 provides pointers to related literature for further reading.

2 XACML Reference Architecture

This Section presents a canonical "XACML Reference Architecture" for managing security and privacy policies within the enterprise. Section 2.1 describes the prerequisite knowledge that the reader should have prior to reading the subsequent sections. Section 2.2 provides an overview of the components and technologies that comprise the reference architecture, and Section 2.3 provides a guided tour of the architecture by describing how it supports several basic use cases.

2.1 Prerequisites

The description of the XACML Reference Architecture in the following sections assumes that the reader already has a basic understanding of the following topics.

1. User Authentication and Identity Management
2. Authorization and Access Control Policy
3. Cryptographic Primitives (data confidentiality and data integrity)
4. eXtensible Markup Language (XML)

Specific components and technologies are briefly described when they are introduced, and the brief descriptions in this document should provide enough detail for the reader to develop a "big picture" understanding of the XACML Reference Architecture. For each technology, links to supplementary information resources are provided.

2.2 Major Architectural Components and Technologies

This section and its subsections provide a brief introduction to the components and technologies used by the XACML Reference Architecture. Subsequent sections build on this section to describe the use cases enabled by the architecture and the integration points at which the architecture connects to its surrounding environment.

2.2.1 Core Architectural Components

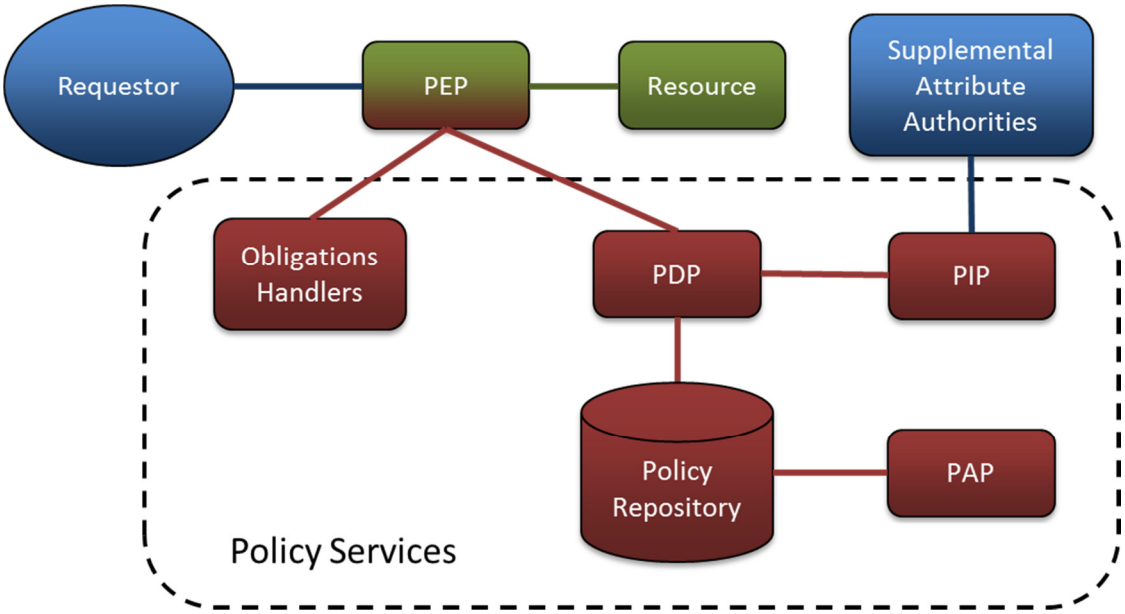

Figure 2: XACML Reference Architecture

Figure 2 depicts the XACML Reference Architecture. XACML-related components are shown in red, and application-specific components are green. The PEP is both green and red as it performs both application-specific and XACML functionality. The blue components are external to the reference architecture, and typically reside outside the trust boundary of the organization that is implementing the architecture to protect sensitive resources. The components within the dotted border comprise a "Policy Services" infrastructure that enables XACML-based protection of many applications and resources within an enterprise. A description of the core architectural components follows.

1. ***Data Requestor*** – The Data Requestor is an entity that makes a request to access a Data Resource. It may be a human user using a personal computing device (e.g. a web browser), a system acting on behalf of a human user, or a system acting on behalf of an entire organization. The Data Requestor may come from within the same enterprise in which the Data Resource is located, or it may come from another enterprise. Regardless of where the Data Requestor resides, the XACML architecture must ensure that the request to access the Data Resource does not succeed unless it is permitted by the access control policy.

2. ***Data Resource*** – The Data Resource is any object or collection of objects containing sensitive data, for which access is mediated by the XACML architecture. Types of access on a Data Resource may include the standard "CRUD" operations[20] on a single object, as well as searching or querying across a collection of multiple objects.

[20] The term "CRUD" – create, read, update, and delete – describes the four basic functions that can be performed on any persistent data object.

3. ***Policy Enforcement Point (PEP)*** – In a XACML architecture, the PEP acts as the "front door" to the system, from the perspective of the Data Requestor, and mediates access to the Data Resource. The PEP typically communicates with a Policy Decision Point (PDP), which renders access control decisions. The PEP enforces the PDP's decisions, permitting or denying access to the Data Resource. All communications between the PEP and PDP take the form of XACML requests and responses.

4. ***Policy Decision Point (PDP)*** – As described above, the PDP communicates with the PEP and makes access control decisions that the PEP subsequently enforces. The PDP receives access control requests in the form of XACML request messages, and renders access control decisions in the form of XACML responses. To render each access control decision, the PDP consults an access control policy that is specified using XACML and stored in a Policy Repository. Note that a single PDP may support multiple PEPs.

5. ***Policy Repository*** – The Policy Repository is a database that stores XACML policy rules and allows for efficient retrieval of the rules by the PDP at policy evaluation time.

6. ***Policy Information Point (PIP) and Supplemental Attribute Authorities*** – The PIP is an auxiliary service used by the PDP to retrieve the values of any supplemental attributes needed by the PDP for policy evaluation. To retrieve supplemental attribute data, the PIP may contact one or more Supplemental Attribute Authorities, which may reside either inside or outside the enterprise.

7. ***Obligation Handlers*** – The Obligation Handlers are a set of software modules that can be invoked by the PEP to fulfill various policy obligations, such as logging or notification. Obligations are described briefly in Section 1.3.3.

8. ***Policy Administration Point (PAP)*** – The PAP is an interface into the Policy Repository, through which policy administrators can manage XACML policies. Common management tasks include authoring, installing, updating, and deleting policies.

2.2.2 Core and Related Technologies

Because it is concerned with protecting sensitive data resources from unauthorized access, the XACML Reference Architecture relies on several fundamental security technologies. These technologies are introduced and briefly described here, along with pointers to information resources for more in-depth study of each technology.

1. ***eXtensible Access Control Markup Language (XACML)*** – XACML, which was previously introduced in Section 1.3.3, provides a robust, XML-based

standard language for expressing access control policies. It also provides a processing model for evaluating specific access requests and rendering decisions based on those policies. For more information about XACML, please see http://www.oasis-open.org/committees/xacml/.

2. ***Security Assertion Markup Language (SAML)*** – SAML provides a set of industry standard protocols and profiles for securely transmitting trusted assertions about subjects. These assertions typically are made by one organization and intended for use by another organization for the purpose of making access control decisions. The design of SAML makes it ideal for use in conjunction with XACML, because SAML provides a secure source of attribute data about a Data Requestor, and XACML uses attributes about the Data Requestor to render an access control decision. For more information about SAML, please see http://saml.xml.org/.

3. ***SOAP Web Services (WS-*)*** – WS-* comprises a set of composable technologies that build upon the SOAP and WSDL standards to implement an XML-based, enterprise-grade protocol suite for a service-oriented architecture. While the SOAP and WSDL standards define the basic format of a message and the basic interface of a service, respectively, other technologies in the WS-* suite provide various add-on features in an "a la carte" fashion. These add-on features include message confidentiality and integrity (via WS-Security), message reliability (via WS-ReliableMessaging), addressing and routing (via WS-Addressing), session management (via WS-SecureConversation), and many others. For more information about WS-*, please see the following links.

    a. SOAP: http://www.w3.org/TR/soap/
    b. WSDL: http://www.w3.org/TR/wsdl
    c. WS-Security: http://www.oasis-open.org/committees/wss/
    d. WS-ReliableMessaging: https://en.wikipedia.org/wiki/WS-ReliableMessaging
    e. WS-Addressing: http://www.w3.org/Submission/ws-addressing/
    f. WS-SecureConversation: https://en.wikipedia.org/wiki/WS-SecureConversation
    g. Other: https://en.wikipedia.org/wiki/List_of_web_service_specifications

4. ***Transport Layer Security (TLS)*** – TLS is an industry standard protocol for communicating securely using certificates within a public key infrastructure (PKI). Formerly known as Secure Sockets Layer (SSL), TLS is used by most web sites and applications that provide secure sessions. For more information about TLS, please see https://en.wikipedia.org/wiki/Transport_Layer_Security.

5. ***Public Key Infrastructure (PKI)*** – A PKI is a centrally managed trust infrastructure that uses public key cryptography to provide a framework within which secure point-to-point communications can be performed. For more information about PKI, please see https://en.wikipedia.org/wiki/Public_key_infrastructure.

2.3 Use Cases and Integration Points

This section builds upon Section 2.2 by describing the basic use cases that the XACML Reference Architecture must support. The two fundamental use cases are as follows.

1. Requestor Accesses Sensitive Resource
2. Administrator Creates Access Control Policy

In the following subsections, each use case is further subdivided into steps that illustrate the role played by each component of the reference architecture to support that use case. Also, each step corresponds to one or more "integration points" at which the components of the reference architecture communicate with each other. Important details about each integration point are discussed in the context of each step for each use case.

A recurring theme throughout each use case, and each step within each use case, is security and trust. Enterprises rely on this architecture to protect their sensitive resources by faithfully implementing their access control and privacy policy rules. It is therefore imperative that each component in the architecture operates and communicates with other components in accordance with basic best practices of computer security. Instead of repeating this common theme at each step, we describe here the basic security requirements that the participating components must meet, as well as the potential consequences of not meeting those requirements.

During each step, it is critical that the XACML architecture components establish mutual trust via a secure communication channel (e.g. TLS with mutual client/server authentication), to avoid eavesdropping as well as various communication-channel attacks (e.g. "man-in-the-middle" attacks, replay attacks, etc.) The requirement for a secure channel between communicating parties is always an important concern when devices communicate via a network.

2.3.1 Requestor Accesses Sensitive Resource

The most common "run-time" use case is when a person or system entity attempts to access a sensitive resource. The reference architecture must perform a series of steps to respond to the request in accordance with the access control policy that is in place to protect the resource.

Step 1: The Requestor connects to the PEP and makes a request to access a sensitive resource.

Figure 3 depicts the components that participate in this step.

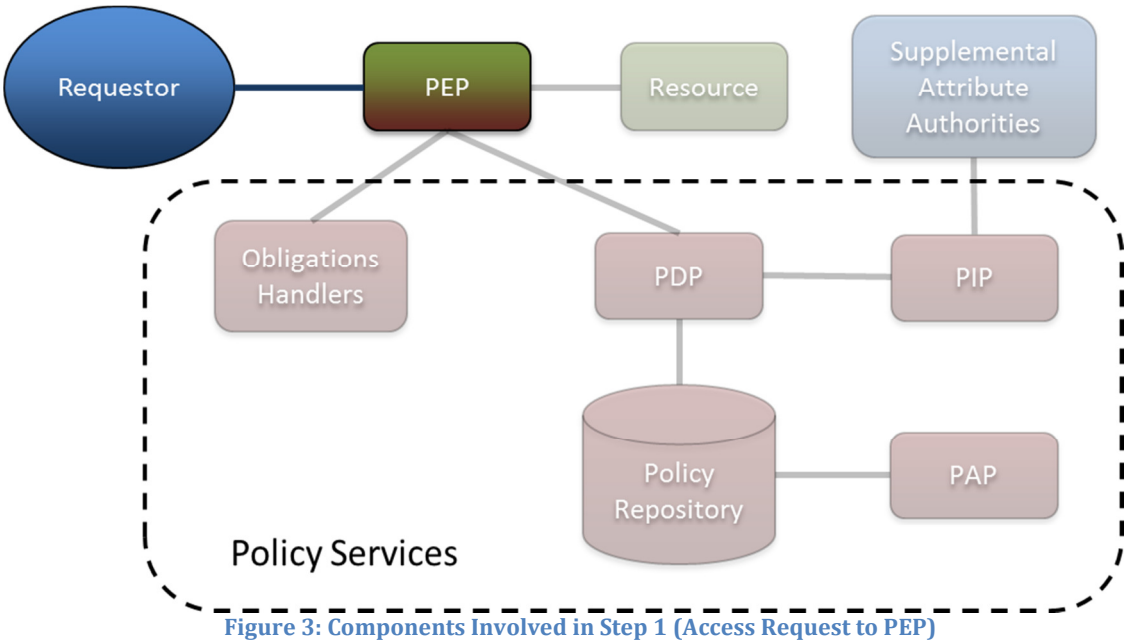

Figure 3: Components Involved in Step 1 (Access Request to PEP)

Requirements:

A. ***Secure Communication Channel between Requestor and PEP***

B. ***Trusted Requestor Attributes*** – The Requestor must present the PEP with a set of facts, or *attributes*, about its identity and credentials. The PEP must not only understand the semantic meaning of these attributes, but also trust that the attribute values are accurate. Note that the Requestor may come from within the same enterprise, or from another enterprise (e.g. another agency). Note also that the Requestor may be a user, a system acting on behalf of a user, or a system acting on behalf of an entire agency. [21]

C. ***Common Interface*** – The interface used between the Requestor and the PEP is outside the scope of the XACML Reference Architecture. The data and messages transferred over this common interface can be complex, structured objects such as IEPDs. The PEP may need to translate these data structures into a data model that can be used within XACML requests. This common interface is often based

[21] The challenge of providing the PDP with well-defined attributes about the Requestor, in a trusted manner, is a major undertaking, particularly when the Requestor comes from another agency. The GFIPM program has developed a robust solution to this problem. We recommend that implementers use the GFIPM solution, if possible, rather than invent their own solution to this problem. See http://gfipm.net/ for more information about GFIPM.

on the SOAP Web Services (WS-*) family of standards, but it can also be based on other technologies, such as REST web services or traditional client-server HTTP communications (e.g. web browser to web server).[22]

Step 2: The PEP formulates a XACML request and sends it to the PDP for evaluation.

Figure 4 depicts the components that participate in this step.

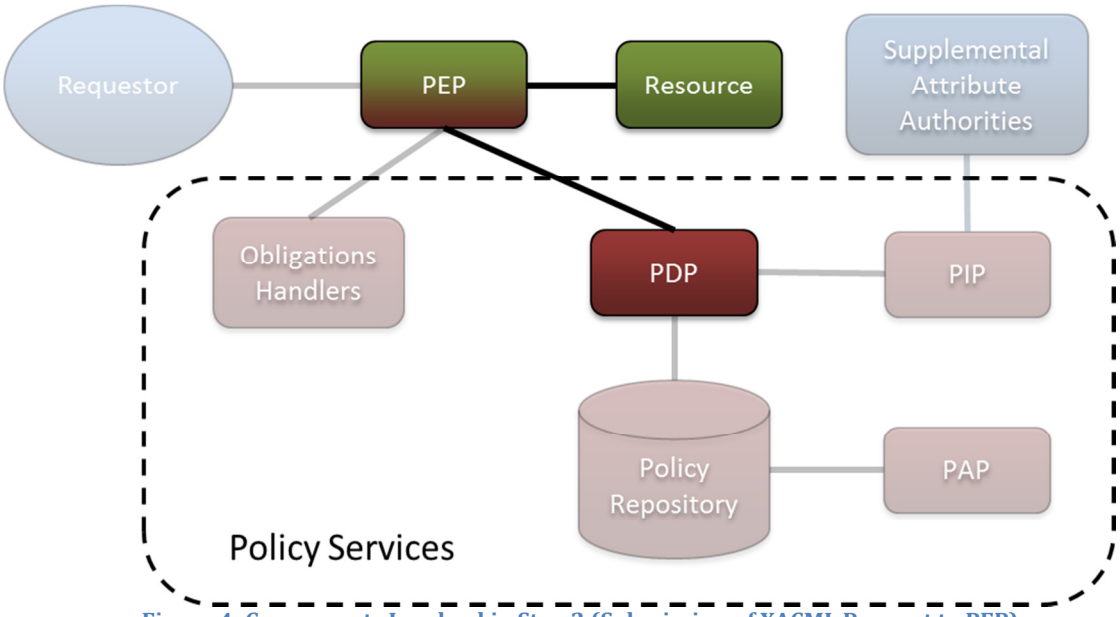

Figure 4: Components Involved in Step 2 (Submission of XACML Request to PEP)

Requirements:

A. ***Secure Communication Channels between PEP and PDP, and between PEP and Resource***

B. ***Translation from Application Protocol to XACML*** – The PEP must formulate a XACML request and populate the request with appropriate domain attributes in the XACML format. The PEP can obtain these attributes from three sources: (1) conversion of attributes retrieved from the Data Requestor into the XACML format; (2) creation of action attributes based on the action requested in Step 1; and (3) retrieval of resource attributes from the target data resource. Also, if required by the installed XACML policies, the PEP may need to include the content of the requested data in the XACML request.[23]

[22] The GFIPM suite of solutions includes a SOAP Web Services framework.
[23] For example, a XACML policy may need to know the identity of the subject of a requested record in order to make a decision, and the policy may be written to extract that identity directly out of the data record.

Step 3: The PDP retrieves the appropriate XACML policy from the Policy Repository.

Figure 5 depicts the components that participate in this step.

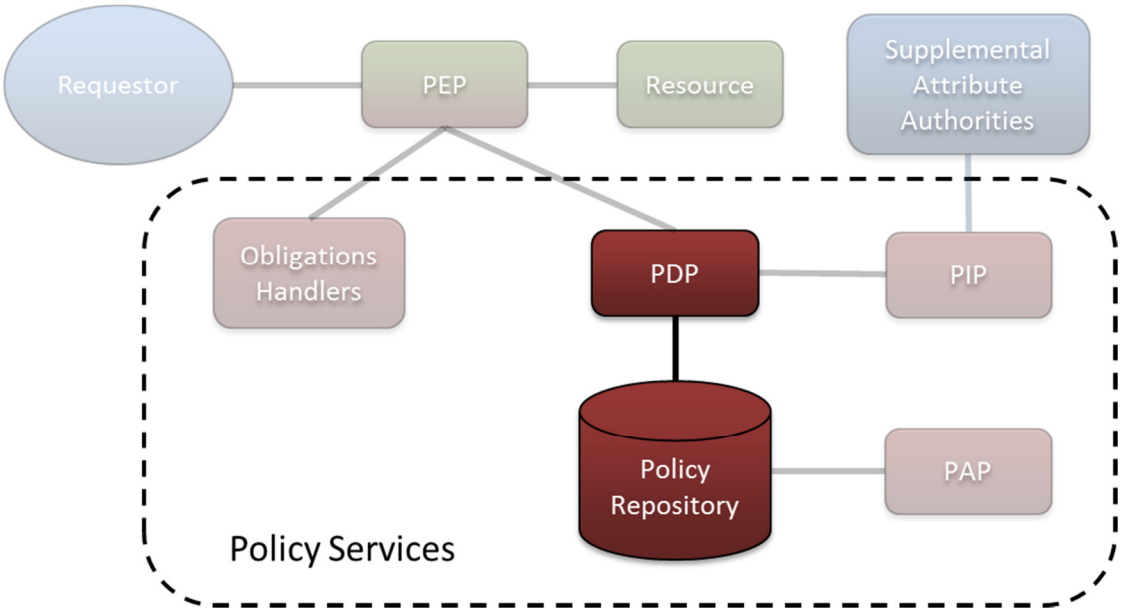

Figure 5: Components Involved in Step 3 (Loading XACML Policy into PDP)

Requirements:

A. ***Proper Configuration of the PDP with All Applicable Policies*** – All rules and policies that are applicable to the application must be properly aggregated into and loaded into the PDP, so the PDP can properly respond to any request from the PEP within the application.

Step 4: The PDP determines whether it must retrieve any supplemental attributes[24] before it can evaluate the XACML request, and retrieves those attributes via the PIP if necessary.

Figure 6 depicts the components that participate in this step.

[24] One example of a potential supplemental attribute is whether there is a current state of emergency.

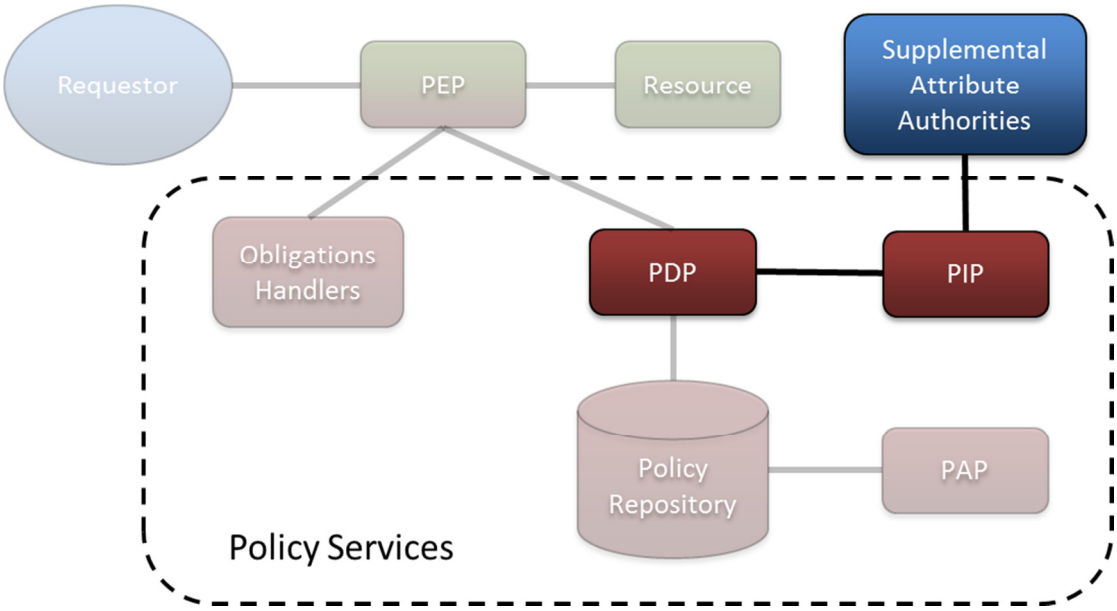

Figure 6: Components Involved in Step 4 (Retrieval of Supplemental Attributes)

Requirements:

A. ***Secure Communication channels between PDP and PIP, and between PIP and each Supplemental Attribute Authority***

B. ***Implementation of an Accurate and Efficient Attribute Retrieval Algorithm*** – The PIP must implement an accurate and efficient attribute retrieval algorithm, so it can determine which Supplemental Attribute Authority to contact for any given attribute, and dispatch the attribute request appropriately. This algorithm must ensure that the attribute retrieved pertains to the correct entity. (E.g. if multiple users named John Smith exist within the enterprise and its partner agencies, the algorithm must ensure that any supplemental attributes pertain to the correct John Smith.) Also, the algorithm must retrieve supplementary attributes efficiently, as the entire architecture may be blocked awaiting a synchronous response from the PIP.

Step 5: The PDP evaluates the request, produces a XACML response, and sends the response to the PEP.

Figure 7 depicts the components that participate in this step.

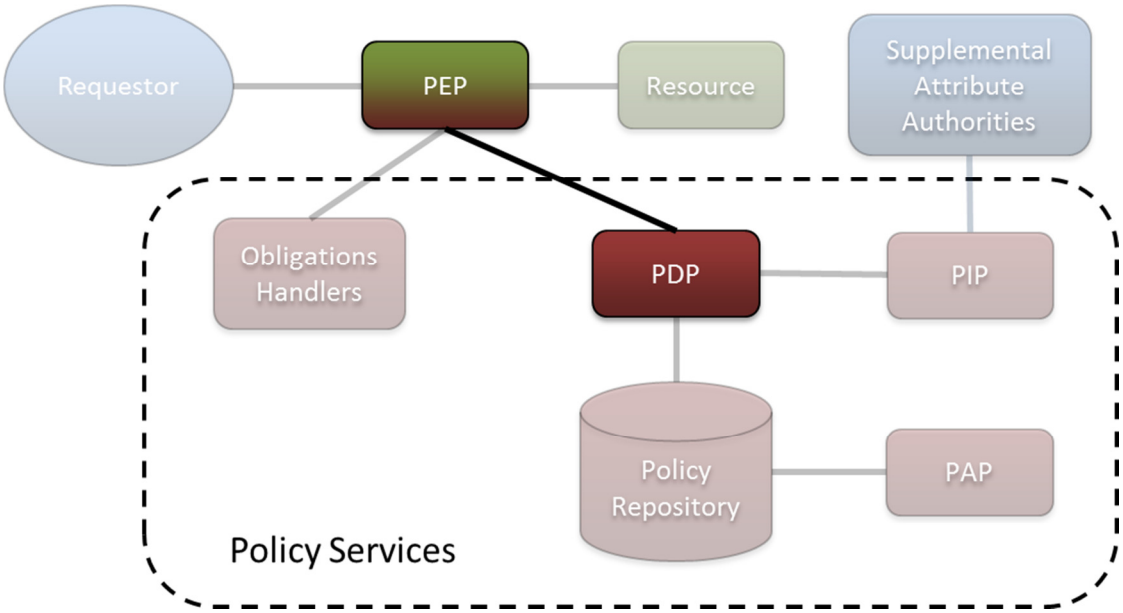

Figure 7: Components Involved in Step 5 (XACML Response to PEP)

Requirements:

A. ***A Secure Communication Transmission of the XACML Response to the PEP***

Step 6: The PEP handles any obligations associated with the response from the PDP.

Figure 8 depicts the components that participate in this step.

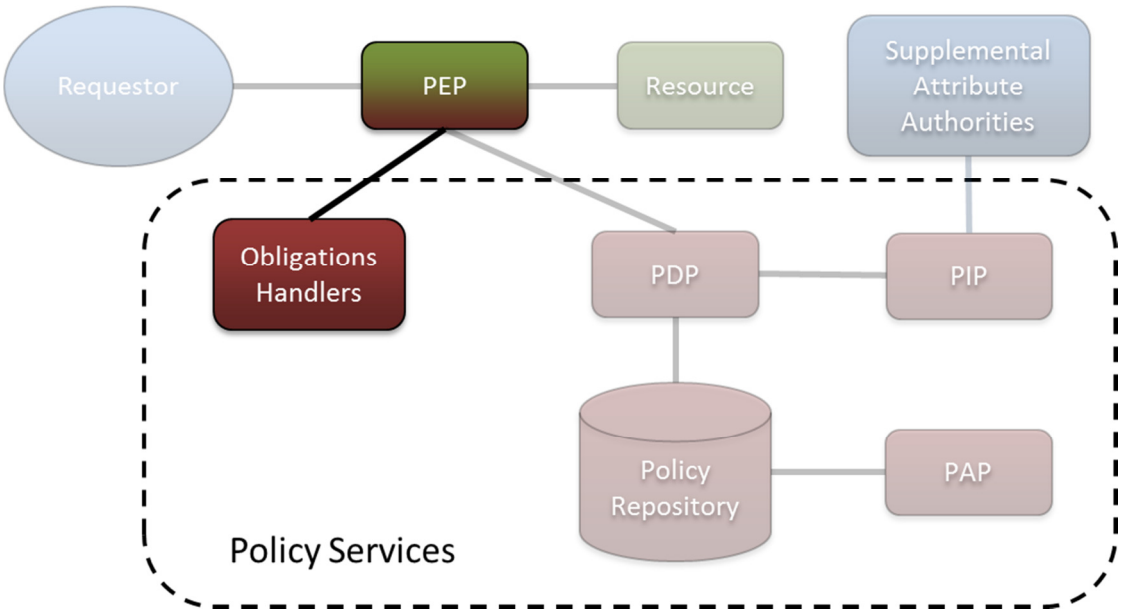

Figure 8: Components Involved in Step 6 (Obligation Handling)

Requirements:

A. ***Secure Communications Channel between PEP and Obligation Handlers***

B. ***Proper Entity Resolution for Entities Specified in Obligations*** – Some obligations require that a message be transmitted to a specific entity (e.g. a specific email address) to fulfill a requirement for notification (e.g. notification of the resource owner). The Obligation Handler for that obligation must be able to resolve the identity and location of that entity accurately based on the context provided to it by the PEP. An example of such an obligation is: "The owner of a data resource must be notified of every access to that resource".

C. ***Processing Model for Handling "Out-of-Band" Obligations*** – In the standard XACML obligation-processing model, the PEP is the obligor[25] for all obligations; in other words, the PEP fulfills all obligations. However, there exist other classes of obligations for which an entity other than the PEP is the obligor; these are called "out-of-band" obligations. An example of such an obligation is: "The data requestor must not further disseminate the data". In this example, the data requestor is the obligor. Proper handling of these out-of-band obligations requires the development and implementation of an obligation-processing model that is deemed acceptable by the appropriate policy authorities. The Global Federated Identity and Technical Privacy Task Team[26] is developing a standardized syntax and processing model for various types of policy obligations, including out-of-band obligations.

Step 7: The PEP performs the action requested by the Requestor, if authorized by the PDP, and responds to the Requestor.

Figure 9 depicts the components that participate in this step.

[25] "Obligor" is a legal term indicating the party obligated to fulfill an obligation.

[26] The Global Federated Identity and Technical Privacy Task Team was created in early 2012 as part of an effort to restructure and streamline the activities of the working groups within the Global Justice Information Sharing Initiative. This task team has subsumed the work of both the GFIPM Delivery Team and the Global Obligations Task Team. More information about the work of this and other Global task teams is available at http://www.globaljusticetools.net/.

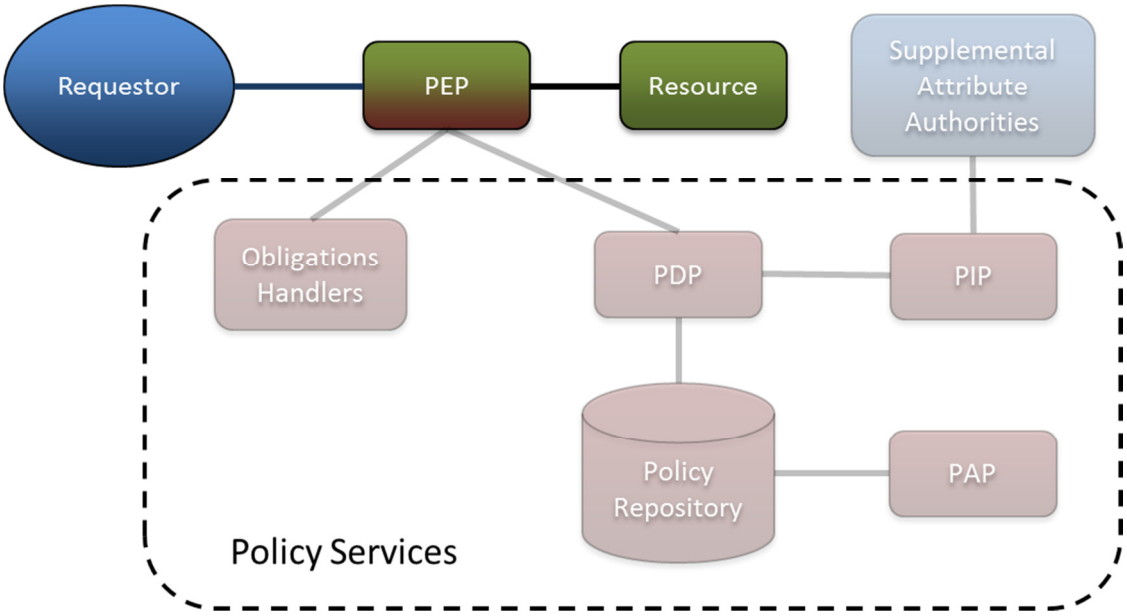

Figure 9: Components Involved in Step 7 (Response to the Requestor)

Requirements:

A. ***Resource-Level Response Processing*** – The PEP must understand and be capable of handling all the resource-level details for any action that the Requestor may request and any policy decision that the PDP may provide in response to that request.

B. ***Secure Communication Transmission of the PEP's Response to the Requestor***

2.3.2 Administrator Manages Access Control Policy

Unlike the previous use case, which describes how the reference architecture handles a request for access to sensitive data, this use case describes how the architecture supports the management of access control policies. This use case occurs on a relatively infrequent basis compared to the previous use case.

Step 1: The Policy Administrator defines a new attribute dictionary or updates the existing attribute dictionary as needed.

This step is not illustrated.

Requirements:

A. ***Attribute Dictionary with Appropriate Terms*** – The Policy Administrator can use an existing attribute dictionary, such as the GFIPM Metadata

Specification, if appropriate, or create a new dictionary with custom attributes or obligations. The Administrator may also extend an existing dictionary with new custom attributes if necessary. Each new attribute dictionary entry should, at a minimum, specify the name of the attribute and the range of acceptable values for that attribute. Other useful information to specify includes semantic or contextual definitions and the rationale for why the attribute exists. Regardless of the specific attributes in the dictionary, it is important that all providers of attributes (e.g. Data Requestor, Supplemental Attribute Authorities, etc.) understand the syntactic and semantic details of the attributes that they are expected to provide. Consumers of attributes (PDP and PEP) must also understand the attributes.

Step 2: The Policy Administrator creates, updates, or deletes a XACML policy.

This step is not illustrated.

Requirements:

A. ***Secure Communication Channel between PAP and Policy Repository***

B. ***Policy Authoring Tool*** – The use of a policy authoring tool with basic XACML editing and syntax checking capabilities can ease the burden on Policy Administrators authoring XACML policies. See Section 5 for references to several commercial and open source XACML authoring tools.

C. ***Identification of All Applicable Source Policies*** – In the case of creating or updating XACML policies, the Policy Administrator must identify all "Plain English" source policies that govern access to the resource, from federal laws to local rules of operation within the enterprise, to ensure that the XACML policy developed for the resource is in full compliance with all applicable rules, regulations, and laws.

D. ***Translation of All Applicable Source Policies into XACML*** – When creating or updating XACML policies, the Policy Administrator must create appropriate XACML statements that faithfully represent the applicable source policies. Some of the source policies may have already been translated into XACML statements; the Policy Administrator may be able to reuse these translations if they conform to the attribute dictionary defined in Step 1. In some cases, the Administrator may need to interpret source policy statements that are ambiguous, and translate them into XACML. It is important that the author(s) or policy maker(s) of the source policies verify that the final, translated XACML policy is a faithful representation of the source policies.

E. ***Discovery of Existing Policies*** – The PAP should allow the administrator to search for and retrieve existing policies. This feature supports the updating and deleting of policies.

Step 3: The Policy Administrator installs the XACML policy in the Policy Repository.

Figure 10 depicts the components that participate in this step.

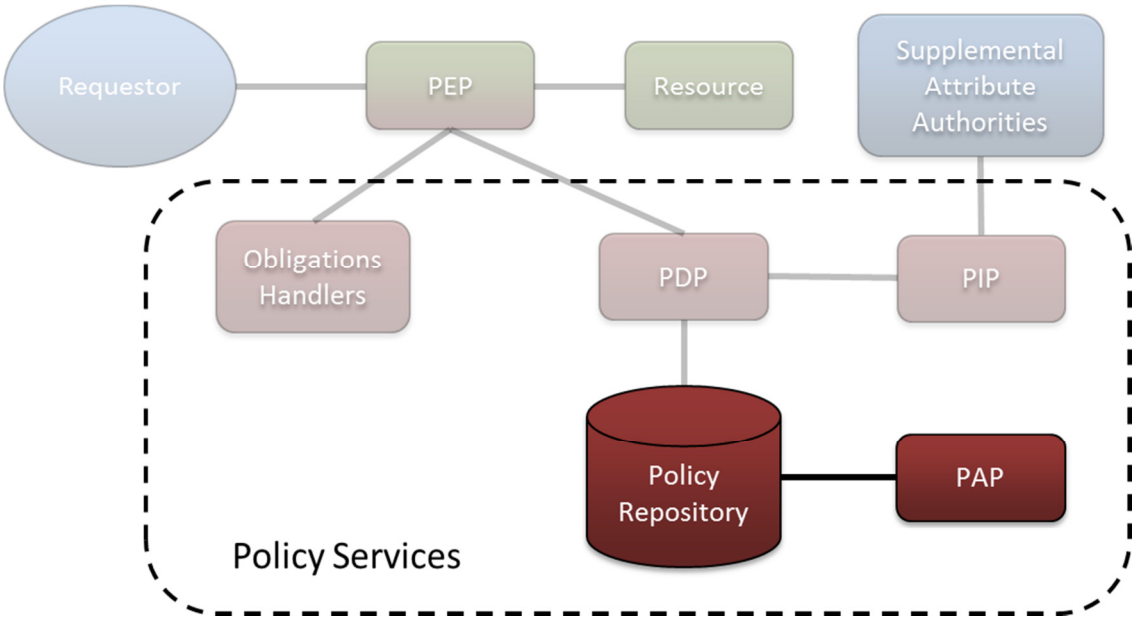

Figure 10: Components Involved in Step 3 (Policy Installation)

Requirements:

A. ***Secure Communication Channel between PAP and Policy Repository***

B. ***Submission of the XACML Policy to the Policy Repository*** – The Policy Administrator must submit the XACML policy to the Policy Repository, to expose the policy to the PDP.

C. ***Proper Configuration of Policy Services Components*** – The Policy Administrator must ensure that all of the Policy Services components are configured properly to support the newly installed XACML policy and the attribute dictionary that it uses. The following configuration steps may be required whenever the XACML policy changes.

    a. Configure the PEP to use attributes that conform to the attribute dictionary.
    b. Configure the PIP to properly retrieve all required supplemental attributes.

c. Configure the Obligation Handlers to properly handle all obligations.

3 Step-by-Step Sample Implementation Tutorial

This Section provides detailed, step-by-step instructions for setting up a sample application with XACML access control policies that are evaluated at runtime using a XACML engine. The content in this Section is organized into Groups of Lessons. Some Lessons include "Challenges"- tasks we ask the reader to complete independently for the purpose of confirming and enhancing their comprehension of the material. The reader will need an XML editor to complete these challenges. Solutions are provided for each Challenge. Accompanying downloads are provided to assist the implementer, including sample solutions for each step of the implementation process.

Notation

XML elements, for XACML and data files, are written as they appear in XML documents, and are indicated in boldface text. For example: **<Policy>**.

XML attributes, for XACML and data files, are written as they appear in XML documents, and are indicated in boldface text. For example: **PolicyId**.

Values of XACML and data elements appear in double quotes. For example: "Permit".

We introduce some terms to serve as labels for certain groups of policy elements; these terms are used to enable discussions about groups of elements as a whole. These terms appear in italics. For example: *class*.

We use labels to refer to files, directories, and data items that exist in the accompanying virtual machine. These labels are used in the style of Linux environment variables- they begin with a dollar sign ($) which is followed by the label in all caps. For example: the label $POLICY_GUIDE refers to the following path on the virtual machine, "/home/guide/policy-guide". The complete list of labels used in this Guide and their definitions are in Appendix B: Labels.

3.1 The Syntax and Evaluation Process of XACML

This Lesson Group covers the XACML syntax and the policy evaluation process. The "$POLICY_GUIDE/xacml_lessons/" directory contains files that accompany this Lesson Group. Files associated with a particular Lesson are located a directory that has the same name as the Lesson number. For example, the files for Lesson 3.1.1 are in the "$POLICY_GUIDE/xacml_lessons/3.1.1/" directory.

3.1.1 Policy Authoring and Evaluation Basics

Goals:

1. Understand the basic structure of a XACML policy.
2. Understand the basic structure of a XACML request.
3. Understand how to evaluate a policy against a XACML request with the SunXACML library.
4. Understand the basic structure of a XACML response.

Summary:

In this Lesson, you will inspect simple XACML policies and XACML requests to learn the basic syntax of XACML. You will then use the SunXACML library to evaluate a policy against a request and learn how to read the XACML result. Also, you will be challenged with making edits to a request to achieve certain results.

Steps:

3.1.1.1 Inspect Permit-Policy.xml

Line 1 contains the standard XML header tag. Line 2 has an XML comment that contains copyright information (XML comments have no effect on XACML files). The **<Policy>** opening tag is on Lines 3 – 5. A XACML policy has either a top-level **<Policy>** element or a top-level **<PolicySet>** element[27].

Line 3 contains the OASIS XML namespace for the XACML policy language. We strongly recommend always including this namespace. Not including the namespace may make it more difficult, or even impossible, for your policies to be processed by some automated tools[28].

Line 4 contains the policy Identifier (**PolicyId**).

Line 5 contains the rule-combining algorithm (**RuleCombiningAlgId**). Rule combining algorithms govern how multiple rules are aggregated within a single policy and are covered in Lesson 3.1.6. The rule combining algorithm used here is "deny-overrides". No rule combining algorithm will have any effect on this **<Policy>** since this **<Policy>** contains only one **<Rule>**.

Lines 7 – 10 contain the **<Description>** of the **<Policy>**. **<Policy>**, **<PolicySet>**, and **<Rule>** elements can contain **<Description>** elements. The **<Description>** is for informational purposes only and has no effect on policy semantics.

[27] The **<PolicySet>** element is covered in Lesson 3.1.7.
[28] There are automated tools available to assist with authoring, testing, and analyzing XACML Policies.

The **<Target>** of the **<Policy>** is on Lines 12 - 22. A **<Target>** is a collection of attribute *predicates* organized into c*lasses*. In general, a *predicate* is defined as a statement that can be shown to be true or false, and in XACML, *predicates* are statements on attributes.

There are four *classes* of attributes in XACML: **<Subjects>**, **<Resources>**, <**Actions>**, and **<Environments>**. A *class* can contain one or more *instances*. The **<Resources>** *class* can contain one or more **<Resource>** *instances*; and so on. An *instance* can contain one or more *match-predicates*. The *match-predicates* in **<Action>** *instances* are **<ActionMatch>** elements; the *match-predicates* in **<Environment>** *instances* are **<EnvironmentMatch>** elements; and so on. The **<Target>** of this **<Policy>** contains *match-predicates* for only the **<Subjects>** *class*.

Lines 13 – 21 contain the **<Subjects>** *class* which contains a single **<Subject>** *instance* (Lines 14 – 20). Lines 15 – 19 contain the single <**SubjectMatch>** *match-predicate* in the single **<Subject>** *instance*. A *match-predicate* consists of three components: a **MatchId**, an **<AttributeValue>**, and an *attribute-reference*. A **MatchId** is a reference to a XACML function that returns a Boolean value[29]. An **<AttributeValue>** contains a **DataType** and a literal value. *Attribute-references* refer to attributes in XACML Requests. An *attribute-reference* can be one of *attribute-designator* or **<AttributeSelector>**[30]. *Attribute-designators* in the **<Subjects>** *class* are **<SubjectAttributeDesignator>** elements; *attribute-designators* in the **<Resources>** *class* are **<ResourceAttributeDesignator>** elements; and so on.

In the **<SubjectMatch>** on Lines 15 – 19, the **MatchId** (Line 15) is specified to be "string-equal". The **<AttributeValue>** (Line 16) has a **DataType** of "string" and a value of "Top Secret". The **<SubjectAttributeDesignator>** refers to the "SecurityClearanceLevelCode" GFIPM attribute. This *predicate* can be read: "the GFIPM Security Clearance Level Code of the Subject equals 'Top Secret'." A request by a user that has a Top Secret clearance will cause this *predicate* to be true.

Let's look at this *predicate* in more detail. There are three parts: (1) "the GFIPM Security Clearance Level Code of the Subject"; (2) "equals"; and (3) "Top Secret". The first part is an attribute, the second part is an operation that will result in true or false (Boolean operation), and the third part is a literal value. In general, a XACML *predicate* is a Boolean operation on two *attribute-expressions*[31]. We define an *attribute-expression* as being a literal value, an attribute, or a manipulation[32] of an attribute.

---

[29] Only functions that take two primitive values (as opposed to collections of values, known as bags) as input are able to be used as a **MatchId**. A complete list of standard XACML functions that can be used as a **MatchId** is in Section 7.5 of the XACML 2.0 Specification.

[30] **<AttributeSelector>** elements are covered in Section 3.1.4.

[31] XACML *predicates* do not always have to be in this form, but the authors have never come across a *predicate* that could not be normalized into this form.

[32] Manipulations on attributes are covered in Lesson 3.1.5.

In XACML, functions are used to build *predicates*. There are functions for common data operators, such as "equals" and "add", and more. A function will be specific to a certain **DataType** (e.g., "string-equals" and "integer-equals"). There are functions to do operations on strings, numeric values, Boolean values, date-time values, and more. A complete list of standard XACML functions is in Appendix A.3 of the XACML 2.0 Specification.

The **<Target>** of the **<Policy>** will be applicable to requests for which the "SecurityClearanceLevelCode" Subject attribute has a value of "Top Secret". When the **<Target>** of a **<Policy>** is applicable to a Request, then the **<Rule>** elements of that **<Policy>** are evaluated against the request.

A XACML rule, represented by a **<Rule>** element, is the articulation of an authorization. A rule contains an **Effect**, a decision of "Permit" or "Deny", and collection of *match-predicates* in a **<Target>**[33]. The *match-predicates* represent the authorized privileges. The **Effect** determines whether the rule is a positive or negative authorization.

Every **<Policy>**, **<Rule>**, and **<PolicySet>** element is required to have exactly one **<Target>** element[34], however, the **<Target>** may be empty. An empty **<Target>** matches every request. Also, every **<Policy>** element must specify exactly one rule-combining algorithm.

Lines 24 – 32 contain the single <**Rule>** of the <**Policy>**. The **Effect** of this Rule is "Permit", and the Rule Identifier (**RuleId**) is "Rule-1" (Line 24). An **Effect** can either be "Permit" or "Deny". Lines 26 – 28 contain the **<Description>** of the <**Rule>**. The <**Rule>** has an empty **<Target>** (Line 30), which means that this **<Rule>** is applicable to all requests.

When a **<Rule>** is applicable to a request, then the **<Rule>** evaluates to its **Effect**. Therefore, this **<Policy>** will evaluate to "Permit" for requests from Subjects that have a GFIPM Security Clearance Level Code of "Top Secret", regardless of any Resource, Action, and Environment attributes that may exist in the requests.

A XACML policy can evaluate to one of four decisions:

- "Permit" – the requested action is to be allowed.
- "Deny" – the requested action is to be prohibited.
- "NotApplicable" – the policy doesn't apply to the request.
- "Indeterminate" – there was an error during the evaluation.

[33] A **<Rule>** can optionally contain a **<Condition>**. **<Condition>** elements are covered in Lesson 3.1.5.
[34] Every **<PolicySet>** element is required to have exactly one **<Target>** element as well.

3.1.1.2 Inspect Request-1.xml

A XACML request is the articulation of one or more Subjects (**<Subject>** elements) seeking to perform a single Action (**<Action>** element) on one or more Resources (**<Resource>** elements) in a single Environment (**<Environment>** element). Each **<Subject>**, **<Action>**, **<Resource>**, and **<Environment>** element contains a set of zero or more **<Attribute>** elements. Each **<Attribute>** contains an **AttributeId** (an attribute Identifier), a **DataType**, and one or more **<AttributeValue>** elements. Each **<AttributeValue>** contains a single, literal value that must match the **DataType**.

This request contains Subject (Lines 5 – 10), Resource (Lines 12 – 17), and Action (Lines 19 – 24) attributes. There are no Environment attributes as shown on Line 26. This request can be read: "A Subject with a GFIPM Security Clearance Level Code of 'Top Secret' is attempting 'write' access to 'Resource-1'."

The **AttributeId** of the single Resource **<Attribute>** is the standard XACML "resource-id" Identifier. Every request must contain at least one **<Attribute>** that has the standard XACML "resource-id" Identifier in at least one **<Resource>**.

3.1.1.3 Evaluate Permit-Policy against Request-1

First, let's manually determine what the result should be. The PDP will first determine the applicability of the policy's **<Target>** to the request. To do this, the PDP will evaluate the *match-predicates* of the **<Target>** using the attributes of the request.

The *match-predicate* in Permit-Policy contains an *attribute-designator*. An *attribute-designator* is a reference to a particular **<Attribute>** in a request. When evaluating an *attribute-designator* against a request, the PDP will attempt to locate the **<Attribute>** in the request that has the following properties[35]:

- The **<Attribute>** must be in the same *class* as the *attribute-designator*.
- The **<Attribute>** must have the same **AttributeId** and **DataType** as the *attribute-designator*.

If a matching **<Attribute>** exists in the request, then the PDP will retrieve the values of all the **<AttributeValue>** elements of the **<Attribute>** (recall that an **<Attribute>** can have multiple **<AttributeValue>** elements) as a **bag**[36] of values. The PDP then invokes the function specified by the **MatchId** of the *match-predicate* one time for each value of the **bag**. For each invocation, the PDP will pass in the literal value of the **<AttributeValue>** of the *match-predicate* as the first parameter, and a value of the **bag** as the second parameter. If at least one invocation returns

[35] An *attribute-designator* can also optionally specify an **Issuer**. If an **Issuer** is specified, then a request **<Attribute>** must have the same **Issuer** value in order to match the *attribute-designator*.
[36] A **bag** is a mathematical set in which a value can appear more than once.

true, then the *match-predicate* evaluates to true. If all invocations return false, then the *match-predicate* evaluates to false[37]. If no matching **<Attribute>** is found in the request, then the PDP will retrieve an empty **bag** and the *match-predicate* will evaluate to false[38].

The Subject attribute of Request-1 (Lines 5 – 10) will cause the *match-predicate* of Permit-Policy on Lines 15 – 19 to be true. The Resource attribute of Request-1 (Lines 12 - 17) will be ignored by Permit-Policy since the Policy is silent on Resource attributes. The Action attribute of Request-1 (Lines 19 – 24) will be ignored by Permit-Policy since the Policy is silent on Action attributes. All the predicates of the Target of Permit-Policy will match the request; therefore the single **<Rule>** of Permit-Policy should be evaluated. Since the **<Rule>** has an empty **<Target>**, it will evaluate to its **Effect** ("Permit"). Since this is the only **<Rule>** in the **<Policy>**, the **<Policy>** should evaluate to "Permit".

Now, execute SunXACML's SimplePDP[39] with Request-1.xml and Permit-Policy.xml, and output the result to Request-1_Permit-Policy_Response.xml and inspect the result.

Note that SimplePDP does not output the XML declaration tag or XML namespace information in XACML responses.

A XACML response is contained in a **<Response>** element (Lines 1 – 8). There is one **<Result>** element (Lines 2 – 7) that corresponds to the Resource Identifier for which access was requested ("Resource-1").

The **<Decision>** is on Line 3 and is "Permit".

Lines 4 – 6 contain the **<Status>** of the result and Line 5 contains the **<StatusCode>**. Returning a **<Status>** is an optional feature of XACML. If a PDP returns a **<Decision>** of "Permit" or "Deny", then the **<Status>** should have a value of "ok" as it does on Line 5. We will not further investigate the status feature in this Guide.

3.1.1.4 Inspect Deny-Policy.xml

This policy consists of a top-level **<Policy>** element. The **<Target>** of the **<Policy>** is very similar to the **<Target>** of Permit-Policy. The **<Target>** of this policy is applicable to Subjects that have a GFIPM Security Clearance Level Code of "Confidential".

---

[37] See the XACML Reference Tables in Appendix C for complete details.
[38] There is an optional **MustBePresent** property of *attribute-references* that changes this behavior. If the **MustBePresent** property is true and no matching **<Attribute>** is found, then the *match-predicate* will evaluate to "Indeterminate". See the XACML Reference Tables in Appendix C for complete details.
[39] Follow the instructions in Appendix A: Common Tasks (Executing SimplePDP).

The **<Policy>** contains a single **<Rule>**, "Rule-1", which has an **Effect** of "Deny". The **<Target>** of "Rule-1" is applicable to requests to perform the "write" Action. This **<Rule>** (and thus the **<Policy>**), when evaluated against requests that are not performing the "write" Action, will evaluate to "NotApplicable".

3.1.1.5 Evaluate Deny-Policy against Request-1

First, let's manually determine what the result should be. The lone **<SubjectMatch>** of the **<Target>** of Deny-Policy (Lines 15 – 19) should match the lone Subject **<Attribute>** of Request-1 (Lines 6 – 9). However, the **<SubjectMatch>** will evaluate to false because the value of the Subject **<Attribute>** of Request-1 ("Top Secret") does not equal the **<AttributeValue>** of the **<SubjectMatch>** of Deny-Policy ("Confidential"). Therefore, the **<Target>** of Deny-Policy will not match Request-1, "Rule-1" will not be evaluated (even though it would have matched Request-1), and Deny-Policy should evaluate to a decision of "NotApplicable".

Now, execute SimplePDP with Deny-Policy.xml and Request-1.xml, and output the results to Request-1_Deny-Policy_Response.xml and inspect the result. Confirm that the **<Decision>** of the **<Result>** for "Resource-1" states "NotApplicable".

3.1.1.6 Challenge: Create a request that will be applicable to Deny-Policy

Make a copy of the Request-1.xml file and name the copy "Request-2.xml". Open Request-2.xml. The value of the Subject **<AttributeValue>** on Line 8 should read "Top Secret". Change this value to a value that will make Request-2 cause Deny-Policy to evaluate to "Deny".

The solution to this Challenge is in Request-2-Solution.xml.

3.1.1.7 Evaluate Deny-Policy against Request-2

First, let's manually determine what the result should be. The Subject **<Attribute>** of Request-2 (Lines 6 – 9) should match the **<SubjectMatch>** of the **<Target>** of Deny-Policy (Lines 15 – 19). The Resource **<Attribute>** of Request-2 (Lines 13 - 16) should be ignored by the **<Target>** of Deny-Policy since the **<Target>** does not specify the **<Resources>** *class*. The Action **<Attribute>** of Request-2 (Lines 20 - 23) should be ignored by the **<Target>** of Deny-Policy since the **<Target>** does not specify the **<Actions>** *class*. Therefore, Rule-1 of Deny-Policy should be evaluated against Request-2.

The Subject **<Attribute>** of Request-2 (Lines 6 - 9) should be ignored by the **<Target>** of Rule-1 since the **<Target>** does not specify the **<Subjects>** *class*. The

Resource **<Attribute>** of Request-2 (Lines 13 - 16) should be ignored by the **<Target>** of Rule-1 since the **<Target>** does not specify the **<Resources>** *class*. The Action attribute of Request-2 (Lines 20 – 23) should match the **<ActionMatch>** of the **<Target>** of Rule-1 (Lines 37 - 41). Therefore, Rule-1 should evaluate to its **Effect** ("Deny"), and subsequently Deny-Policy should evaluate to "Deny".

Now, execute SimplePDP with Deny-Policy.xml and Request-2.xml, and output the results to Request-2_Deny-Policy_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "Deny".

3.1.1.8 Evaluate Permit-Policy against Request-2

First, let's manually determine what the result should be. The lone **<SubjectMatch>** of the **<Target>** of Permit-Policy (Lines 15 – 19) should match the lone Subject **<Attribute>** of Request-2 (Lines 6 – 9). However, the **<SubjectMatch>** will evaluate to false because the value of the Subject **<Attribute>** of Request-2 ("Confidential") does not equal the **<AttributeValue>** of the **<SubjectMatch>** of Permit-Policy ("Top Secret"). Therefore, the **<Target>** of Permit-Policy will not match Request-2, "Rule-1" will not be evaluated (even though it would have matched Request-2), and Permit-Policy should evaluate to a decision of "NotApplicable".

Now, execute SimplePDP with Permit-Policy.xml and Request-2.xml, and output the results to Request-2_Permit-Policy_Response.xml. Confirm that the **<Decision>** for "Resource-1" states "NotApplicable".

3.1.2 The "Attribute Value Spacing" Pitfall

Goals:

1. Understand what the Attribute Value Spacing Pitfall, and why it is problematic[40].

Summary:

In this Lesson, you will compare two policies that have a subtle difference in the value of an **<AttributeValue>** element. You will evaluate both policies against the same request and analyze the different results.

Steps:

3.1.2.1 Inspect Permit-Policy.xml and Request-1.xml

Confirm that this Permit-Policy and Request-1 are the same as the Permit-Policy and Request-1 from Lesson 3.1.1.

[40] You should be very diligent when authoring policies to avoid this problem. Also, it may be possible to construct an XSLT stylesheet to ensure that this condition never occurs.

3.1.2.2 Evaluate Permit-Policy against Request-1

Recall from Lesson 3.1.1 that the **<Decision>** should be "Permit". Confirm that this is the case.

3.1.2.3 Compare Permit-Policy with Permit-Policy-2

See if you notice the subtle difference. The closing tag of the **<AttributeValue>** element in Permit-Policy-2 on Line 16 does not come immediately after the value "Top Secret". The **MatchId** of the **<SubjectMatch>** (Line 15) is "string-equal"; during evaluation, this function will take into account the extra spaces after the value "Top Secret".

3.1.2.4 Evaluate Permit-Policy-2 against Request-1

Execute SimplePDP with Permit-Policy-2.xml and Request-1.xml, and output the results to Request-1_Permit-Policy-2_Result.xml. Inspect Request-1_Permit-Policy-2_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "NotApplicable". It is "NotApplicable" because the value "Top Secret" in the **<AttributeValue>** in Request-1 on Line 7 is not the same as "Top Secret" with extra spaces as stated in Permit-Policy-2.

3.1.3 Multiple Match-Predicates per Instance, Multiple Instances per Class

Goals:

1. Understand the policy evaluation semantics for multiple *match-predicates* in an *instance*.
2. Understand the policy evaluation semantics for multiple *instances* in a *class*.

Summary:

In this Lesson, you will analyze policies that have multiple *match-predicates* in an *instance* and multiple *instances* in a *class*. You will learn the policy evaluation semantics for both scenarios; multiple *match-predicates* in an *instance* are conjunctive while multiple *instances* in a *class* are disjunctive. You will be challenged to author requests that will achieve certain results.

Steps:

3.1.3.1 Inspect Multiple-Predicate-Policy.xml

Open Multiple-Predicate-Policy.xml. Multiple-Predicate-Policy contains a **<Target>** (Lines 12 - 27) with only the **<Subjects>** *class* specified (Lines 13 – 26). It contains a single **<Rule>**, "Rule-1" (Lines 29 – 37), that has an empty **<Target>** (Line 35) and an **Effect** of "Permit".

The **<Subjects>** *class* of the policy **<Target>** contains a single **<Subject>** *instance* (Lines 14 – 25). This *instance* contains two **<SubjectMatch>** *match-predicate* elements; the first is on Lines 15 – 19, and the second is on Lines 20 – 24. The first *match-predicate* can be read: "The GFIPM Security Clearance Level Code of the Subject is 'Top Secret'." The second *match-predicate* can be read: "The Subject is a Sworn Law Enforcement Officer."
Note that the second *match-predicate* uses a **MatchId** of "boolean-equal". This Function compares two Boolean values for equality. When using the SunXACML library, literal Boolean values (i.e., "true" and "false") must be in lower case.

All *match-predicates* need to evaluate to true for the parent *instance* to match a request (see the Table 17: Instance Evaluation Table for more details). In this policy, requests for which the Subject is a Sworn Law Enforcement Officer with a Top Secret Clearance will match the **<Subject>** *instance*. Since this is the only *instance* in the policy, and the policy has a single rule, then this policy should evaluate to "Permit" for Sworn Law Enforcement Officers who have a Top Secret Clearance performing any action to any resource in any environment.

3.1.3.2 Inspect Multiple-Instance-Policy.xml

Open Multiple-Instance-Policy.xml. Multiple-Instance-Policy contains a **<Target>** (Lines 12 - 29) with only the **<Subjects>** *class* specified (Lines 13 – 28). It contains a single **<Rule>**, "Rule-1" (Lines 31 – 39), that has an empty **<Target>** (Line 37) and an **Effect** of "Permit".

The **<Subjects>** *class* of the policy **<Target>** contains two **<Subject>** *instances*. The first is on Lines 14 – 20, and the second is on Lines 21 – 27. Each **<Subject>** *instance* contains a single **<SubjectMatch>** *match-predicate*.

The **<SubjectMatch>** of the first **<Subject>** (Lines 15 – 20) also exists in Multiple-Predicate-Policy. It can be read: "The GFIPM Security Clearance Level Code of the Subject is 'Top Secret'."

The **<SubjectMatch>** *match-predicate* of the second **<Subject>** *instance* (Lines 22 – 26) also exists in Multiple-Predicate-Policy. It can be read: "The Subject is a Sworn Law Enforcement Officer."

For a *class* to match a request, at least one of its *instances* must match the request (see Table 18 for more details). For this policy, the **<Subjects>** *class* will match requests for which the Subject either has a Top Secret Clearance, or is a Sworn Law

Enforcement Officer, or both. Since the **<Subjects>** *class* is the only *class* specified, and there is only one rule, this policy will evaluate to "Permit" for requests that its **<Subjects>** *class* matches.

3.1.3.3 Compare Multiple-Predicate-Policy to Multiple-Instance-Policy

These policies both include the same *match-predicates*. However, since Multiple-Predicate-Policy organizes the *match-predicates* within the same *instance*, and Multiple-Instance-Policy organizes the *match-predicates* in separate *instances*, the semantics of these two policies are different (as described in Steps 3.1.3.1 and 3.1.3.2). Multiple-Predicate-Policy is more restrictive since both *match-predicates* must evaluate to true for that policy to be applicable to a request. Also, Multiple-Instance-Policy will be applicable to every request to which Multiple-Predicate-Policy is applicable.

3.1.3.4 Challenge: Create Request-1

Create a new XML file called "Request-1.xml". In this file, author a request that will be applicable to Multiple-Predicate-Policy. Because of how the two policies are written, this request should also be applicable to Multiple-Instance-Policy. The request should include Subject attributes, and a "resource-id" Resource attribute. For the "resource-id" attribute, use a value of "Resource-1". You can leave the Action and Environment sections empty.

A solution to this Challenge is in Request-1-Solution.xml.

3.1.3.5 Evaluate Multiple-Predicate-Policy against your Request-1

Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.3.6 Evaluate Multiple-Instance-Policy against your Request-1

Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.3.7 Challenge: Create Request-2

Create a new XML file called "Request-2.xml". In this file, author a request that will not be applicable to Multiple-Predicate-Policy, but will be applicable to Multiple-Instance-Policy. The request should include Subject attributes, and a "resource-id" Resource attribute. For the "resource-id" attribute, use a value of "Resource-1". You can leave the Action and Environment sections empty.

A solution to this Challenge is in Request-2-Solution.xml.

3.1.3.8 Evaluate Multiple-Predicate-Policy against your Request-2

Confirm that the **<Decision>** for "Resource-1" is "NotApplicable".

3.1.3.9 Evaluate Multiple-Instance-Policy against your Request-2

Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.4 Referencing Resource Content

Goals:

1. Understand how to use the **<AttributeSelector>** element.
2. Understand how a Policy can use the content of resources in the evaluation process.

Summary:

This Lesson introduces the use of the **<AttributeSelector>** element. You will be asked to inspect and analyze policies using this element, and make comparisons with policies that only use *attribute-designators*. You will confirm the analysis by evaluating the policies against requests. Finally, you will be challenged to edit a policy and a request to achieve specific results.

Steps:

3.1.4.1 Inspect Permit-Policy.xml and Request-1.xml

Confirm that this Permit-Policy and Request-1 are the same as the Permit-Policy and Request-1 from Lesson 3.1.1.

3.1.4.2 Evaluate Permit-Policy against Request-1

Recall from Lesson 3.1.1 that the **<Decision>** should be "Permit". Confirm that this is the case.

3.1.4.3 Inspect Selector-Policy.xml

Selector-Policy is semantically the same as Permit-Policy, with two significant syntactical differences. Recall from Lesson 3.1.1 that a *match-predicate* consist of three parts:

- A **MatchId**
- An **<AttributeValue>**
- An *attribute-reference* which can be an *attribute-designator* or an **<AttributeSelector>**
    - The name of *attribute-designator* elements are dependent on which *class* the *attribute-designator* is in. **<Subjects>** contain **<SubjectAttributeDesignator>** elements and so on.

The **<SubjectMatch>** *match-predicate* of Selector-Policy uses an **<AttributeSelector>** *attribute-reference* (Lines 18 – 19), while the **<SubjectMatch>** of Permit-Policy uses an *attribute-designator* (**<SubjectAttributeDesignator>**) *attribute-reference* (Lines 17 – 18). The particular **<AttributeSelector>** in Selector-Policy causes the exact same semantic effect as the **<SubjectAttributeDesignator>** in Permit-Policy: it causes the PDP, when evaluating the policy against a request, to retrieve the values of all **<AttributeValue>** elements (as a **bag** of values) of the GFIPM Security Clearance Level Code Subject attribute of the request.

An **<AttributeSelector>** contains a **DataType** and a **RequestContextPath**. The value of a **RequestContextPath** must be an XPath expression into the request context[41]. The PDP will retrieve the set of nodes[42] referenced by the **RequestContextPath** as a **bag** of values. If no nodes are found, then the PDP returns an empty **bag**[43].

Beware that XACML defines one XML namespace for policies and a separate namespace for the XACML context. Since a **RequestContextPath** is an XPath expressions into the request context, any policy that uses an **<AttributeSelector>** must declare the XACML context namespace. This is done in Selector-Policy on Line 4; a prefix of "ctx" is used to represent the context namespace. On Line 19, the "ctx" prefix is used in the XPath expression.

3.1.4.4 Evaluate Selector-Policy against Request-1

Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.4.5 Inspect ArrestRecord.xsd

[41] The XACML "context" is the XML structures of requests and responses.
[42] A "node" is a term used in the context of XPath that means a part of an XML document.
[43] The optional **MustBePresent** property of *attribute-references* changes this behavior.

This file is in the "$POLICY_GUIDE/arrest_record_simple/" directory. It contains an XML Schema[44] for an **<ArrestRecord>** element. We will use this schema to represent a set of Arrest Records for which we want to protect access.

The schema defines an **<ArrestRecord>** element that contains seven sub-elements:

- **<Id>** - the identifier of the record.
- **<SubjectId>** - the identifier of the individual who was arrested.
- **<Jurisdiction>** - the jurisdiction in which the arrest occurred.
- **<Date>** - the date at which the arrest occurred.
- **<ArrestingOfficerId>** - the identifier of the arresting officer.
- **<ArrestingOfficerAgencyName>** - the name of the agency that employs the arresting officer.
- **<ArrestingOfficerEmailAddress>** - the email address of the arresting officer.

An arrest record articulates that an officer arrested some individual on a particular date within a particular jurisdiction. Valid values for jurisdiction are defined by the GFIPM Jurisdiction Code Set[45].

3.1.4.6 Inspect Record-1.xml

Confirm that this XML document conforms to ArrestRecord.xsd[46]. Record-1 states that Officer-1 arrested Subject-1 in Georgia on Valentine's Day 2012.

3.1.4.7 Inspect Content-Request-1.xml

This request shows an example of how XML content can be included in a request. The **<ResourceContent>** element (Lines 8 – 16) contains the content of the Record-1 Arrest Record (Lines 9 – 15). Notice the declaration of the Arrest Record namespace on Line 10 and the use of the "ar" prefix throughout the content of the Arrest Record.

Policies must use an **<AttributeSelector>** to retrieve values from a **<ResourceContent>** element in a request.

3.1.4.8 Inspect Content-Policy-1.xml

[44] The IEPD schema used here is technically not a genuine IEPD; however, the schema is NIEM IEPD-conformant and provides a close approximation of a genuine IEPD.

[45] The GFIPM Jurisdiction code set is available at http://gfipm.net/standards/metadata/2.0/codesets/GFIPMJurisdictionCode.html.

[46] This can be done by using an XML Schema validator to validate Record-1.xml against ArrestRecord.xsd

This policy will evaluate to "Permit" for requests that contain an Arrest Record with a Jurisdiction value of "GA".

The **<Target>** (Lines 16 – 27) contains a single **<ResourceMatch>** *match-predicate* (Lines 19 – 24) that uses an **<AttributeSelector>** (Lines 21 – 23). The **RequestContextPath** (Line 23) expression points to the value of the **<Jurisdiction>** element in an **<ArrestRecord>**.

Notice the use of the "ar" namespace prefix in the XPath expression and the declaration of the Arrest Record namespace on Line 5. When using the SunXACML library, XPath expressions in **RequestContextPath** XML-attributes must be XML namespace qualified.

This policy contains a single "Permit" **<Rule>** that has an empty **<Target>**.

3.1.4.9 Evaluate Content-Policy-1 against Content-Request-1

First, let's manually determine what the result should be. If the single **<ResourceMatch>** of the policy evaluates to true, then the policy should evaluate to "Permit", otherwise the policy should evaluate to "NotApplicable".

The **<ResourceMatch>** will be true for requests that include an **<ArrestRecord>** that has a **<Jurisdiction>** value of "GA". Content-Request-1 has such an **<ArrestRecord>**, therefore Content-Policy-1 should evaluate to "Permit".

Now, execute SimplePDP with Content-Request-1.xml and Content-Policy-1.xml, and output the results to Content-Request-1_Content-Policy-1_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.4.10 Challenge: Add match-predicates to Content-Policy-1

Create a copy of Content-Policy-1 and call the new file: "Content-Policy-2.xml". Open Content-Policy-2.xml. On Line 5, change the end of the **PolicyId** to read "Content-Policy-2".

In the existing **<Resource>** *instance* (Lines 18 – 25), create a new **<ResourceMatch>** that articulates this predicate: "the Subject Id of the Arrest Record equals 'Subject-1'."

Notice that the subject of the Arrest Record is handled in the **<Resources>** *class* because it is a part of the resource content and is not the subject of the request.

Create the **<Subjects>** *class* (currently non-existent) in the **<Target>** of the policy, and include one **<Subject>** *instance*. In this **<Subject>**, create a **<SubjectMatch>**

that articulates this predicate: "The request Subject is a Sworn Law Enforcement Officer." Use the GFIPM Sworn Law Enforcement Officer Indicator attribute identifier[47].

A solution to this Challenge is in Content-Policy-2-Solution.xml.

3.1.4.11 Evaluate Content-Policy-2 against Content-Request-1

Evaluate your Content-Policy-2 against Content-Request-1. Confirm that the **<Decision>** for "Resource-1" is "NotApplicable". This should be because the **<SubjectMatch>** you created in Content-Policy-2 should evaluate to false; Content-Request-1 is silent on Subject attributes. Recall that all four *classes* must match a request in order for the parent **<Target>** to match the request.

3.1.4.12 Challenge: Create a request that is applicable to Content-Policy-2

Create a copy of Content-Request-1 and call the new file: "Content-Request-2". Edit Content-Request-2 to make it applicable to Content-Policy-2.

A solution to this Challenge is in Content-Request-2-Solution.xml.

3.1.4.13 Evaluate Content-Policy-2 against Content-Request-2

Evaluate your Content-Policy-2 against Content-Request-2. Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.5 Rule Conditions

Goals:

1. Understand the need for rule conditions.
2. Understand how rule conditions affect rule evaluation.
3. Understand how to properly author rule conditions.

Summary:

This Lesson introduces rule conditions. Through inspecting, analyzing, and evaluating sample policies, you are led to understand the need for and semantics of conditions. You will be challenged to author a rule that contains a condition.

[47] Details on the GFIPM Sworn Law Enforcement Officer Indicator attribute are at http://gfipm.net/standards/metadata/2.0/user/SwornLawEnforcementOfficerIndicator.html.

Steps:

3.1.5.1 Inspect Condition-Policy-1.xml

This policy has an empty **<Target>** and a single **<Rule>**, "Rule-1", whose **Effect** is "Deny". Rule-1 has an empty **<Target>** and a **<Condition>** (Lines 25 – 33).

A **<Condition>** is a type of *predicate* that is more flexible than a *match-predicate*. *Match-predicates* have three main limitations:

1. They use a "hard-coded", literal value (in an **<AttributeValue>**).
2. They can only involve a single attribute.
3. Only a subset of XACML functions can be used.

Examples of predicates that *match-predicates* cannot articulate are:

- The Jurisdiction of the Resource content does not equal "GA".
- The Jurisdiction of the Resource content is one of "MD" or "VA".[48]
- The Subject's Security Clearance Level Code equals the Resource's Security Clearance Level code.

A **<Condition>** contains a single top-level **<Apply>** element. An **<Apply>** element, via the **FunctionId** property (see Line 26), is the specification of an invocation of a XACML function. Unlike with *match-predicates*, this function can be any function that returns a Boolean value; there is no restriction on the number or types of input parameters[49]. Therefore, parameters to the function may include:

- Literal values, via an **<AttributeValue>** element
- **<AttributeSelector>** elements
- *Attribute-designator* elements
- Other function invocations, via other **<Apply>** elements
- "Function pointers", via **<Function>** elements[50]

If a **<Condition>** exists in a **<Rule>**, then that **<Condition>** must evaluate to true for the **<Rule>** to be applicable to a request (see Table 20 for more details).

The **<Condition>** of Rule-1 in Condition-Policy-1 uses the XACML "not" function as its top-level function call. This function takes in a single Boolean parameter and returns the opposite of that parameter (i.e., true becomes false, and false becomes true). The parameter to the "not" function is another **<Apply>** element (Line 27) specifying a call to the "string-is-in" function.

The "string-is-in" function takes in two parameters. The first must be a primitive string value. Line 28 specifies an **<AttributeValue>** with a literal value of "GA" as

[48] This predicate could be expressed using multiple *match-predicates*, but not a single one.
[49] Recall that *match-predicates* can only use functions that return a Boolean value and takes in two primitive values as parameters.
[50] The difference between the **<Apply>** element and the **<Function>** element should become apparent through the examples provided in this Lesson.

the first parameter. The second parameter to "string-is-in" must be a **bag** of string values. Lines 29 – 30 specify an **<AttributeSelector>**, which results in a **bag** of values, as the second parameter. The "string-is-in" function returns true if the first parameter is equal to at least one of the values of the second parameter.

This **<Condition>** *predicate* can be read: "the Arrest Record does not contain a Jurisdiction value of 'GA'." When this rule condition is true, the rule will evaluate to its Effect: "Deny".

3.1.5.2 Inspect Request-1.xml

This is the same as Content-Request-1 of Lesson 3.1.4, except that the Jurisdiction of the Arrest Record is "FL" instead of "GA".

3.1.5.3 Evaluate Condition-Policy-1 against Request-1

The **<Decision>** for "Resource-1" should be "Deny" since the Jurisdiction of the Arrest Record in the Resource content is not "GA" (it is "FL"). Confirm that this is the case.

3.1.5.4 Inspect Condition-Policy-2.xml

This policy, like Condition-Policy-1, has an empty **<Target>** and a single **<Rule>**, "Rule-1", with an empty **<Target>** and a **<Condition>**.

The **<Condition>** (Lines 25 – 33) uses the "any-of-any" XACML function at its top-level. This function takes in three parameters. The first parameter must be a **<Function>** element specifying a function that returns a Boolean and takes in two primitive values. The second and third parameters must be **bags** of values, and the **DataType** values of those **bags** must match the expected **DataType** values of the **<Function>** element. The "any-of-any" function applies the function specified by the first parameter between each value of the second parameter and each value of the third parameter. The "any-of-any" function returns true if at least one of the **<Function>** invocations returns true. Otherwise, the "any-of-any" function returns false.

The **<Condition>** of Rule-1 will evaluate to true if the Employment Jurisdiction of the request Subject matches the Jurisdiction of the Arrest Record.

3.1.5.5 Inspect Request-2.xml

The structure of Request-2 is similar to Request-1 of this Lesson, except that Request-2 contains a GFIPM Employment Jurisdiction Subject attribute. Also note that the values of the requested Resource Arrest Record have been changed.

3.1.5.6 Evaluate Condition-Policy-2 against Request-2

The **<Decision>** for "Resource-2" should be "Permit" since the GFIPM Employment Jurisdiction of the Subject is the same as the Jurisdiction of the Arrest Record ("FL"). Confirm that this is the case.

3.1.5.7 Evaluate Request-3

Request-3 is similar to Request-2. The only difference is that Request-3 has a different Subject attribute. Request-3 specifies that the Subject's GFIPM Federation Id is "Officer-3".

3.1.5.8 Challenge: Create a new policy

Create a file called: "Condition-Policy-3.xml". In this file, author a policy that will permit a Subject to read Arrest Records for which the Subject was the arresting officer. In other words, the GFIPM Federation Id of the request Subject must equal the value of the **<ArrestingOfficerId>** element in the Arrest Record, and the request Subject must be performing the "read" Action.

A solution to this Challenge is in Condition-Policy-3-Solution.xml.

3.1.5.9 Evaluate Condition-Policy-3 against Request-3

Confirm that the **<Decision>** for "Resource-3" is "Permit".

3.1.5.10 Inspect Condition-Policy-4.xml

This policy has an empty **<Target>** and a single **<Rule>** with an empty **<Target>** and one **<Condition>**. The **<Condition>** expresses the *predicate*: "the current date is less than the date of the accessed record plus sixty months (five years)." Note that a simpler way to word this *predicate* is: "the accessed record is less than sixty months (five years) old." However, this simpler wording is not in a form that's directly implementable in XACML.

The *attribute-expression* "the date on the accessed record plus sixty months" expresses a manipulation on the "record date" attribute; that attribute is manipulated by adding 60 months.

The top-level Function of the **<Condition>** is "date-less-than", which takes in two parameters of type "date" and returns true if the first parameter is an earlier date than the second parameter. If the first parameter equals the second parameter or if the first parameter is a later date than the second parameter, then "date-less-than" returns false.

The first parameter to "date-less-than" (Lines 28 – 32) is effectively the date at which the request was constructed by the PEP. The XACML "current-date" Environment attribute represents this date[51]. An *attribute-designator* is used (see Lines 29 – 31) which provides a **bag** of values, but the "date-less-than" function requires a single primitive value, not a **bag**. Therefore, the "date-one-and-only" function (Line 28) is used. This function returns the single date primitive value from a **bag** of date values or throws an error if there is more than one value in the **bag**.

The second parameter to "date-less-than" (Lines 33 – 41) expresses the "date on the accessed record plus sixty months" *attribute-expression*. The "date-add-yearMonthDuration" function (Line 34) returns the result of adding a duration of years and months (in this case 60 months; see Lines 39 - 40) to a date value (in this case the date on the accessed record; see Lines 35 – 38).

3.1.5.11 Evaluate Condition-Policy-4 against Request-4 and Request-5

Request-4 is similar to Request-1. The main difference is that Request-4 seeks access to an Arrest Record from Valentine's Day 2007 and includes a value for the XACML current-date Environment attribute. Recall that this attribute represents the date at which the XACML request was created and is used in the **<Condition>** in Condition-Policy-4. Request-4 expresses a XACML request that was constructed by the PEP on Valentine's Day 2012.

The value of the current-date Environment attribute is exactly five years later than the date of the record. Therefore, Request-4 should cause Condition-Policy-4 to evaluate to "NotApplicable". Evaluate Condition-Policy-4 against Request-4 and confirm this result.

Now, inspect Request-5.xml. Request-5 is the same as Request-4 except that the current-date attribute of Request-5 has the value of "2012-02-13" which is just one day less than five years later than the date of the record. Request-5 should therefore cause Condition-Policy-4 to evaluate to "Permit". Evaluate Condition-Policy-4 against the Request-5 and confirm the result.

[51] XACML request construction is covered in Lesson 3.3.3.2.

3.1.6 Aggregating Multiple Rules

Goals:

1. Understand how to aggregate multiple rules into a single policy.
2. Understand the potential for conflicts.
3. Understand how rule combining algorithms are used.

Summary:

In this Lesson, you will inspect, analyze, manipulate, and evaluate policies that have multiple rules. You will learn about conflicts among rules and how rule-combining algorithms resolve those conflicts. Also, you will be challenged with authoring a policy that expresses a source policy with multiple rules.

Steps:

3.1.6.1 Inspect Policy-1.xml

This policy has an empty **<Target>** and contains two rules. The first rule (Lines 16 – 35), "Rule-1", has an **Effect** of "Permit". The second rule (Lines 37 – 65), "Rule-2", has an **Effect** of "Deny". Rule-1 can be read: "Officers can perform any Action in any Environment on Arrest Records for which they are the arresting officer." Rule-2 can be read: "Arrest Records in the 'MD' Jurisdiction cannot be deleted by any Subject in any Environment."

The two rules have conflicting **Effect** values. During evaluation against a request, if both rules are applicable to the request, the policy will evaluate to "Deny" due to its rule-combining algorithm.

The rule-combining algorithm of the policy is "deny-overrides" (see Line 6). With "deny-overrides", if any rule evaluates to "Deny", then the policy will evaluate to "Deny". If no rule evaluates to "Deny", but at least one rule evaluates to "Permit", then the policy will evaluate to "Permit". Otherwise, the policy will evaluate to "NotApplicable".

Along with "deny-overrides", main rule-combining algorithms available in XACML are "permit-overrides" and "first-applicable"[52]. The "permit-overrides" algorithm can be considered the inverse of "deny-overrides": "Permit" decisions take precedence over "Deny" decisions. With the "first-applicable" algorithm, the rules are evaluated in the order as they appear in the policy; the policy evaluates to the

[52] The complete list and semantic definitions of all standard rule combining algorithms is in Appendix C of the XACML 2.0 Specification.

Effect of the first rule that is applicable to the request, or "NotApplicable" if no rules are applicable.[53]

3.1.6.2 Inspect Request-1.xml

Request-1 is the articulation of a request by Officer-1 to delete Resource-1 which is an Arrest Record. Officer-1 is the arresting officer and the arrest Jurisdiction is "VA".

3.1.6.3 Evaluate Policy-1 against Request-1

First, let's manually determine what the result should be. Since the policy uses the "deny-overrides" combining algorithm, we should check Rule-2 (the "Deny" rule) first. Rule-2 is not applicable to the request since the Jurisdiction in the request is "VA" and not "MD".

Now, let's consider Rule-1. Rule-1 is applicable to the request since Officer-1 is attempting access on a record of which Officer-1 is the arresting officer. Therefore, the policy should evaluate to "Permit".

Confirm that the **<Decision>** of Resource-1 is "Permit".

3.1.6.4 Inspect Request-2.xml

Request-2 is the articulation of a request by Officer-2 to delete Resource-2, which is an Arrest Record. Officer-2 is the arresting officer and the Jurisdiction is "MD".

3.1.6.5 Evaluate Policy-1 against Request-2

First, let's manually determine what the result should be. We'll check Rule-2 first. Rule-2 should be applicable to the request since the Jurisdiction in the request is "MD". Since a "Deny" rule is applicable, and since the rule-combining algorithm is "deny-overrides", there is no need to check Rule-1. The policy should evaluate to "Deny".

Confirm that the **<Decision>** of Resource-2 is "Deny".

3.1.6.6 Inspect Request-3.xml

[53] These are simplified descriptions of the semantics of "deny-overrides", "permit-overrides", and "first-applicable". These algorithms also handle cases where a rule evaluates to "Indeterminate".

Request-3 is the articulation of a request by Officer-3 to read Resource-3, which is an Arrest Record. Officer-4 is the arresting officer and the Jurisdiction is "MD".

3.1.6.7 Evaluate Policy-1 against Request-3

First, let's manually determine what the result should be. We'll check Rule-2 first. Rule-2 should not be applicable to Request-3 since Rule-2 applies to the delete Action and Request-3 seeks a read Action.

Now, let's consider Rule-1. Rule-1 should not be applicable to the request since the request Subject, Officer-3, does not match the arrest record's OfficerID, Officer-4. Therefore, the policy should evaluate to "NotApplicable".

Confirm that the **<Decision>** for Resource-3 is "NotApplicable".

3.1.6.8 Challenge: Create a new policy with multiple rules

The **<Description>** of Policy-1 (Lines 8 – 12) states: "Officers can perform any Action on Arrest Records for which they are the arresting officer. However, under no circumstances can records in the 'MD' Jurisdiction be deleted." Your Challenge is to create a new policy with a slightly different articulation: "Officers can perform any Action on Arrest Records for which they are the arresting officer. However, under no circumstances can records in the 'MD' Jurisdiction be deleted, except by holders of a Top Secret Clearance. Holders of a Top Secret Clearance can perform any Action on any Record in any Environment."

Save your new policy in a file called "Policy-2.xml". There are several possible solutions to this Challenge. One solution is provided in Policy-2-Solution.xml.

3.1.6.9 Inspect Policy-2-Solution.xml

Let's compare Policy-2-Solution to Policy-1. The rule-combining algorithm was changed to "first-applicable". A new Rule-1 provides total access to request Subjects with a Top Secret Security Clearance Level Code. Rule-2 stayed the same. Rule-1 from Policy-1 became Rule-3 in Policy-2-Solution.

The Description of Rule-1 of Policy-2-Solution (Lines 20 – 23) states: "Holders of a Top Secret Clearance can perform any Action on any Record in any Environment." When evaluating this policy against a request, the PDP will first evaluate Rule-1. If Rule-1 applies to a request, then the "first-applicable" rule-combining algorithm tells the PDP to proceed no further and to apply the Effect of Rule-1: "Permit". If Rule-1 is not applicable to the request, then the PDP will evaluate Rule-2. If Rule-2 is

not applicable to the request, then the PDP will evaluate Rule-3. If Rule-3 is not applicable, then the policy will evaluate to "NotApplicable".

3.1.6.10 Inspect Request-4.xml

Request-4 can be articulated as follows: "Officer-4, who has a Top Secret Clearance, is attempting to delete Resource-4, which is an Arrest Record. Officer-4 is the arresting officer and the Jurisdiction is 'MD'."

3.1.6.11 Evaluate Policy-2 against Request-4

Use Request-4 to test your Policy-2 (and Policy-2-Solution). Confirm that the **<Decision>** for Resource-4 evaluates to "Permit", since the request matches Rule-1.

3.1.6.12 Inspect Request-5.xml

Request-5 can be articulated as follows: "Officer-5, who has a Secret Clearance, is attempting to delete Resource-5, which is an Arrest Record. Officer-5 is the arresting officer and the Jurisdiction is 'MD'."

3.1.6.13 Evaluate Policy-2 against Request-5

Use Request-5 to test your Policy-2 (and Policy-2-Solution). Rule-1 should not be applicable to the request since Officer-5 does not have a Top Secret Clearance. Rule-2 should be applicable since the request is an attempt to delete an Arrest Record in the "MD" Jurisdiction. Therefore, Policy-2 should evaluate to "Deny".

Confirm that the **<Decision>** for Resource-5 is "Deny".

3.1.6.14 Inspect Request-6.xml

Request-6 can be articulated as follows: "Officer-6, who has a Secret Clearance, is attempting to delete Resource-6, which is an Arrest Record. Officer-6 is the arresting officer and the Jurisdiction is 'VA'."

3.1.6.15 Evaluate Policy-2 against Request 6

Use Request-6 to test your Policy-2 (and Policy-2-Solution). Rule-1 should not be applicable to the request since Officer-6 does not have a Top Secret Clearance. Rule-2 should not be applicable since the Jurisdiction of the record is not "MD". Rule-3

should be applicable since Officer-6 is both the Subject of the request and the arresting officer on the record. Therefore, Policy-2 should evaluate to "Permit".

Confirm that the **<Decision>** for Resource-6 is "Permit".

3.1.7 Aggregating Multiple Policies

Goals:

1. Understand how to aggregate multiple **<Policy>** elements in a **<PolicySet>** element.

Summary:

In this Lesson, you will inspect, analyze, and evaluate a policy consisting of a top-level **<PolicySet>** element and multiple **<Policy>** sub-elements. We provide the context of a local implementing agency needing to aggregate policies from multiple levels of authority, illustrating how policy-combining algorithms resolve conflicts among policies in a policy set.

Steps:

3.1.7.1 Inspect PolicySet-1.xml

PolicySet-1 is a **<PolicySet>**. It has a **PolicySetId** identifier (Line 5), and a **PolicyCombiningAlgorithm** of "permit-overrides" (Line 6). Policy-combining algorithms work in a similar manner to rule-combining algorithms. PolicySet-1 specifies a **<Target>** (Lines 16 – 26) and two **<Policy>** elements (the first on Lines 28 – 103 and the second on Lines 105 – 160). The first **<Policy>** represents a federation-level policy (of the "ExampleFederation") and the second represents a policy that's local to the agency that is implementing this **<PolicySet>** ("Agency-A").

This **<PolicySet>** is concerned with access to the criminal history records of Agency-A. Arrest Records constitute the entirety of criminal history data of Agency-A. Accordingly, the **<Target>** of the **<PolicySet>** specifies that the GFIPM Criminal History Data Indicator of the Resource must be true.

The first **<Policy>**, Federation-Policy-1, is the federation-level policy. It can be articulated as: "A federated user can read criminal history data (Arrest Records) if that user meets the following criteria: they are a sworn law enforcement officer, they possess the criminal history data agency home search privilege, and they have legal jurisdiction in the jurisdiction of the record."[54] The Subject *match-predicate* on

[54] Note that Federation-Policy-1 duplicates the **<ResourceMatch>** that is in the **<Target>** of the **<PolicySet>** because Federation-Policy-1 needs to be a complete policy in and of itself.

Lines 53 – 57 uses the "string-regexp-match" XACML function to determine if the Subject is a member of ExampleFederation by checking the GFIPM Federation Id attribute[55].

The second **<Policy>**, Local-Policy-1, is the local-level policy. It can be articulated as: "All sworn law enforcement officers of Agency-A who are authorized to search criminal history data are allowed to read any criminal history record."

PolicySet-1 uses the "permit-overrides" policy combining algorithm, therefore "Permit" decisions take precedence over "Deny" decisions. However, since PolicySet-1 does not contain any "Deny" rules, it will never evaluate to "Deny". It can only evaluate to "Permit" or "NotApplicable"[56].

3.1.7.2 Evaluate PolicySet-1 against Request-1

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will not be applicable to the request since the jurisdiction of the request Subject ("VA") does not match the jurisdiction of the record ("GA"). Therefore, Local-Policy-1 will be evaluated.

Local-Policy-1 will be applicable to the request since the Subject is a member of Agency-A (see Lines 6 – 8 of Request-1), is a sworn law enforcement officer, and is authorized to search criminal history data records (see Lines 24 – 27 of Request-1). Therefore, PolicySet-1 should evaluate to "Permit".

Confirm that the **<Decision>** for Resource-1 is "Permit".

3.1.7.3 Evaluate PolicySet-1 against Request-2

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will not be applicable to the request since the Subject does not have the criminal history data home agency search privilege (see Lines 20 – 23 of Request-2). Therefore, Local-Policy-1 will be evaluated.

Local-Policy-1 will not be applicable to the request since the Subject is not a member of Agency-A (see Lines 6 – 8 of Request-2). Therefore, PolicySet-1 should evaluate to "NotApplicable".

---

[55] See http://gfipm.net/standards/metadata/2.0/user/FederationId.html.

[56] Theoretically, PolicySet-1 can also evaluate to "Indeterminate", however, we have designed the policy and requests to avoid this result.

Confirm that the **<Decision>** for Resource-2 is "NotApplicable".

3.1.7.4 Evaluate PolicySet-1 against Request-3

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will be applicable to the request (you should be able to determine why). Therefore Federation-Policy-1 should evaluate to "Permit". Given the policy-combining algorithm "permit-overrides", there will be no need to evaluate Local-Policy-1, and PolicySet-1 should evaluate to "Permit".

Confirm that the **<Decision>** for Resource-3 is "Permit".

3.1.8 Obligations

Goals:

1. Understand how obligations are expressed in XACML.
2. Understand that obligation semantics are outside of the scope of XACML.

Summary:

This Lesson introduces obligations. You will analyze and evaluate a policy containing multiple obligations. There is a discussion on the design and handling of obligations.

Steps:

3.1.8.1 Inspect Policy-1.xml

Policy-1 is based on the Policy-1 from Lesson 3.1.6, with the addition of obligations using the **<Obligations>** element (Lines 67 – 91).

In the abstract sense, an obligation is an action that must be performed in conjunction with policy enforcement. This policy contains three obligations: the first is on Line 69, the second is on Line 71, and the third is on Lines 73 – 89.

An obligation in XACML (an **<Obligation>** element) has a **FulfillOn** property, an **ObligationId** property, and a set of zero or more **<AttributeAssignment>** elements. The **FulfillOn** property specifies the decision on which the obligation must be fulfilled; the value of this property can be "Permit" or "Deny". The **ObligationId** is the identifier of the obligation. An **<AttributeAssignment>** is an

argument[57] of the obligation. An **<AttributeAssignment>** contains a **DataType**, an identifier as an **AttributeId**, and a literal value.

The first obligation in Policy-1, LogValidAccess, is to be fulfilled on "Permit"; the PDP will include this obligation in the result when the decision is "Permit". The LogInvalidAccess obligation is to be fulfilled on "Deny"; the PDP will include this obligation in the result when the decision is "Deny". These obligations instruct the PEP to write data about the access to an audit log. The PEP must recognize and know how to handle these obligations. If a PEP does not understand or cannot fulfill an obligation, then the PEP must not allow access. For these example obligations in particular, we assume that the PEP (or its Obligation Handler components) will know how to retrieve the appropriate data to write to the log.

The third obligation, NotifyDataOwner, instructs the PEP to send a notification to the owner of the accessed record. In our scenario, the owner of an Arrest Record is the arresting officer. This obligation has three **<AttributeAssignment>** elements. The first is DataOwnerId and the value is actually an **<AttributeSelector>** containing an XPath expression selecting the value of the **<OfficerId>** element of the Arrest Record being accessed. Notice that the angled brackets are URL encoded (i.e., "<" becomes "<" and ">" becomes ">"); the PDP will decode these in the result. We assume that the PEP/Obligation Handler will process this **<AttributeSelector>** to retrieve the value for the DataOwnerId argument. We also assume that the PEP/Obligation Handler will be able to retrieve the appropriate address for the arresting officer.

The second argument is DataRequestorId and the value is the URL encoded **<SubjectAttributeDesignator>** that will retrieve the appropriate value.

The third argument is Message; this is the actual text that should be sent to the arresting officer. We assume that the PEP will replace "[DataRequestorId]" with the result of processing the second argument.

How obligations are designed will affect how the PEP (or its Obligation Handler components) will be designed. Design options include identifier naming conventions, whether to include arguments, and which arguments to include. We developed a particular design style for this tutorial, but there are currently no standard obligation design patterns available. As stated in Section 2.3.1 (Step 6, Requirement C), the Global Federated Identity and Technical Privacy Task Team is currently developing a standardized syntax and processing model for various types of policy obligations.

Since no XACML obligations are returned on the "NotApplicable" decision, care must be taken in designing policies to avoid this decision where appropriate so that all necessary obligations are properly returned to the PEP.

[57] An "argument" is data that is needed for the proper processing of the obligation.

3.1.8.2 Evaluate Policy-1 against Request-1

This is the same Request-1 from Lesson 3.1.6; therefore we know that the decision will be "Permit". Use SimplePDP to evaluate Policy-1 against Request-1, output the result to "Request-1_Policy-1_Response.xml", and open the result. Confirm that the **<Decision>** for Resource-1 is "Permit".

Notice the **<Obligations>** element on Lines 7 – 22. This element contains the two obligations that were specified to be fulfilled on "Permit". The PDP simply copies the appropriate obligations into the result (and decodes any URL-encoded values).

3.1.8.3 Evaluate Policy-1 against Request-2

This is the same Request-2 from Lesson 3.1.6; therefore we know that the decision will be "Deny". Use SimplePDP to evaluate Policy-1 against Request-2, output the result to "Request-2_Policy-1_Result.xml", and open the result. Confirm that the **<Decision>** for Resource-2 is "Deny".

This result includes the obligation that was specified to be fulfilled on "Deny" (see Lines 7 – 10).

3.1.8.4 Evaluate Policy-1 against Request-3

This is the same Request-3 from Lesson 3.1.6; therefore we know that the decision will be "NotApplicable". Use SimplePDP to evaluate Policy-1 against Request-3, output the result to "Request-3_Policy-1_Result.xml", and open the result. Confirm that the **<Decision>** for Resource-3 is "NotApplicable". Since the decision is "NotApplicable", no obligations were returned in the result.

3.2 The Sample Implementation Policy and Data Resources

This Lesson Group will introduce the set of source policies that will be used for the sample implementation. The policies will be based on protecting access to a set of Arrest Records. We introduce a NIEM IEPD-compliant set of XML schemas for representing Arrest Records. The Lessons in this Group will systematically build the XACML policy that will be used in the sample implementation. The resulting XACML policy will make use of all the XACML features discussed in Lesson Group 3.1 and will be similar to the policies discussed in Lessons 3.1.7 and 3.1.8.

3.2.1 The NIEM IEPD-Compliant Data Schemas

Goals:

1. Understand the structure defined by the NIEM IEPD-compliant Arrest Record data schemas.

Summary:

The Arrest Record schemas are in the "$POLICY_GUIDE/arrest_record_iepd/" directory. The main schema is in the "exchangeSchema.xsd" file. This file references the schema in the "extensionSchema.xsd" file and several schema files in the "niem" directory. The schemas are referenced on Lines 18 – 27 in exchangeSchema.xsd.

The file "exchangeSchema.xml" contains a sample conformant XML file that contains all the data elements that will be used in the Sample Application.

There are no Steps in this Lesson.

3.2.2 The Source Policies and Sample Implementation Context

Goals:

1. Understand the context of the sample implementation.
2. Review the source policy directives that will be translated into XACML.

Summary:

In this Lesson, we provide the context and assumptions for the sample implementation; these set the stage for the remaining Lessons in this Group. We then introduce the source policy directives and explain how they are organized.

Steps:

3.2.2.1 Review the Context and Assumptions

We will make the following assumptions when developing the XACML policy.

1. Agency-A and Agency-B are members of the GFIPM Reference Federation[58] (the federation may have other member agencies).
2. The source policies will be implemented by Agency-A.
3. The policies are to support a web service, operated by Agency-A, that accesses criminal history data.
4. All criminal history records take the form of an Arrest Record.
5. The web service provides a service interface for reading records.

[58] The GFIPM Reference Federation contains a set of systems and services used for testing compliance and interoperability with GFIPM.

6. Agency-A has a source policy for controlling access to its own records by its own employees.
7. All members of the GFIPM Reference Federation have agreed, via a legally binding contract, to share criminal history data within the federation under the restrictions of a federation source policy.
8. Agency-A has authored directives that are supplemental to the federation source policy.

3.2.2.2 Review the Source Policies

The source policies are in the "$POLICY_GUIDE/source-policies.txt" file. Inspect this file with a text editor. There are three sets of directives - the local organizational source directive, the federation-level source directive, and the set of directives that supplement the federation directive. The directives for the local organizational policy have labels with a prefix of "OD". The prefix "FD" is used in the federation-level policy, and the prefix "SD" is used by the supplemental directives.

Each directive either expresses an authorization or an obligation. The authorization directives are OD1 and FD1. The other directives are obligation directives. Real source policies may not be as cleanly organized as the directives in this guide. When dealing with these real source policies, we recommend that the policy author attempt to re-write the real source policies into organized collections of directives using the style of the directives in this guide.

3.2.3 Identification of Attributes and Predicates

Goals:

1. Understand how to identify the attributes and *predicates* of an authorization directive.

Summary:

When translating source policies into XACML, we need to first identify all the attributes and *predicates* that exist in the authorization directives of the source policy. We will step through this task for each source policy directive.

Steps:

3.2.3.1 Process the Local Organization Source Policy Directive

To translate source policies into XACML, the XACML author must determine which attributes (from the attribute dictionaries being used) are used in the policy, and what *predicates* exist in the policy. Identifying the attributes first will help us

identify the *predicates*. Our XACML policy will use standard XACML attributes and attributes from the GFIPM Metadata Specification.

There is a single directive in the local organization source policy. Directive OD1 reads "All sworn law enforcement officers of Agency-A who are authorized to search criminal history data in their home agency, may read any criminal history data for which the arresting officer is a member of Agency-A." Let's list all the attributes that appear in this directive:

1. "Sworn law enforcement officer" corresponds to the GFIPM Sworn Law Enforcement Officer Indicator attribute.

2. "Agency-A" is a value of the GFIPM Employer Name attribute.

3. The phrase "authorized to search criminal history data in their home agency" can correspond to one of two GFIPM criminal history data search privilege attributes: the Self privilege[59], or the Agency privilege[60]. Most current GFIPM-based federations use the Self privilege and not the Agency privilege. Therefore, in this tutorial, we will use the Self privilege which is represented by the GFIPM Criminal History Data Self Search Home Privilege Indicator attribute.

4. The action in this directive is "read". Actions are specified using the GFIPM Action Type attribute. Note that the appropriate value we need use in the XACML policy is "Read" (The GFIPM Metadata Spec specifies that for values of the Action Type attribute, the first letter needs to be capitalized).

   Note that we could have alternatively chosen to use the XACML "action-id" attribute to represent action values. A significant difference is that the XACML action-id attribute does not specify a code set of valid values.

5. "Criminal history data" corresponds to the GFIPM Criminal History Data Indicator attribute.

6. "Arresting officer" corresponds to a field in an Arrest Record. More specifically, we care about the field that contains the agency name of the arresting officer. This value does not correspond to a "named" attribute[61] from an attribute dictionary.

Now let's list all the *predicates* in directive OD1. Recall that, as explained in Lesson Step 3.1.1.1, a XACML *predicate* is a true or false statement about attributes.

---

[59] See http://gfipm.net/standards/metadata/2.0/user/CriminalHistoryDataSelfSearchHomePrivilegeIndicator.html.
[60] See http://gfipm.net/standards/metadata/2.0/user/CriminalHistoryDataAgencySearchHomePrivilegeIndicator.html.
[61] A named attribute is an **<Attribute>** element in a XACML request. This is in contrast to attribute values retrieved via an XPath expression in an **<AttributeSelector>** element in a policy.

1. Directive OD1 states that all sworn law enforcement officers may perform some action. Therefore the corresponding *predicate* is: "the GFIPM Sworn Law Enforcement Officer Indicator attribute of the Subject is true."

2. Directive OD1 states that the requestor must be a member of Agency-A. The corresponding *predicate* is: "the GFIPM Employer Name attribute of the Subject equals 'Agency-A'."

3. The directive states that the requestor must be authorized to search criminal history data in Agency-A. The corresponding *predicate* is: "the GFIPM Criminal History Data Self Search Home Privilege Indicator attribute of the Subject is true."

4. The directive is for the "read" action. The corresponding *predicate* is: "the GFIPM Action Type attribute equals 'Read'."

5. The directive states that criminal history data may be accessed. The corresponding *predicate* is: "the GFIPM Criminal History Data Indicator attribute of the Resource is true."

6. The directive states that the agency of the arresting officer must be Agency-A. The corresponding *predicate* is: "the field in the resource that refers to the agency of the arresting officer equals 'Agency-A'."

We will now put these *predicates* into tabular form to make them easier to process; Table 2 contains these *predicates*. Recall from Lesson Step 3.1.1.1 that a XACML *predicate* takes the form: "*attribute-expression*, Boolean operation, *attribute-expression*". The XACML *class* of each attribute is shown in parenthesis in the table.

Predicate Label	First Attribute Expression	Boolean Operation	Second Attribute Expression
OD1P1	GFIPM Sworn Law Enforcement Officer Indicator User attribute (Subject)	equals	"true"
OD1P2	GFIPM Employer Name User attribute (Subject)	equals	"Agency-A"
OD1P3	GFIPM Criminal History Data Self Search Home Privilege Indicator User attribute (Subject)	equals	"true"
OD1P4	GFIPM Action Type attribute (Action)	equals	"Read"
OD1P5	GFIPM Criminal History Data Indicator Resource attribute (Resource)	equals	"true"
OD1P6	The resource field corresponding to the agency of the arresting officer (Resource)	equals	"Agency-A"

Table 2: Predicate Table for Directive OD1

Directive OD1 permits access, for a given request, if all the *predicates* evaluate to true for that request.

3.2.3.2 Process the Federation Source Policy Directive

Directive FD1 reads: "All sworn law enforcement officers of the GFIPM Reference Federation, who are authorized to search criminal history data in their home agency, may read criminal history data, provided that the following are true: (1) the requestor is assigned to the jurisdiction of the record; and (2) the requestor has been authenticated at NIST level 3 or 4." Table 3 contains the list of *predicates* for directive FD1 in tabular form.

Predicate Label	First Attribute Expression	Boolean Operation	Second Attribute Expression
FD1P1	GFIPM Sworn Law Enforcement Officer Indicator attribute (Subject)	equals	"true"
FD1P2	GFIPM Identity Provider Id attribute (Subject)	begins with	"GFIPM:"
FD1P3	GFIPM Criminal History Data Self Search Home Privilege Indicator attribute (Subject)	equals	"true"
FD1P4	GFIPM Action Type attribute (Action)	equals	"Read"
FD1P5	GFIPM Criminal History Data Indicator attribute (Resource)	equals	"true"
FD1P6	GFIPM Electronic Authentication Assurance Level Code attribute (Subject)	equals	"NISTLEVEL3"
FD1P7	GFIPM Electronic Authentication Assurance Level Code attribute (Subject)	equals	"NISTLEVEL4"
FD1P8	GFIPM Employment Jurisdiction attribute (Subject)	at least one value equals	The resource field that corresponds to the jurisdiction of the record (Resource)

Table 3: Predicate Table for Directive FD1

Note that both attribute expressions of the eighth *predicate* include attributes and not literal values. Therefore, FD1P8 will need to be implemented in XACML as a rule condition[62].

Directive FD1 permits access, for a given request, if all of the following are true:

- Each of FD1P1, FD1P2, FD1P3, FD1P4, FD1P5, and FD1P8 are true
- At least one of FD1P6 or FD1P7 is true

Directives SD1 and SD2 express obligations. These obligations will be covered in Section 3.2.4.

[62] A *match-predicate* cannot express a *predicate* involving multiple attributes.

3.2.4 Obligation Design

Goals:

1. Understand our approach to designing XACML obligations.
2. Process the obligation directives.

Summary:

In this Lesson, we identify the arguments and semantics of each obligation directive, in preparation for translation into XACML. We explain the design approach that we use. However, recall that there are no known standardized obligation design patterns at the time of this writing.

Steps:

3.2.4.1 Process Directive SD1

Directive SD1 reads: "On successful access attempts, the arresting officer of the accessed record must be notified of the access via email." This directive expresses an obligation that requires a notification be sent to an entity.

In general, when designing obligations, the designer needs to specify the set of arguments and define the processing semantics for each obligation. The processing semantics needs to specify how the arguments are to be processed and what actions need to occur for the obligation to be considered fulfilled. For this tutorial, we will allow an argument of an obligation to be either an *attribute-expression* or a *predicate*.

The component that will fulfill this obligation will need to determine the address to which the notification will be sent. We will design the obligation to provide this address to the handler component. This address will be an argument to the obligation.

We will use "NotifyViaEmail" as the identifier for this obligation. The arguments are listed in Table 4.

Argument Identifier	Type	Value
Message	Literal	"[RecordId] has been accessed by [RequestorId] of [RequestorAgencyName] at [AccessDateTime]."
ArrestingOfficerEmailAddress	Attribute	The Resource field corresponding to the email address of the arresting officer (Resource)
RecordId	Attribute	XACML resource-id attribute (Resource)
RequestorId	Attribute	GFIPM Federation Id attribute (Subject)
RequestorAgencyName	Attribute	The Resource field corresponding to the agency name of the arresting officer (Resource)
AccessDateTime	Attribute	XACML current-dateTime attribute (Environment)

Table 4: Arguments of the NotifyViaEmail Obligation

In the value of the Message argument, there are four terms that are in square brackets. These terms are the identifiers of four other arguments. For the obligation design pattern used in this tutorial, the obligation handler is expected to replace these terms with the values of the respective arguments. If the obligation handler cannot successfully perform a replacement, then the obligation must be treated as not fulfilled.

The description of the processing semantics for this obligation follows. The Message must be sent via email to ArrestingOfficerEmailAddress. If the email is sent successfully, then the obligation is fulfilled, otherwise the obligation is not fulfilled.

Note that this processing semantics does not require any confirmation that the email has been received by the appropriate recipient. The obligation can be treated as fulfilled even if a "bounce-back" email is returned to the sender.

3.2.4.2 Process Directive SD2

Directive SD2 reads: "Successful attempts to access data must be logged for at least 60 months. The following data must be logged: (1) the requestor id, (2) the record id, (3) the action, and (4) the date-time of the access." This directive expresses an obligation to write to an audit log. This obligation is able to be fulfilled by a handler that is local to the PEP.

We will use "LogValidAccess" as the identifier of this obligation. The arguments are in Table 5.

Argument Identifier	Type	Value
RequestorId	Attribute	GFIPM Federation Id attribute (Subject)
RecordId	Attribute	XACML resource-id attribute (Resource)
ActionType	Attribute	GFIPM Action Type attribute (Action)
AccessDateTime	Attribute	XACML current-dateTime attribute (Environment)
ExpirationDateTime	Attribute (Manipulated)	XACML current-dateTime attribute (Environment) plus “60 months”.

Table 5: Arguments of the LogValidAccess Obligation

The processing semantics of this obligation are: A log entry must be written to the PEP’s local audit log. The entry must include RequestorId, ResourceId, ActionType, and AccessDateTime. The entry must be configured to not be deleted prior to the date and time specified by the ExpirationDateTime argument.

3.2.5 Sample Implementation Users, Resources, and Test Cases

Goals:

1. Review the user accounts that were created for testing the sample implementation.
2. Review the resources that were created for testing the sample implementation.
3. Review the test cases we will use to verify that our XACML policy and sample implementation have been implemented correctly.

Summary:

Five test user accounts were provisioned in the GFIPM Reference Federation for the purposes of testing the sample implementation. Two Arrest Record files were created for testing. We will explore the attributes of these user accounts and details of the records. We will then review the test cases that we will use to verify that our XACML policy and sample implementation have been implemented correctly. We will determine the expected policy evaluation results of executing each test case against the *predicates* and obligations that we designed in Lessons 3.2.3 and 3.2.4.

Steps:

3.2.5.1 Review the Sample Implementation Users

Table 6 contains a list of the federation user accounts that will be used for testing the policies and the sample application. The first six columns correspond to GFIPM user attributes.

User Id	Employer Name	Jurisdiction [63]	SLEO? [64]	CHD Search? [65]	Auth Level [66]	Identity Provider Id
xu01	Agency-A	GA	true	false	NISTLEVEL4	Agency-A
xu02	Agency-A	GA	true	true	NISTLEVEL2	Agency-A
xu03	Agency-A	VA	true	true	NISTLEVEL3	Agency-A
xu04	Agency-B	GA	true	true	NISTLEVEL2	Agency-B
xu05	Agency-B	VA	true	true	NISTLEVEL3	Agency-B

Table 6: Sample Implementation Users

The user ids in the first column are an abbreviation of each user's GFIPM Federation Id attribute value. Federation Id attribute values are of the form "GFIPM:TIB:XACMLTestBroker:IDP:<EmployerName>:USER:<user-id>"[67], where <EmployerName> is the actual name of the user's employer, as shown in the second column, and <user-id> is the abbreviated identifier in the first column.

The identity provider identifiers in the last column are also abbreviated. The full identifiers are of the form "GFIPM:TIB:XACMLTestBroker:IDP:<EmployerName>"[68], where <EmployerName> is the actual name of the user's employer, as shown in the second column.

Note that the email address of each user is not shown in Table 6. The email address of a user, in our sample implementation, is <user-id>@<EmployerName>.gov.

3.2.5.2 Review the Sample Implementation Resources

Two criminal history records, in the form of Arrest Records, are provided in the sample application. The details of these records are shown in Table 7.

[63] This column corresponds to the GFIPM Employment Jurisdiction attribute.
[64] This column corresponds to the GFIPM Sworn Law Enforcement Officer Indicator attribute.
[65] This column corresponds to the GFIPM Criminal History Data Self Search Privilege Indicator attribute.
[66] This column corresponds to the GFIPM Electronic Authentication Assurance Level Code attribute.
[67] Details on the parts of this identifier are at http://gfipm.net/standards/metadata/2.0/user/FederationId.html.
[68] Details on the parts of this identifier are at http://gfipm.net/standards/metadata/2.0/user/IdentityProviderId.html.

Record Id	Subject Id	Jurisdiction	Date	Officer Id
Record-1	Subject-1	GA	2012-01-19	xu02
Record-2	Subject-2	VA	2012-02-21	xu03

Table 7: Sample Implementation Resources

The first record states that officer xu02 arrested Subject-1 on January 19, 2012, in Georgia. The second record states that officer xu03 arrested Subject-2 on February 21, 2012, in Virginia.

The XML files that correspond to these records are in the "$POLICY_GUIDE/data_records/" directory.

3.2.5.3 Review the Sample Implementation Test Cases

Table 8 contains six test cases that will be used to test the policy and sample application. Each test case represents a certain user attempting to read a certain record. The table shows the expected results of evaluating the two non-obligation, source directives against each test case.

Test Case #	Requestor Id	Record Id	OD1 Evaluation	FD1 Evaluation
1	xu01	Record-1	NotApplicable	NotApplicable
2	xu02	Record-2	Permit	NotApplicable
3	xu03	Record-2	Permit	Permit
4	xu04	Record-1	NotApplicable	NotApplicable
5	xu05	Record-1	NotApplicable	NotApplicable
6	xu05	Record-2	NotApplicable	Permit

Table 8: Sample Implementation Test Cases

When a test case will cause an appropriate combination of *predicates* in the source policy directives to be true, then the corresponding directives will evaluate to "Permit". For each scenario in which a source policy directive evaluates to "NotApplicable", we provide the reasoning in the descriptions that follow.

Test Case 1 will cause directives OD1 and FD1 to evaluate to "NotApplicable" because user xu01 does not have the criminal history data self search privilege. *Predicates* OD1P3 and FD1P3 will be false for this test case.

Test Case 2 will cause directive FD1 to evaluate to "NotApplicable" because *predicates* FD1P6 and FD1P7, concerning the GFIPM Electronic Authentication Assurance Level Code attribute, will both be false. Also, *predicate* FD1P8 will be false due to the jurisdictions of the user and the record not being equal.

Test Case 4 will cause directive OD1 to evaluate to "NotApplicable" because user xu04 is not in Agency-A; *predicate* OD1P2 will be false. This test case will cause directive FD1 to evaluate to "NotApplicable" because user xu04 has a GFIPM Electronic Authentication Assurance Level Code attribute value of "NISTLEVEL2"; *predicates* FD1P6 and FD1P7 will be false.

Test Case 5 will cause directive OD1 to evaluate to "NotApplicable" because user xu05 is not in Agency-A; *predicate* OD1P2 will be false. This test case will cause directive FD1 to evaluate to "NotApplicable" because the jurisdictions of the user and the record not being equal; *predicate* FD1P8 will be false.

Test Case 6 will cause directive OD1 to evaluate to "NotApplicable" because user xu05 is not in Agency-A; *predicate* OD1P2 will be false.

In Lesson 3.2.6, we will translate the source policy directives into XACML policies, translate the test cases into XACML requests, execute the test cases, and compare the results with the expected outcomes that were covered in this Step.

3.2.6 XACML Policy Implementation

Goals:

1. Understand how to translate the test cases into XACML requests.
2. Understand how to translate the designed *predicates* and obligations into a XACML policy.
3. Test our XACML policy against the XACML requests.

Summary:

In this Lesson, we explain our strategy for implementing the designed *predicates,* obligations, and test cases into XACML. This strategy includes using a combining algorithm to aggregate multiple individual XACML policies together to form a single, top-level policy for the sample application. You will be challenged with using the knowledge presented in Lesson Group 3.1 to create the XACML implementations.

Steps:

3.2.6.1 Review the Implementation Strategy

There are three levels of authority represented by the source directives: Agency-A (OD1); the federation (FD1, FD2); and Agency-A supplementing the federation directives (SD1, SD2). We will create a separate XACML policy for each level of authority. This will result in a module policy design in which changes to a source policy of a single authority will be isolated in a single XACML policy. Since all agencies in the federation will abide by the federation-level source policy, the XACML policy representing this source policy should theoretically be able to be shared and installed by each agency.

The three XACML policies will need to be aggregated into a single **<PolicySet>**. The three individual policies and the aggregated policy will be stored at the "$POLICY_GUIDE/policies/" directory.

3.2.6.2 Challenge: Create a XACML Request for Each Test Case

In the "$POLICY_GUIDE/test_cases/" directory, create a XACML request for each test case in Table 8. Name each request file: "Request-<test-case-number>-Challenge.xml". For example, name the file for the second test case: "Request-2-Challenge.xml".

In the Subject section of each request, include the following attributes:

- GFIPM Federation Id
- GFIPM Employer Name
- GFIPM Employment Jurisdiction
- GFIPM Sworn Law Enforcement Officer Indicator
- GFIPM Criminal History Data Self Search Home Privilege Indicator
- GFIPM Electronic Authentication Assurance Level Code
- GFIPM Identity Provider Id
- GFIPM Email Address Text

Populate the values of these attributes with the values from Table 6 and Section 3.2.5.1.

In the Resource section of each request, include the appropriate data record in the **<ResourceContent>** element, and include the following attributes:

- XACML resource-id
    - The value of this attribute is the id of the requested record.
- GFIPM Criminal History Data Indicator
    - The value of this attribute should be "true" (without the quotes).

In the Action section of each request, include the GFIPM Action Type attribute with a value of "Read" (without the quotes).

In the Environment section of each request, include the XACML current-dateTime attribute with a value that you choose. You should choose a date that is after 2012-02-21, the date of the latest data record, to make sure our tests simulate that the requests happen after the data records have been created. The format for the date-time value is "YYYY-MM-DDTHH:MM:SS"[69], where:

- YYYY is a four digit year
- MM is a two digit month
- DD is a two digit day
- HH is a two digit hour value (use the 24 hour clock style)
- MM is a two digit minute value
- SS is a two digit second value
- - is the literal "-" character separating the date parts
- T is the literal "T" character denoting the beginning of the time section
- : is the literal ":" character separating the time parts.

Solutions for this Challenge are in the "$POLICY_GUIDE/test_cases" directory. Compare each of your requests with the corresponding solution. For example compare Request-3-Challenge.xml with Request-3.xml.

3.2.6.3 Challenge: Create a XACML Policy for the Local Organization Source Directives

In the "$POLICY_GUIDE/policies/" directory, create a file called "Agency-A-Policy-Challenge.xml". Create the XACML policy in this file. Use the *predicates* in Table 2 as a reference. Recall from the Lessons in Section 3.1 that a *predicate* in XACML can take the form of a *match-predicate* or a rule **<Condition>**. A *match-predicate* includes an *attribute-reference*, which can be an *attribute-designator* or an **<AttributeSelector>**. Also, recall that a XACML policy can have a top-level **<Policy>** or **<PolicySet>** element.

A solution to this Challenge is in Agency-A-Policy.xml in the "$POLICY_GUIDE/policies/" directory.

3.2.6.4 Inspect Agency-A-Policy.xml

This policy has a top-level **<Policy>** element. All of the *predicates* are in the **<Target>** of the **<Policy>**. It contains a single "Permit" **<Rule>** that has an empty **<Target>**. Each *predicate* from Table 2 is represented by a *match-predicate* in **<Target>** of the **<Policy>** as follows:

[69] XACML uses the XML Schema format for the dateTime data type. See http://www.w3.org/TR/xmlschema-2/.

- OD1P1 (Lines 22 – 26)
- OD1P2 (Lines 29 – 33)
- OD1P3 (Lines 36 – 41)
- OD1P4 (Lines 66 – 70)
- OD1P5 (Lines 48 – 52)
- OD1P6 (Lines 55 – 59)

3.2.6.5 Evaluate the Local Organization XACML Policy

First, confirm that evaluating the Agency-A-Policy against the provided requests yield the expected results. Output each XACML response to a file named: "Request-<test-case-number>_Agency-A-Policy_Reponse.xml" in the "$POLICY_GUIDE/test_case_results/" directory. Confirm that each response has the expected result as stated in Table 8.

Now, evaluate your Challenge policy against each of the provided requests. Output each XACML response to a file named: "Request-<test-case-number>_Agency-A-Policy-Challenge_Response.xml" in the "$POLICY_GUIDE/test_case_results/" directory. Confirm that each response has the expected result as stated in Table 8.

Now, evaluate the provided policy against each of your Challenge requests. Output each XACML response to a file named: "Request-<test-case-number>-Challenge_Agency-A-Policy_Response.xml" in the "$POLICY_GUIDE/test_case_results/" directory. Confirm that each response has the expected result as stated in Table 8.

3.2.6.6 Challenge: Create a XACML Policy for the Federation Source Directives

In the "$POLICY_GUIDE/policies/" directory, create a file called "Federation-Policy-Challenge.xml". Create the XACML policy in this file. To handle *predicates* FD1P2, FD1P6, and FD1P7, consider using a XACML regular expression function.

A solution to this Challenge is in Federation-Policy.xml.

3.2.6.7 Inspect Federation-Policy.xml

The implementation of *predicates* FD1P1 (Lines 20 – 24), FD1P3 (Lines 35 – 40), FD1P4 (Lines 67 – 71), and FD1P5 (Lines 56 – 60) are the same as in Agency-A-Policy.

The implementation of *predicate* FD1P2 (Lines 27 – 32) uses the string-regexp-match XACML function, which is a regular expression function[70].

The implementation of *predicates* FD1P6 and FD1P7 is handled in a single *match-predicate* (Lines 43 – 49) using the string-regexp-match function.

As stated in Lesson Step 3.2.3.2, directive FD1P8 will need to be implemented as a condition. This condition is on Lines 85 – 93.

3.2.6.8 Evaluate the Federation XACML Policies

First, confirm that evaluating the Federation-Policy against the provided requests yield the expected results. Output each XACML response to a file named: "Request-<test-case-number>_Federation-Policy_Reponse.xml" in the "$POLICY_GUIDE/test_case_results" directory. Confirm that each response has the expected result as stated in Table 8. For the XACML response that have a "Permit" decision, confirm that the proper obligation is present.

Now, evaluate your Challenge policy against each of the provided requests. Output each XACML response to a file named: "Request-<test-case-number>_Federation-Policy-Challenge_Response.xml" in the "$POLICY_GUIDE/test_case_results" directory. Confirm that each response has the expected result as stated in Table 8. For the XACML response that have a "Permit" decision, confirm that the proper obligation is present.

Now, evaluate the provided policy against each of your Challenge requests. Output each XACML response to a file named: "Request-<test-case-number>-Challenge_Federation-Policy_Response.xml" in the "$POLICY_GUIDE/test_case_results" directory. Confirm that each response has the expected result as stated in Table 8. For the XACML response that have a "Permit" decision, confirm that the proper obligation is present.

3.2.6.9 Strategy for Implementing the Supplemental Directives

The supplemental directives, SD1 and SD2, are obligations created by Agency-A to supplement the federation directives. As such, and because we don't want to edit the federation XACML policy directly, we will create a new policy that (1) has a **<PolicySet>** top-level element, (2) references the federation XACML policy, and (3) implements directives SD1 and SD2.

[70] Regular expressions in XACML are handled as specified in the W3C "XQuery 1.0 and XPath 2.0 Functions and Operators" specification, located at http://www.w3.org/TR/2002/WD-xquery-operators-20020816.

3.2.6.10 Challenge: Implement the Supplemental Directive Obligations

Recall from Lesson 3.1.8, when including XML element arguments inside of an **<AttributeAssignment>** element, the author needs to replace all occurrences in the argument of "<" with "<" and replace all occurrences of ">" with ">" (URL encoding).

To ease the burden of directly authoring obligations using the URL encoding, first author the obligations using the normal notation for XML elements. Create a file, in the "$POLICY_GUIDE/policies" directory, named: "Supplemental-Obligations-Challenge.xml". Create a top-level **<Obligations>** element and specify the XACML policy namespace as the default namespace. Create the implementations of SD1 and SD2 within the top-level **<Obligations>** element.

To implement the ExpriationDate argument of the LogValidAccess obligation, consider using the **<Apply>** element to specify a call to the date-add-yearMonthDuration XACML function.[71]

A solution to this Challenge is in Supplemental-Obligations.xml. Inspect this file and confirm that each **<AttributeAssignment>** element matches the appropriate argument as stated in Table 4 and Table 5.

We will use this set of obligations in the full implementation of the supplemental directives.

3.2.6.11 Inspect Supplemented-Federation-PolicySet.xml

Line 13 contains the reference to the federation XACML policy.

The obligations from Supplmental-Obligations.xml have been copied into this file and all the occurrences of "<" and ">" inside all **<AttributeAssignment>** elements have been replaced with their respective URL encodings.

3.2.6.12 The Complete Policy for the Sample Implementation

As stated in Section 2.3.1, Step 3A, an application should be supported exactly one top-level policy. We will now create the top-level XACML policy for the sample application.

This top-level policy needs to include the local organization XACML policy and the supplemented federation policy. There are requests that will apply to both of these

[71] Durations are handled in XACML as specified in the W3C XML Schema Part 2: Datatypes Second Edition Sepcification, located at http://www.w3.org/TR/xmlschema-2/ and in the W3C XQuery 1.0 and XPath 2.0 Functions and Operators specification, located at http://www.w3.org/TR/2002/WD-xquery-operators-20020816.

subordinate policies. However, if a request applies to the local organization policy, which essentially handles request from internal users, there is no need to process any of the obligations. Therefore, we will use the first-applicable policy combining algorithm to configure the top-level policy to have the PDP first evaluate the local organization policy and then only evaluate the supplemented federation policy if necessary.

The sample application is used to access criminal history data and any request to access such data should be applicable to the top-level policy, before considering the subordinate policies. Therefore, we will include a *predicate* in the **<Target>** of the top-level **<PolicySet>** ensuring that the policy will only be applicable to requests to access criminal history data.

With this policy configuration, there are requests that are applicable to the **<Target>** of the top-level policy, but are not applicable to any of the subordinate policies. These requests will cause the top-level policy to evaluate to "NotApplicable". This is misleading, as these requests should theoretically be applicable to the top-level policy, and the top-level policy should evaluate to "Deny" for these requests. To achieve this, we will include a third subordinate policy, placed at the end of the top-level policy, which denies all requests. Therefore, if the sample application supplies a request to access criminal history data, and the first two subordinate policies are not applicable to that request, then the top-level policy will return a decision of "Deny" instead of "NotApplicable".

The top-level sample application policy is implemented in Sample-Application-Policy.xml. Open this file. The *predicate* for ensuring that requests are for accessing criminal history data is on Lines 13 – 17. Line 22 contains the reference to Agency-A-Policy. Line 24 contains the reference to Supplemented-Federation-Policy. The deny policy is on Lines 26 – 31.

3.2.6.13 Testing the Sample Application Policy

We will now re-examine the six test cases and discern their effects on the evaluation of the sample application policy. Table 9 shows the expected result of the sample application policy when evaluated against each test case request.

Test Case #	Requestor Id	Record Id	Sample Application Policy Evaluation	Obligations
1	xu01	Record-1	Deny	N/A
2	xu02	Record-2	Permit	N/A
3	xu03	Record-2	Permit	N/A
4	xu04	Record-1	Deny	N/A
5	xu05	Record-1	Deny	N/A
6	xu05	Record-2	Permit	All

Table 9: The Effects of the Test Cases on the Sample Application Policy

While the XACML specification allows policy references, it does not specify how policy references are to be resolved. Fittingly, SunXACML supplies a mechanism for developers to provide a module that resolves policy references and we have developed such a module. We use a SunXACML XML configuration file ("SunXACMLConfig.xml" in the "$POLICY_GUIDE/policies/" directory) to instruct SunXACML to use Sample-Application-Policy.xml as the top-level policy and to use our module for resolving policy references. The policy reference module also has its own configuration file ("ReferencePolicyFinderConfig.xml" in the "$POLICY_GUIDE/policies/" directory). Details on how to invoke the SimplePDP using a configuration file is in Appendix A: Common Tasks. A more detailed description of how the SunXACML configuration and policy reference module configuration files are used is in Lesson Step 3.3.2.1.

Now evaluate the sample application policy against each test case request and output the XACML response to a file named "Request-<test-case-number>_Sample-Application-Policy_Response.xml. Confirm that the actual results are the same as those in Table 9.

3.3 The Sample Implementation Components

This Lesson Group contains Lessons that cover the functionality, configuration, and integration details of the components of the sample implementation.

3.3.1 Overview of the Sample Implementation

Goals:

1. Understand the sample implementation architecture.
2. Understand how the sample implementation uses GFIPM Web Services.
3. Understand how the sample implementation software is packaged.

Summary:

In this Lesson, we provide an overview of the design and structure of the sample implementation. We describe how the sample implementation architecture maps to the XACML reference architecture. We provide an overview of the GFIPM Web Services User-Consumer-Provider Service Interaction Profile, which is used by the sample implementation. Finally, we describe how the sample implementation software is organized and packaged.

Steps:

3.3.1.1 The Sample Application Architecture and Design

The sample application is written in Java, and includes a consumer and a provider of a GFIPM Web Service. It runs in the Glassfish application server and makes use of an operational GFIPM Identity Provider service running in the GFIPM Reference Federation. Figure 11 depicts the architecture of the sample application, and Table 10 maps the sample application components to the XACML reference architecture.

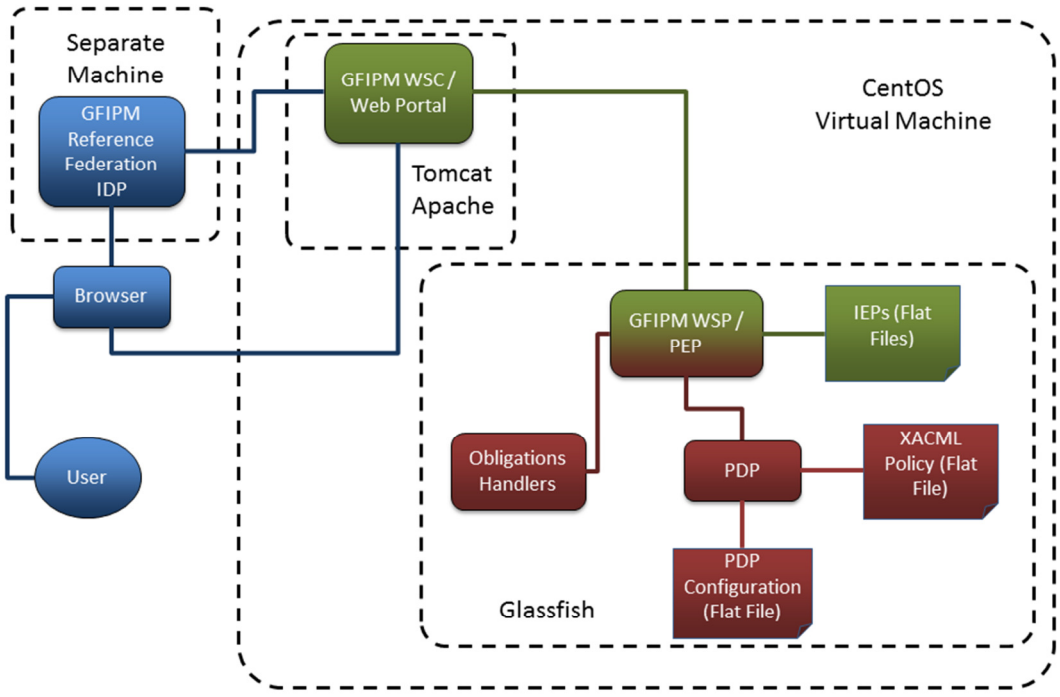

Figure 11: Sample Application Architecture

XACML Reference Architecture Component(s)	Sample Implementation Component(s)	Description
Requestor	User, Browser, Identity Provider (IDP), Web Service Consumer (WSC)	The User connects to the WSC Web Portal via a Browser. The WSC makes a Web Service call to the WSP on behalf of the User using attributes provided from the IDP.
PEP	Web Service Provider (WSP)	The WSP is the GFIPM Web Service Provider and enforces access control decisions made by the PDP
Resource	XML Data (Flat Files)	The WSP reads the XML data files directly from disk.
Obligations Handlers	Obligation Handlers	There is a handler component for each obligation present in the sample implementation XACML policy.
PDP	PDP, PDP Configuration	The sample application PDP uses the SunXACML library.
Policy Repository	XACML Policy (Flat File)	The PDP reads the XACML Policy directly from disk.
PAP, PIP, Supplemental Attribute Authorities	N/A	These components are not included in the sample application.

Table 10: Mapping of the Sample Application Architecture to the XACML Reference Architecture

3.3.1.2 GFIPM Web Services

GFIPM Web Services (GFIPM-WS) is defined by the System-to-System Profile of GFIPM. It is a normative technical specification containing a set of profiles that enable secure, interoperable, standards-based SOAP web services communication within a GFIPM federation.

One of the profiles described in the GFIPM-WS System-to-System Profile is called the GFIPM-WS User-Consumer-Provider SIP. It is designed to apply to any information exchange scenario in which a Web Service Consumer (WSC) interacts with a Web Service Provider (WSP) on behalf of a user. This scenario can occur whenever a user interacts with an application that requires user identity information for the purpose of access control, auditing, etc. This SIP is often applied to a scenario in which a web portal serves the requests of its authenticated users by performing back-end web services transactions on behalf of those users, as done by our sample

implementation. A high-level description of this SIP's behavior and message flow is as follows. Numbered steps in the description below correspond to the steps depicted in Figure 12.

Step 1: The (yet-to-be-authenticated) user browses to the web portal (also acting as the WSC) using a web browser, and the WSC authenticates the user. The authentication event is outside the scope of the GFIPM-WS User-Consumer-Provider SIP; however, since the basic GFIPM paradigm is built on the concept of federated identity management, authentication of the user to the web portal often requires the use of a GFIPM Identity Provider service, and is typically performed using the GFIPM Web Browser User-to-System Profile.[72]

Steps 2-3-4: The web portal (WSC) performs a web services transaction with a WSP on behalf of the user, using a SAML assertion issued and signed by the user's IDP to fulfill the WSP's requirement for attributes about the user.[73] During this transaction, the WSP uses the attributes in the SAML assertion to make a policy decision about whether to release the requested data to the WSC.

Step 5: The WSC presents the results to the user, if the WSP released the data.

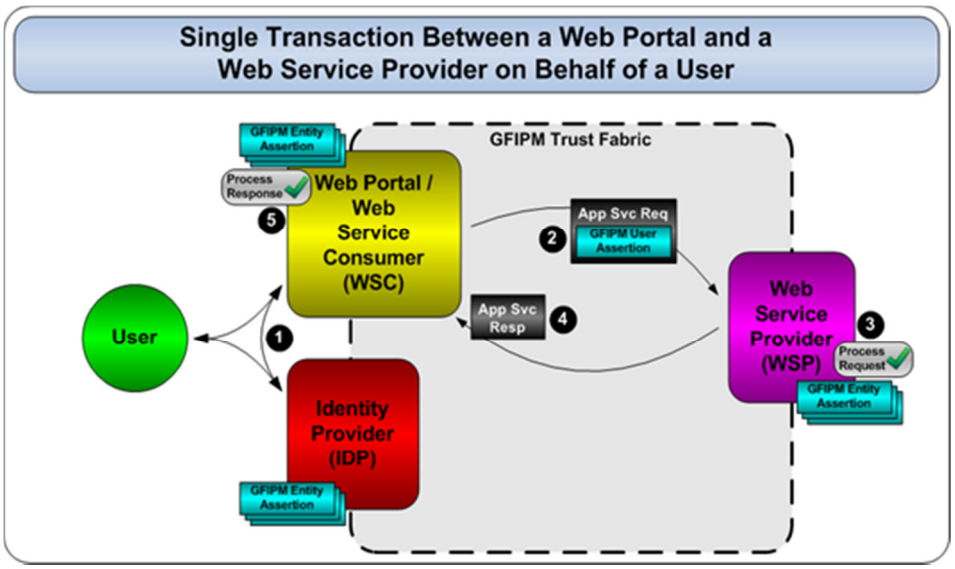

Figure 12: GFIPM Web Services User-Consumer-Provider SIP

[72] For more information, see http://it.ojp.gov/docdownloader.aspx?ddid=1336.

[73] This description omits certain details about how the WSC obtains a SAML assertion that is suitable for presentation to the WSP. For more information, please see Section 8.8 ("GFIPM-WS SAML Assertion Delegate Service SIP") of the GFIPM Web Services System-to-System Profile.

This project uses the Java reference implementation of GFIPM Web Services, which is based on the Metro Java project[74], which is the reference implementation of the Java API for XML-Based Web Services (JAX-WS) specification[75].

3.3.1.3 The Sample Implementation Software Project Structure

The sample implementation consists of two parts: (1) the sample information-sharing web service application; and (2) the consumer of the web service application.

The information-sharing application includes a GFIPM Web Service Provider (WSP) which is the PEP of the XACML architecture. The WSP provides a service that allows consumers to retrieve arrest records. The application uses the SunXACML PDP and a set of Obligation Handlers.

The consumer of the application is a web portal that has a GFIPM Web Service Consumer (WSC) module. The web portal provides a web interface that allows users to (1) select an arrest record to retrieve, and (2) read the contents of retrieved arrest records. The web portal uses the WSC to communicate with the WSP. The web portal authenticates GFIPM Reference Federation users against the GFPIM Reference Identity Provider (IDP)[76]. The users specified in the test cases of Lesson 3.2.5 have been provisioned in the GFIPM Reference IDP.

The sample implementation is packaged as a series of binary and Apache Maven[77] (referred to herein as Maven) modules in the "$SAMPLE_IMPL/" directory. The binary modules provide functionality necessary to run the sample implementation, but the details of their structure are not important to this guide. Each Maven module contains the source code of the module and the remaining Lessons will explore the details of these modules. Table 11: List of Sample Implementation Modules contains a description about each module.

[74] See http://jcp.org/en/jsr/detail?id=224.
[75] See http://jax-ws.java.net/.
[76] Authentication is performed using the GFIPM Web Browser User-to-System Profile. For more information, see http://it.ojp.gov/docdownloader.aspx?ddid=1336.
[77] Apache Maven is a software project management and comprehension tool. See http://maven.apache.org/.

Module Name	Type	Location ($LABEL)	Description
STS/ADS	Binary Module	m2sts.war	A module required by GFIPM Web Services
SunXACML	Binary Module	sunxacml.jar	The SunXACML library
Obligations	Maven Module	obligation/ ($OBLG_PRJ)	Core obligation handling functionality
SunXACML Modules	Maven Module	ImplGuideSunXACML/ ($SX_MOD_PRJ)	Modules that support the SunXACML functionality
WSLib	Binary Module	wslib.jar	Core web service configuration and functionality
WSC	Maven Module	wsc/ ($WSC_PRJ)	The WSC/Web Portal implementation
WSP	Maven Module	wsp/ ($WSP_PRJ)	The WSP implementation

Table 11: List of Sample Implementation Modules

The locations are relative to the "$SAMPLE_IMPL" directory. The label for each Maven module is listed in parenthesis after the location. Details on building, installing, and deploying these modules are in Appendix A: Common Tasks.

The XML data resources developed in the test cases of Lesson 3.2.5 have been copied into the WSP project at the $WSP-PRJ/src/main/webapp/META-INF/records directory. The sample implementation XACML policies that were developed in Lesson 3.2.6 have been copied into the WSP project at the $WSP-PRJ/src/main/webapp/META-INF/policies directory[78].

3.3.2 Implementation of the Policy Services

Goals:

1. Understand how the SunXACML PDP is configured.
2. Understand how the Obligation Handlers are implemented.

Summary:

In this Lesson, we explore the configuration and details of the policy services components that were implemented- the PDP and Obligation Handlers.

Steps:

[78] The configuration files are different since the sample application runs in a web application context.

3.3.2.1 PDP Configuration

The SunXACML library includes a PDP module that can be programmatically used by a Java application. The SunXACML PDP module needs to be supplied with modules for retrieving the XACML policy (or policies) that the PDP will evaluate- these modules are called "policy finder modules". There are two types of policy finder modules. The first type, which we'll call "context policy finder modules", will be asked to supply a single policy based on a XACML request that the PDP needs to evaluate. The second type, which we'll call "reference policy finder modules", will be asked to supply a single policy based on a policy reference. Also, the PDP will need to be supplied with modules that can retrieve the values for (1) attributes referenced by **<AttributeSelector>** elements and (2) supplemental attributes- these modules are called "attribute finder modules".

The PDP can be configured with a SunXACML XML configuration file[79]. The configuration file allows an administrator to specify the policy and attribute finder modules that the PDP will use. The configuration file also allows an administrator to instruct the PDP to use modules that implement extensions to the XACML standard; extensions can consist of implementations of non-standard datatypes, combining algorithms, or functions.

The WSP Project uses the SunXACML configuration file located at "$WSP_PRJ/src/main/webapp/META-INF/policies/SunXACMLConfig.xml". This configuration file specifies the use of two policy finder modules and a single attribute finder module. The first policy finder module (Lines 6 – 11) is the org.gtri.icl.iead.policy_guide.sunxacml.ReferencePolicyFinderModule Java class; this is a reference policy finder module that is in the $SX_MOD_PRJ module. This class needs to be initialized with the "UseServletContext" flag, the path to where the referenced policies reside, and the path to its own configuration file that maps policy identifiers to file paths (Lines 8 - 10).

The second policy finder module (Lines 12 – 17) is the org.gtri.icl.iead.policy_guide.sunxacml.WSFilePolicyModule; this is a context policy finder module that is in the $SX_MOD_PRJ module. This class needs to be initialized with the "UseServletContext" flag and a list of files containing XACML policies (see Lines 13 – 16). For a given XACML request, the WSFilePolicyModule will find a single applicable policy from the list; otherwise it will throw an error. In this configuration file, it is initialized with a single policy- the sample implementation policy (Line 15). The SunXACML library already contains a module for retrieving policies from files on the local file system[80]. However, the WSP executes within a Java servlet context and the built-in SunXACML module will not work in this context.

[79] This is the same configuration file format that was used for the SunXACML SimplePDP module in Lesson 3.2.6. The SimplePDP Java class is an extension of the SunXACML PDP that allows the PDP to be used via a command line.
[80] This module is the com.sun.xacml.finder.PolicyFinderModule Java class.

The attribute finder (Line 18) is the built-in SunXACML com.sun.xacml.finder.impl.SelectorModule Java class. It is used to evaluate **<AttributeSelector>** elements.

Lines 20 - 23, of the configuration file, instruct the SunXACML library to use the standard XACML datatypes, combining algorithms, and functions.

3.3.2.2 The Obligation Handlers

The sample implementation policies include two obligations: NotifyViaEmail and LogValidAccess. We have developed a separate Java class for handling each of those obligations. These are the NotifyViaEmailImpl and LogValidAccessImpl Java classes in the org.gtri.icl.iead.policy_guide.obligation Java package in the $WSP_PRJ module.

Our obligation design allows the use of XACML *attribute-expressions* in obligation arguments[81]. Any *attribute-expressions* in an obligation need to be resolved into literal values before the obligation can be fulfilled. We designed the obligation handler implementations to operate on arguments that have been resolved. We have developed an "obligation handler wrapper" class[82] that uses the modules of the SunXACML library to evaluate the *attribute-expressions* of obligation arguments[83].

The WSP associates each instance of each obligation handler implementation with a separate instance of the wrapper class. To fulfill an obligation, the WSP asks the appropriate wrapper to fulfill the obligation, the wrapper resolves any *attribute-expressions*, and then the wrapper provides its associated obligation handler with the resolved arguments to fulfill the obligation. In XACML 3.0, the resolution of *attribute-expressions* in obligation arguments can be handled by the PDP, and our obligation handler wrapper may not be needed.

3.3.3 The GFIPM WSP / PEP

Goals:

1. Understand the service interfaces and functionality of the WSP.

Summary:

[81] The use of *attribute-expressions* in obligation arguments provides the basis for the fulfillment of the "entity resolution" architectural requirement discussed in Section 2.3.1 Step 6(B).
[82] This is the org.gtri.icl.iead.policy_guide.obligation.ObligationHandlerWrapper class in the $OBLG_PRJ module.
[83] We had to make a very small change to the SunXACML library to allow it to support this functionality. The sample implementation includes this modified SunXACML library and does not include the library provided by Sun/Oracle. The library provided directly by Sun/Oracle can support the entire functionality of the sample implementation except the obligation handlers.

In this Lesson, we will describe the configuration and functionality of the WSP / PEP component and describe how it interacts with the Obligations Handler and the PDP.

Steps:

3.3.3.1 WSP Implementation

The org.gtri.icl.iead.policy_guide.service.SampleWebServiceImpl class implements the actual web service read operation. On receipt of a read service request, this class performs the following steps.

1. Uses the DataAdapter class to retrieve the requested arrest record.

2. Builds a XACML request based on the user's SAML assertion, and the retrieved arrest record (See Lesson Step 3.3.3.2).

3. Obtains an access control decision (XACML response) from the PDP. To do this, the WSP first initializes the PDP using the SunXACML configuration file[84] (see Lesson Step 3.3.2.1). Then the WSP invokes the "evaluate" method on the PDP.

4. Checks to see if the XACML response is valid, i.e., it has a status of "ok". If the status is not "ok", then a fault[85] is returned to the requestor. If the status is "ok", then processing continues.

5. Checks to see if the XACML decision is "Indeterminate". If the decision is "Indeterminate", then a fault is returned to the requestor, otherwise processing continues.

6. Processes obligations. For each obligation received in a XACML response, the WSP identifies the appropriate obligation handler wrapper based on the obligation ID, and passes the obligation to that wrapper for fulfillment. The wrapper resolves all obligation arguments and provides the resolved arguments to its associated obligation handler for fulfillment. If any obligation is not fulfilled, then the WSP returns a fault to the service requestor.

7. Adds the XACML request and response (for demo purposes) to the service response.

8. Enforces the XACML decision. If the decision is "Permit", then the arrest record is added to the service response. If the decision is not "Permit" (it is

[84] In a production system, the WSP may have no knowledge of the PDP configuration. The configuration of the PDP can occur via a separate mechanism.
[85] A fault is a SOAP-based web service message that encodes server-side error conditions.

either “Deny” or “NotApplicable”), then the arrest record is not added to the service response.

9. Return the service response to the requestor.

If any other errors or exceptions are caught during the processing of these steps, then a fault is returned to the requestor.

3.3.3.2 Building the XACML request

Building the XACML request involves creating attributes for the Subject, Resource, Action, and Environment *classes* based on the received service request and the requested data record. Subject attributes are created from the SAML assertion that was received with the service request. Resource, Action, and Environment attributes are generated by the WSP based on the contextual details of the service request.

Creating the Subject attributes from the SAML assertion

The SAML assertion will contain a list of GFIPM user attributes pertaining to the requestor. The WSP converts each SAML attribute into a XACML Subject attribute and inserts this XAML attribute into the XACML request. Figure 13 shows an instance of the Sworn Law Enforcement Officer Indicator GFIPM attribute as it would appear in a SAML assertion. Figure 14 shows the same attribute as it would appear in a XACML request.

```
<!-- The namespace for the "saml2" prefix is "urn:oasis:names:tc:SAML:2.0:assertion" -->

<saml2:Attribute
   FriendlyName="SLEO"
   Name="gfipm:2.0:user:SwornLawEnforcementOfficerIndicator"
   NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">

   <saml2:AttributeValue
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="xs:string">true</saml2:AttributeValue>

</saml2:Attribute>
```

Figure 13: Example GFIPM Attribute in SAML

```
<!-- The namespace for the "xacml-ctx" prefix is
"urn:oasis:names:tc:xacml:2.0:context:schema:os" -->

<xacml-ctx:Attribute
   AttributeId="gfipm:2.0:user:SwornLawEnforcementOfficerIndicator"
   DataType="http://www.w3.org/2001/XMLSchema#boolean"
   issuer="https://idp.ref.gfipm.net/idp/shibboleth">

   <xacml-ctx:AttributeValue>true</xacml-ctx:AttributeValue>

</xacml-ctx:Attribute>
```

Figure 14: GFIPM Attribute Converted from SAML to XACML

The *Name* XML attribute within the SAML **<saml2:Attribute>** element maps to the *AttributeId* XML attribute with the XACML **<xacml-ctx:Attribute>** element. While a *NameFormat* XML attribute is present in the SAML **<saml2:Attribute>** element, all XACML *AttributeId* XML attributes are of type URI.

In GFIPM Web Services, every SAML user attribute has a datatype of "xs:string" (where the "xs" prefix denotes the XML Schema namespace). XACML uses a different format for datatypes. Also, our implementation sets the datatype for each GFIPM attribute in XACML according to the type of the attribute as defined in the GFIPM Metadata Specification. Table 12 contains a mapping from GFIPM Metadata type to XACML datatype. Note that the GFIPM Metadata type of the Sworn Law Enforcement Officer Indicator attribute is "Boolean".

GFIPM Metadata Type	XACML Datatype
Boolean	http://www.w3.org/2001/XMLSchema#boolean
Date	http://www.w3.org/2001/XMLSchema#date
URI	http://www.w3.org/2001/XMLSchema#anyURI
Base-64 Binary	http://www.w3.org/2001/XMLSchema#base64Binary
[all other types]	http://www.w3.org/2001/XMLSchema#string

Table 12: Mapping from GFIPM Metadata Type to XACML Datatype

Creating the Resource, Action, and Environment attributes

The WSP generates the Resource, Action, and Environment attributes according to Table 13.

Class	Attribute ID	Value
Resource	XACML resource-id	The value of the "ID" parameter of the Read service request.
Resource	GFIPM Criminal History Data Indicator	"true"
Action	GFIPM Action Type	"Read"
Environment	XACML current-dateTime	The date and time at which the service request was received by the WSP implementation.

Table 13: Resource, Action and Environment Attributes used in the Sample Implementation

Finally, the WSP adds the retrieved arrest record to a **<ResourceContent>** element in the Resource *class* of the XACML request.

3.3.4 The WSC / Web Portal and Test Cases

Goals:

1. Understand the functionality of the sample implementation Web Portal / WSC.
2. Manually execute the test cases described in Table 9 on the sample implementation.
3. Understand how to configure an automated testing module.

Summary:

In this Lesson, we will describe the functionality of the Web Portal / WSC, the user interface it provides, and how it uses the WSP. We will describe the necessary steps for manually testing the end-to-end functionality of the sample implementation, using the test cases described in Table 9. Finally, we describe how configure test cases to be executed automatically.

Steps:

3.3.4.1 WSC/Web Portal Implementation

The Web Portal consists of a set of JSP[86] files in the "$WSC_PRJ/src/main/webapp/" directory that provide the web user interface. The interface is supported by Java classes in the org.gtri.icl.iead.policy_guide.portal Java package.

[86] See http://www.oracle.com/technetwork/java/javaee/jsp/index.html.

The WSC functionality is contained in the org.gtri.icl.iead.policy_guide.wsc Java package. It relies on the WSLib module to perform the actual web service communications.

3.3.4.2 Execute the end-to-end test cases

Table 14 contains the test case information from Table 9, except that the Username of the user account is listed instead of the Federation ID. Also, the fourth column has been renamed.

Test Case #	Username	Record Id	Access Control Decision	Obligations
1	XACML Agency-A Test User 1	Record-1	Deny	N/A
2	XACML Agency-A Test User 1	Record-2	Permit	N/A
3	XACML Agency-A Test User 2	Record-2	Permit	N/A
4	XACML Agency-A Test User 3	Record-1	Deny	N/A
5	XACML Agency-B Test User 1	Record-1	Deny	N/A
6	XACML Agency-B Test User 2	Record-2	Permit	LogValidAccess, NotifyViaEmail

Table 14: Sample Implementation Test Cases

There are six test cases. Each test case expresses a user requesting to retrieve an arrest record. For each combination of username and record ID, perform the steps below to execute the test cases.

1. From the virtual machine, or from a computer that can access the virtual machine over a network, open a browser and browse to the Web Portal at $WEBPORTAL-URL. Your browser should be redirected to the GFIPM Reference Federation Discovery Service web page at http://ref.gfipm.net.

2. Select the entry that reads: "(Public) GFIPM 2.0 Reference IDP". Click the Select button.

3. Your browser should be redirected to the GFIPM Metadata 2.0 Reference Identity Provider at https://idp.ref.gfipm.net/idp/. This site allows the user to select a GFIPM Reference Federation user account with which to authenticate. There are different groups of accounts. The accounts in the group labeled: "The following test accounts are for use with the XACML

Training Toolkit" were provisioned for use with this Implementation Guide. From this group, select the appropriate username that corresponds to the current test case. Click Login.

4. Your browser should be redirected back to the sample implementation web portal. Note that there is a link to view the SAML assertion of the authenticated user account. There should be a text box labeled: "Enter a record ID to read". Enter the ID of the arrest record that corresponds to the current test case. Click Submit.

5. The web portal will use the WSC to retrieve the requested arrest record. When this is done, the web portal will display links to the following data:
    a. The requested arrest record if the service request was successful and the service request was permitted by the policy framework.
    b. The XACML request that was created and used by the WSP to obtain the access control decision.
    c. The XACML response generated by the PDP containing the access control decision.

6. Verify that these data items are correct. The arrest record, if it is available, should contain the information as shown in Table 7. The XACML response should contain the decision and obligations as shown in Table 14. Compare the XACML request with the SAML assertion of the authenticated user account; also check the value of the XACML current-dateTime attribute.

3.3.5 Modification Points of the Sample Implementation

Goals:

1. Understand how to make modifications to the sample implementation.

Summary:

In this Lesson, we will describe how to make the following modifications to the sample implementation:

- Editing and adding XACML policies in the WSP.
- Creating a new obligation handler.
- Creating new data records.

Steps:

3.3.5.1 Edited and adding policies

If you change the name of a policy, you'll have to make the appropriate name changes in the $WSP_SX_CONFIG file and the $WSP_RFP_CONFIG file.

If you add a new sub-policy that's referenced by the top-level policy, then you will have to add a new entry in the $WSP_RFP_CONFIG file.

If you edit or add a policy in the WSP, then you will have to re-build and re-deploy the WSP project to use the updated/new policy.

3.3.5.2 Creating obligation handlers

Every obligation handler must be a Java class that extends the org.gtri.icl.iead.policy_guide.obligation.ObligationHandler class that's in the $OBLG module. We recommend you put your new obligation handler in the org.gtri.icl.iead.policy_guide.obligation package in the $WSP_PRJ module (if you don't then you'll have to make sure your obligation handler is on the Java classpath of the WSP).

Also, you must add the appropriate entry in the WSP's obligation handler configuration file ($WSP_OH_CONFIG). This file contains a mapping from obligation ID to obligation handler class name and tells the WSP which obligation handler to use for a given obligation ID.

3.3.5.3 Creating new data records

Every data record must conform to the schema in the $POLICY_GUIDE/arrest_record_iepd/exchangeSchema.xsd file. The ID of the record must be put in the "s:id" XML-attribute of the "ext:Arrest" element. The record must be saved in a file whose name is the record ID appended with the ".xml" extension. The record must be saved in the $WSP_PRJ/src/main/webapp/META-INF/records/ directory and it must have a record ID distinct from any existing records in that directory.

4 Analysis and Discussion of Sample Implementation

This Section provides analysis and discussion about the various architectural and implementation decisions made for the sample implementation developed in Section 3, and how they would likely differ from the decisions made for a production-grade implementation based on the requirements of the XACML Reference Architecture described in Section 2.

As a primer for this discussion, please review Table 15 below. For each component within the XACML Reference Architecture, Table 15 provides the implementation technique used to build it within the sample implementation as well as a list of implementation techniques that could be used to build it within a production system.

Component Name	Implementation Technique in Sample	Likely Implementation Techniques in Production
Requestor	GFIPM Web Services Consumer (WSC)	• GFIPM WSC • Non-GFIPM WSC • Web Browser
PEP	GFIPM Web Services Provider (WSP)	• GFIPM WSP • Non-GFIPM WSP • Web Application
PDP	Library within PEP	• Standalone Web Service • Other Standalone Service
Resource	Flat File(s) on Local File System	• Database Server • Flat File(s) on Local File System • Standalone Web Service • Other Standalone Service
PIP	N/A (Not Part of Sample)	• Standalone Web Service • Other Standalone Service
Supplemental Attribute Authority (SAA)	N/A (Not Part of Sample)	• GFIPM WSP • GSA Backend Attribute Exchange (BAE) WSP • Other WSP
Obligation Handler	Library within PEP	• Standalone Web Service • Other Standalone Service • Library within PEP
Policy Repository	Flat File on Local File System	• Database Server
PAP	N/A (Not Part of Sample) [87]	• Web Application • Standalone Application

Table 15: Implementation Techniques by Component

The remainder of the section contains a brief analysis of selected XACML Reference Architecture requirements from Section 2. For each requirement that we discuss here, we provide: (1) a brief review of the requirement, (2) a brief description of whether and how the requirement is met by the sample implementation, and (3) a brief discussion of how the requirement can or should be met within a production environment.

4.1 Requirement for Secure Communications Channels

[87] Within the sample implementation, the PAP is represented via a text editor that edits a policy file.

In a production environment, all communication channels between components within the architecture must be secured from the fundamental security threats of eavesdropping and tampering. This includes the channels between each of the following pairs of components.

1. Requestor and Policy Enforcement Point (PEP)
2. PEP and Policy Decision Point (PDP)
3. PEP and Resource
4. PDP and Policy Information Point (PIP)
5. PIP and Supplemental Attribute Authorities (SAAs)
6. PEP and Obligation Handlers
7. Policy Administration Point (PAP) and Policy Repository
8. PDP and Policy Repository

Table 16 lists each pair of components. For each pair of components, it denotes the strategy used to secure the channel between the components in the sample implementation, and it also lists one or more strategies that can be used to secure the channel in a production implementation. The list of recommended strategies for a production environment is not exhaustive.

Pair of Components	**Channel Security Strategy in Sample Implementation**	**Recommended Channel Security Strategies in Production**
Requestor and PEP	GFIPM Web Services	• GFIPM Web Services
PEP and PDP	Colocation in Same Process Space	• Enterprise PKI • Private Network • Colocation on Same Host
PEP and Resource	Colocation on Same Host	• Resource-Based Security[88] • Colocation on Same Host
PDP and PIP	N/A (No PIP in Sample)	• Enterprise PKI • Private Network • Colocation on Same Host
PIP and SAAs	N/A (No PIP or SAAs in Sample)	• GFIPM Web Services • GSA Backend Attribute Exchange (BAE)
PEP and Obligation Handlers	Colocation in Same Process Space	• Enterprise PKI • Private Network • Colocation on Same Host
PAP and Policy Repository	N/A (No PAP in Sample)	• Resource-Based Security • Colocation on Same Host

[88] "Resource-Based Security" refers to a security scheme in which the resource itself is a service that supports secure communication with authorized clients. A common example of this type of resource is an Oracle or MySQL database server.

PDP and Policy Repository	Colocation on Same Host	• Resource-Based Security • Colocation on Same Host

Table 16: Communication Channel Security Strategies

Note the recommendation to use GFIPM Web Services for the channels between Requestor and PEP and between PIP and SAAs. We recommend the use of GFIPM Web Services for these channels because it not only provides a robust inter-enterprise trust framework that forms a basis for cryptographically secure communication channels, but also supports federated identity and attribute-based privilege management for both users and non-users, via a well-defined set of attributes. You may choose not to use GFIPM for securing these channels, but if so then you will likely need to solve most or all of the problems for which GFIPM already provides a complete, cohesive solution.

4.2 Requirement for Trusted Requestor Attributes

As noted in Section 2, the PEP must understand and trust the attributes that it receives from the Requestor. The sample implementation satisfies the "trust" requirement by leveraging a trusted 3rd party (the user's identity provider) from the GFIPM Trust Fabric, which is a robust trust framework that is suitable for a production environment. In addition, the sample implementation satisfies the "understand" requirement via the use of attributes, that have syntactically and semantically precise definitions, from (1) the XACML 2.0 Spec, which defines attributes that may generically apply in any access control scenario, and (2) the GFIPM Metadata Spec, which defines attributes that are pertinent within the law enforcement community. The sample implementation therefore satisfies the PEP's requirement for trusted Requestor attributes at a level that is suitable for production. You may chose not to leverage these GFIPM work products in a production implementation, but if so then you need to find an alternate means of satisfying this requirement.

4.3 Requirement for a Common Interface Between Requestor and PEP

The API exposed by a PEP typically needs to include low-level communication protocol standards (e.g. TLS, HTTP, and SOAP) as well as application-level service descriptions and data formats (e.g. WSDL, NIEM IEPDs, etc.) The sample implementation meets these needs via the GFIPM Web Services spec, an application-specific service specification defined via WSDL, and a sample NIEM IEPD to define the data payload. The sample IEPD used within the sample implementation is not an actual IEPD, but it is similar in structure and format to an actual IEPD. A typical production system would use GFIPM Web Services or a similar protocol stack. The service interface and data payload formats would of course depend on the specific nature of the application service exposed by the PEP.

4.4 Requirement for Translation from Application Environment to XACML

The translation process from the application environment to XACML is essentially a process of mapping attributes from the application's domain language to the appropriate XACML attributes. This mapping process differs slightly for each type of XACML attribute. For XACML subject attributes, the sample implementation harvests the necessary attribute data from the SAML assertion it receives from the Requestor. For XACML action attributes, the PEP typically generates the attribute values based on the type of request made by the Requestor (e.g. read, write, etc.) and various metadata about that request (e.g. IP address, geo-location of requestor, etc.) For both subject attributes and action attributes, a typical production implementation would work in a manner very similar to the sample implementation. XACML resource attributes depend heavily on the properties of the resource protected by the PEP. The sample implementation uses a relatively small, simple set of resource attributes. In a production environment, the set of resource attributes would typically be larger, but similar in style to those used by the sample implementation. XACML environment attributes can be retrieved either from trusted sources within the local host system (e.g. date and time) or from trusted external attribute services (e.g. weather conditions). The sample implementation retrieves the value of a date-time environment attribute from the local host system. In a production system, the implementation of environment attribute collection would depend on the nature of available attribute sources, from either the local host system or other trusted data sources via a network.

4.5 Requirement for an Accurate and Efficient Attribute Retrieval Algorithm

As stated in Section 2, the PIP must implement an accurate and efficient attribute retrieval algorithm, so it can determine which SAA to contact for any given attribute, and dispatch the attribute request appropriately. Retrieval of supplemental attributes by a PIP is outside the scope of the sample implementation. This is a very broad and complex topic due to the trust implications of retrieving attributes from sources that are outside the trust perimeter of the enterprise. Implementation of attribute retrieval in a production environment is therefore highly dependent on the surrounding context, including security, trust, and availability of supplemental attribute sources. BAE and GFIPM are both good starting points for a SAA framework. Any further discussion of this topic is outside the scope of this guide.

4.6 Requirement for Proper Resolution of Entities Specified in Obligations

Section 2 notes that an Obligation Handler must be able to accurately resolve the identity and location of entities referenced within obligations, based on the context provided to it by the PEP. For example, if an obligation says: "The owner of a data

resource must be notified via email upon every access to that resource", then the Obligation Handler must be able to resolve the email address of the owner of the data resource.

In the sample implementation, entity resolution is handled in the obligation specifications in the policy. The arguments of the obligations contain the XACML elements necessary to properly resolve the data needed by the Obligation Handlers.

The approach taken in the sample implementation is just one paradigm for achieving proper entity resolution. This approach should be suitable for many production environments in which the resolution details for each obligation are known by the policy author at the time the policy is created. There are no other well-known approaches for meeting this requirement. However, this problem is currently being explored by the Global Federated Identity and Technical Privacy Task Team.

4.7 Requirement for a Processing Model to Handle "Out-of-Band" Obligations

As discussed in Section 2, there exist certain "out-of-band" obligations that an entity other than the PEP is required to fulfill. An example of an "out-of-band" obligation is: "The data requestor must not further disseminate the data upon receiving it".

The sample implementation does not contain any out-of-band obligations. There are currently no well-known approaches for processing out-of-band obligations, but this issue is under investigation by the Global Federated Identity and Technical Privacy Task Team.

5 Further Reading

This section provides a list of resources for further reading about topics covered in this guide as well as other related topics.

Technology Standards and Paradigms

eXtensible Access Control Markup Language (XACML)
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

Security Assertion Markup Language (SAML)
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Global Justice Information Sharing Initiative
http://www.it.ojp.gov/global

Global Federated Identity and Privilege Management (GFIPM)
http://gfipm.net/

SOAP
http://www.w3.org/TR/soap/

REST
http://en.wikipedia.org/wiki/Representational_state_transfer

XML
http://www.w3.org/XML/

JAX-WS
http://jcp.org/en/jsr/detail?id=224

Open Source and Vendor Software Implementations

SunXACML
http://sunxacml.sourceforge.net/

Axiomatics
http://www.axiomatics.com/

BiTKOO
http://www.bitkoo.com/

Jericho Systems
http://www.jerichosystems.com/

Holistic Enterprise-Ready Application Security Architecture Framework (HERAS-AF)
http://www.herasaf.org/

Java
www.java.com/

Glassfish
http://jax-ws.java.net/

Metro
http://metro.java.net/

Apache Maven
http://maven.apache.org/

Appendix A: Common Tasks

Executing SimplePDP

Prerequisites:

1. Install the Dependency Libraries.

Steps:
Invoke Java, with the following arguments:

- The classpath needs to include the following:
    - the SunXACML library (see Installing the Dependency Libraries)
- The main class is "com.sun.xacml.support.SimplePDP"
- The arguments to the main class are

"<path/to/request-file> <path/to/policy-file>" replacing "<path/to/request-file>" with the path to the file that contains the XACML request and replacing "<path/to/policy-file> with the path to the file that contains the XACML policy.

The following steps explain how to do this on the virtual machine:

1. Open a terminal.
2. Enter the following command.

"java –cp /home/guide/policy_guide/sample_impl/sunxacml/sunxacml.jar com.sun.xacml.support.SimplePDP <path/to/request-file> <path/to/policy-file>"

Replace "<path/to/request-file>" with the path to the file that contains the XACML request and replacing "<path/to/policy-file> with the path to the file that contains the XACML policy. Be sure to not hit the enter key until the entire command is typed into the terminal.

If you want to output the XACML response to a file, then add the following to the end of the above command (be sure to first type a space):
" > <path/to/response-file>" replacing "<path/to/response-file>" with the path to the file that will contain the XACML response.

While you are going through Lesson Group 3.1, we suggest you enter this command from the directory that corresponds to the particular lesson in which you are currently working.

The end

Executing SimplePDP (with a Configuration File)

Prerequisites:

1. Install the Dependency Libraries.

Steps:
Invoke Java, with the following arguments:

- The classpath needs to include the following:
    - the SunXACML library (see Installing the Dependency Libraries)
    - the SunXACML module library (see Installing the Dependency Libraries)
    - the Java EE 6 library (this is needed by the Policy Reference Module and gets installed automatically when the Dependency Libraries are installed)
- The "com.sun.xacml.SunXACMLConfigFile" environment variable needs to be set to the path to the SunXACML configuration file.
- The main class is "com.sun.xacml.support.SimplePDP"
- The arguments to the main class are "-config <path/to/request-file"> replacing "<path/to/request-file>" with the path to the XACML request file you want to evaluate.

The following steps explain how to do this on the virtual machine:

3. Open a terminal.
4. Enter the following command.

"java –cp
/home/guide/policy_guide/sample_impl/sunxacml/sunxacml.jar:/home/guide/policy_guide/sample_impl/ImplGuideSunXACML/target/PolicyGuide-SunXACML-Modules-1.0-SNAPSHOT.jar:/home/guide/.m2/repository/javax/javaee-web-api/6.0-RC2/javaee-web-api-6.0-RC2.jar –
Dcom.sun.xacml.SunXACMLConfigFile=<path/to/SunXACML-config-file>
com.sun.xacml.support.SimplePDP –config <path/to/request-file>"

Replace "<path/to/SunXACML-config-file>" with the path to the SunXACML configuration file. Replace "<path/to/request-file>" with the path to the file that contains the XACML request you want to evaluate. Be sure to not hit the enter key until the entire command is typed into the terminal.

If you want to output the XACML response to a file, then add the following to the end of the above command (be sure to first type a space):
" > <path/to/response-file>" replacing "<path/to/response-file>" with the path to the file that will contain the XACML response.

We suggest you enter this command from the "/home/guide/policy_guide/" directory.

The end

Installing the Dependency Libraries

Prerequisites:

1. Download and extract the accompanying file set. The virtual machine already has this file set extracted at the "/home/guide/policy_guide/" directory.
2. Note that this has already been done on the distributed virtual machine.

Steps:

1. Open a terminal.
2. Go to the "$SAMPLE_IMPL/" directory.

3. Install the SunXACML library; do the following:
    a. Go to the "$SAMPLE_IMPL/sunxacml/" directory.
    b. Enter

"mvn install:install-file –Dfile=sunxacml.jar –DpomFile=pom.xml"

4. Install the wslib library; do the following:
    a. Go to the "$SAMPLE_IMPL/wslib/" directory.
    b. Enter

"mvn install:install-file –Dfile=wslib.jar –DpomFile=pom.xml"

5. Install the SunXACML modules; do the following:
    a. Go to the "$SAMPLE_IMPL/ImplGuideSunXACML/" directory.
    b. Enter "mvn clean install"

6. Install the obligations module; do the following:
    a. Go to the "$SAMPLE_IMPL/obligation" directory.
    b. Enter "mvn clean install"

7. (Re-)Deploy the STS/ADS module; do the following:
    a. Make sure "domain1" is running in Glassfish.
    b. Open a web browser on the local machine
    c. Browse to "http://localhost:4848"
    d. On the left, vertical menu, click "Applications".
    e. If "m2sts" is listed, then click the checkbox next to "m2sts" and then click "Undeploy".
    f. Click "Deploy".
    g. Under "Packaged File to Be Uploaded to the Server", click "Browse".
    h. Go to the "$SAMPLE_IMPL/" directory.
    i. Select the "m2sts.war" file and click "Open".
    j. Click "OK".

The end

Building the WSC/Web Portal

Prerequisites:

1. Install the dependency libraries.

Steps:
1. Open a terminal.
2. Go to the "$SAMPLE_IMPL/wsc" directory.
3. Enter "mvn clean package"

The end

Building the WSP

Prerequisites:
1. Install the dependency libraries.

Steps:
1. Open a terminal.
2. Go to the "$SAMPLE_IMPL/wsp" directory.
3. Enter "mvn clean package"

The end

(Re-)Deploying the WSC/Web Portal in the virtual machine

Prerequisites:
1. Build the WSC/Web Portal.
2. Make sure Apache is running.

Steps:
1. Copy the "$SAMPLE_IMPL/wsc/target/m2wsc.war" file to the "/opt/tomcat/webapps/" directory (replacing the existing m2wsc.war file if it's already in "/opt/tomcat/webapps/").
    a. This needs to be done as root.

To do this you can run the following command:
"sudo cp /home/guide/policy_guide/sample_impl/wsc/target/m2wsc.war /opt/tomcat/webapps/".

2. Restart Apache.

The end

(Re-)Deploying the WSP in the virtual machine

Prerequisites:

1. Build the WSP.
2. Make sure "domain1" is running in Glassfish.

Steps:

1. Open a web browser on the local machine.
2. Browse to "http://localhost:4848".
3. On the left, vertical menu, click "Applications".
4. If "m2wsp" is listed, then click the checkbox next to "m2wsp" and then click "Undeploy".
5. Click "Deploy".
6. Under "Packaged File to Be Uploaded to the Server", click "Browse".
7. Go to the "$SAMPLE_IMPL/wsp/target" directory.
8. Select the "m2wsp.war" file and click "Open".
9. Click "OK".

The end

Restarting the "domain1" in Glassfish in the virtual machine

Notes:
In the virtual machine, Glassfish and domain1 are configured to start automatically on boot.

Steps:

1. Open a terminal.
2. Go to the "/opt/glassfish3/glassfish/bin/" directory.
3. Enter "sudo asadmin stop-domain domain1"
4. Wait for the command to complete.
5. Enter "sudo asadmin start-domain domain1"

The end

Restarting Apache in the virtual machine

Open a terminal and enter: "sudo /etc/init.d/httpd restart"
The end

Appendix B: Labels

Label	Definition	Description
$POLICY_GUIDE	/home/guide/policy_guide	Base directory containing all files associated with this Guide
$SAMPLE_IMPL	$POLICY_GUIDE/sample_impl	Base directory containing all sample implementation files
$WSP_PRJ	$SAMPLE_IMPL/wsp	The WSP module
$WSC_PRJ	$ SAMPLE_IMPL /wsc	The WSC module
$OBLG_PRJ	$ SAMPLE_IMPL /obligation	The Obligations module
$SX_MOD_PRJ	$ SAMPLE_IMPL /ImplGuideSunXACML	The SunXACML Modules module
$WEBPORTAL_URL	http://sp.example.org/m2wsc/index.jsp	The URL to access the sample application
$WSP_SX_CONFIG	$WSP_PRJ/src/main/webapp/META-INF/policies/SunXACMLConfig.xml	The WSP's SunXACML configuration file
$WSP_RPF_CONFIG	$WSP_PRJ/src/main/webapp/META-INF/policies/ReferencePolicyFinderConfig.xml	The WSP's Reference Policy Finder configuration file
$WSP_OH_CONFIG	$WSP_PRJ/src/main/webapp/META-INF/ObligationHandlerConfig.xml	The WSP's Obligation Handler configuration file

Appendix C: XACML Reference Tables

Predicate value(s)	Resulting Instance value
All True	Match
No False and at least one Indeterminate	Indeterminate
At least one False	No-Match

Table 17: Instance Evaluation Table

Instance value(s)	Resulting Class value
At least one Match	Match
No Matches and at least one Indeterminate	Indeterminate
All No-Match	No-Match

Table 18: Class Evaluation Table

Subjects Value	Resources Value	Actions Value	Environments Value	Resulting Target Value
Match	Match	Match	Match	Match
No-Match	Match or No-Match	Match or No-Match	Match or No-Match	No-Match
Match or No-Match	No-Match	Match or No-Match	Match or No-Match	No-Match
Match or No-Match	Match or No-Match	No-Match	Match or No-Match	No-Match
Match or No-Match	Match or No-Match	Match or No-Match	No-Match	No-Match
Indeterminate	Don't Care	Don't Care	Don't Care	Indeterminate
Don't Care	Indeterminate	Don't Care	Don't Care	Indeterminate
Don't Care	Don't Care	Indeterminate	Don't Care	Indeterminate
Don't Care	Don't Care	Don't Care	Indeterminate	Indeterminate

Table 19: Target Evaluation Table

Target Value	Condition Value	Resulting Rule Value
Match	True	Effect
Match	False	NotApplicable
Match	Indeterminate	Indeterminate
No-Match	Don't Care	NotApplicable
Indeterminate	Don't Care	Indeterminate

Table 20: Rule Evaluation Table

Target Value	Rule Values	Resulting Policy Value
Match	At Least One Effect	Specified by the Rule-Combining Alg
Match	All NotApplicable	NotApplicable
Match	At Least One Indeterminate	Specified by the Rule-Combining Alg
No-Match	Don't Care	NotApplicable
Indeterminate	Don't Care	Indeterminate

Table 21: Policy Evaluation Table

Target Value	Policy Values	Resulting Policy Set Value
Match	At Least One Permit/Deny	Specified by the Rule-Combining Alg
Match	All NotApplicable	NotApplicable
Match	At Least One Indeterminate	Specified by the Rule-Combining Alg
No-Match	Don't Care	NotApplicable
Indeterminate	Don't Care	Indeterminate

Table 22: Policy Set Evaluation Table

Appendix D: Virtual Machine Details and Installed Software

VMWare virtual machine:

- 1 GB RAM
- 8 GB hard disk
- 1 CPU
- NAT networking
- User account:
    - username: guide
    - password: gtrincscglobal
- root password: gtrincscglobal

Software:

- CentOS v5
- Apache v2
- Tomcat v6
- Maven v2
- Java v6
- Glassfish v3