Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare

Project Acronym: **ASCLEPIOS**
Project Contract Number: **826093**

Programme**: Health, demographic change and wellbeing**
Call: **Trusted digital solutions and Cybersecurity in Health and Care to protect privacy/data/infrastructures**
Call Identifier: **H2020-SC1-FA-DTS-2018-2020**

Focus Area: **Boosting the effectiveness of the Security Union**
Topic**: Toolkit for assessing and reducing cyber risks in hospitals and care centres**
Topic Identifier: **H2020-SC1-U-TDS-02-2018**

Funding Scheme: **Research and Innovation Action**

Start date of project: 01/12/2018 Duration: 36 months

Deliverable:
D3.3 Context-aware ABAC Enforcement Mechanism

Due date of deliverable: 31/05/2020 Actual submission date: 16/06/2020

WPL: ICCS

Dissemination Level: Public

Version: final

ASCLEPIOS is supported by the H2020 Programme under contract no. 826093

Table of Contents

List of Figures and Tables

Status, Change History and Glossary

Status:	**Name:**	**Date:**	**Signature:**
Draft:	Panagiotis Gouvas	27/05/2020	
Reviewed:	Evmorfia Biliri	05/06/2020	
Approved:	Tamas Kiss	16/06/2020	

Table 1: Status Change History

Version	Date	Pages	Author	Modification
V0.1	03/04/2020	5	Giannis Ledakis (UBI)	TOC
V0.1	22/04/2020	20	Panagiotis Gouvas (UBI)	Sections 2, 3
V0.3	05/05/2020	29	Panagiotis Gouvas (UBI)	Section 4, updates on section 2,3
V0.4	14/05/2020	31	Yiannis Verginadis(ICCS)	Authorization flow with ABE and ABAC
V0.5	19/05/2020	33	Panagiotis Gouvas (UBI)	Section 5 complete
V0.6	23/05/2020	38	Ioannis Patiniotakis (ICCS)	Added policy validation part
V0.7	25/05/2020	40	Panagiotis Gouvas (UBI)	Updates on all sections, Intro, Conclusions
V0.8	27/05/2020	41	Panagiotis Gouvas, Giannis Ledakis	Ready for review
V0.9	30/05/2020	41	Yiannis Verginadis(ICCS)	Review
V0.9.1	03/05/2020	44	Panagiotis Gouvas (UBI), Giannis Ledakis (UBI)	Updates based on ICCS review
V0.9.2	05/05/2020	44	Evmorfia Biliri (SUITE5)	Review
V0.1	16/05/2020	44	Giannis Ledakis (UBI)	Final for Submission

Table 2: Deliverable Change History

Glossary

ABAC	Attribute Based Access Control
ABE	Attribute Based Encryption
ACL	Access Control List
ACM	Access Control Mechanism
AMPLE	ASCLEPIOS Models and PoLicies Editors
BPMN	Business Process Model and Notation
CASM	Context-Aware Security Model
DAC	Discretional Access Control
IBAC	Identity Based Access Control
JWT	JSON Web Token
MAC	Mandatory Access Control
NGAC	Next Generation Access Contro
OAUTH	Open Authorization
OIDC	Open Id Connect
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role Based Authorization & Control
SOTA	State of the Art
XACML	eXtensible Access Control Markup Language
XSD	XML Schema Definition

Table 3: Glossary

Executive Summary

The need of a trusted environment in which only authorized users are permitted to access data is of imperative importance and is a blocking factor for adopting cloud systems when sensitive data are treated, as the case of the health care domain. Our goal is to enhance the access control mechanisms that can be used in the healthcare. WP3 is addressing this exact topic by defining and evaluating contextual information (e.g., the identity of a user, its role, patterns of access, connection type, etc.) and attributes that characterize sensitivity levels of data. Based on the work reported on this deliverable, access control policies will be enforced as part of two different authorisation paradigms; i) the Attribute Based Access Control (ABAC layer which permits or denies access and/or editing rights to (encrypted) EHRs; and ii) the Attribute-Based Encryption (ABE layer which handles the way sensitive data should be decrypted. The decision of adopting the model of ABAC for medical data is elaborated along with its architecture implications. Furthermore, although ABAC is a pure authorization scheme, the synergy that ABAC has with ABE is discussed since ABE entails some characteristics of authorization without being an authorization framework per se.

The Balana ABAC engine is a cornerstone of ASCLEPIOS Context-Aware Authorization Engine presented in this deliverable. However, ASCLEPIOS Context-Aware Authorization Engine is using a combination of ABAC with Attribute-Based-Encryption to augment the authorization functionality in a distributed cloud environment. These two schemes are complemented by an identity management scheme that abstracts the extraction of authentication info. Hence the Context-Aware Authorization Engine is efficiently combining OpenIDConnect signalling for Identity Extraction (user authentication), ABAC Policy Enforcement for accessing allowing/disallowing access to an ABE Server that issues attribute-based encryption/decryption keys.

1 Introduction

WP3 focuses on defining and evaluating contextual information (e.g., the identity of a user, its role, patterns of access, connection type, etc.) and attributes that characterize sensitivity levels of data. This information is considered in ASCLEPIOS before granting any data access request. Based on such information, the contextual model was developed in D3.1[1], and building on it a number of enforcement rules can be created as the most elementary structural elements of policies. Indicatively, attributes that are organized in a hierarchical structure may include concepts related to: i) the device from which there is an access attempt, ii) the actor that tries to access the data (e.g. location, IP, role in the healthcare use case, etc.) and iii) historical data that reveal patterns of access (e.g. frequency, usual dates or hours of access, the usual duration of access, previously accessed data, etc.). Such concepts, along with a number of properties that interrelate them, serve as background knowledge for the ASCLEPIOS access control policies. These access control policies are then enforced as part of two different authorization paradigms that are employed in ASCLEPIOS in sequence for achieving even higher levels of security controls. These paradigms are the Attribute-based Access Control (ABAC) and the Attribute-based Encryption (ABE).
This deliverable reports on the theoretical aspects of the adopted Access Control Mechanism that has been employed. The reader is introduced in the theoretical background of different Access Control models. The decision of adopting the model of ABAC for medical data is elaborated along with its architecture implications. Furthermore, although ABAC is a pure authorization scheme, the synergy that ABAC has with ABE is discussed since ABE entails some characteristics of authorization without being an authorization framework per se. However, this synergy has to be fully clarified because it is the cornerstone of the advanced security characteristics that are offered by ASCLEPIOS.

1.1 Objectives

The primary objectives of this deliverable are to:

1. Elaborate on the various Authorization schemes
2. Justify the decision to follow ABAC as a basic means of implementing access control
3. Elaborate on the decision to adopt XACML as formal language for ABAC policies
4. Discuss the synergy of ABAC and ABE in the frame of ASCLEPIOS

1.2 Relationship to ASCLEPIOS Deliverables

The deliverable documents the Context-Aware Authorization Engine that has been developed in the frame of work package 3. Therefore, it builds on deliverables D3.1[1] and D3.2[2], as depicted in Figure 1.

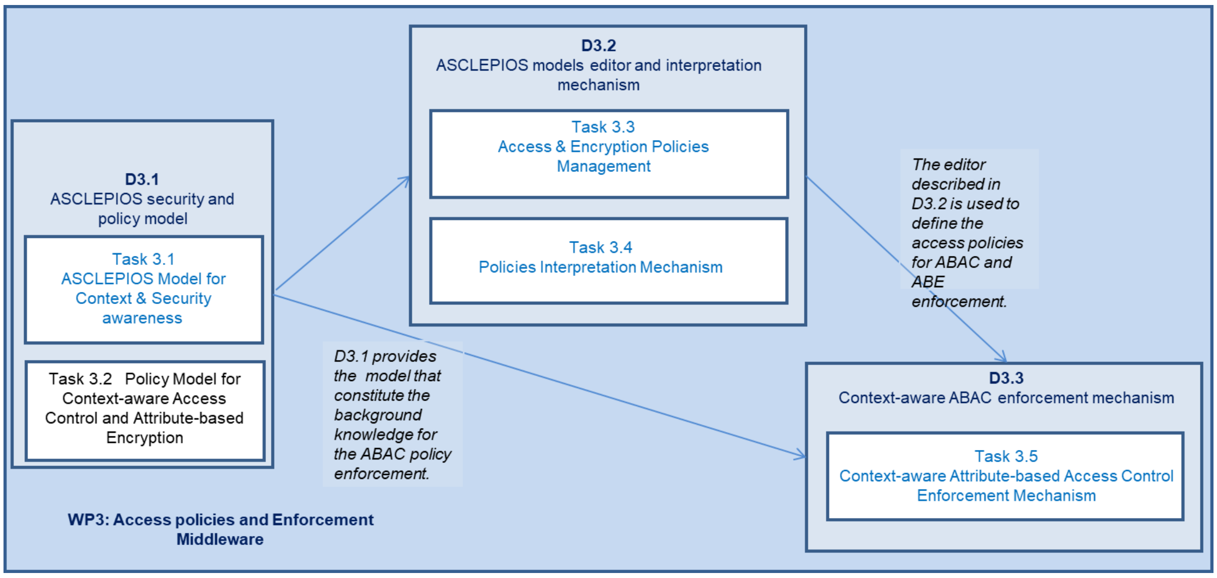

D3.2
ASCLEPIOS models editor and interpretation mechanism
Task 3.3
Access & Encryption Policies Management
Task 3.4
Policies Interpretation Mechanism
D3.1
ASCLEPIOS security and policy model
Task 3.1
ASCLEPIOS Model for Context & Security awareness
Task 3.2 Policy Model for Context-aware Access Control and Attribute-based Encryption
The editor described in D3.2 is used to define the access policies for ABAC and ABE enforcement.
D3.1 provides the model that constitute the background knowledge for the ABAC policy enforcement.
D3.3
Context-aware ABAC enforcement mechanism
Task 3.5
Context-aware Attribute-based Access Control Enforcement Mechanism
WP3: Access policies and Enforcement Middleware

Figure 1 Relationship of D3.3 to deliverables of WP3

1.3 Organization

Chapter 2 provides a formal definition of what is an authorization scheme. It also shortly elaborates on the different schemes that have been proposed historically. Chapter 3 elaborates on ABAC that is the preferred scheme. The two most prominent implementations of ABAC are discussed. The selection of XACML is justified.

Chapter 4 provides an overview of the existing software artefacts that could be used as a basis for implementing the ASCLEPIOS authorization engine. A short description of the most prominent open-source implementations that are provided along with their traits. Chapter 5 presents the actual mechanism by analysing the synergy among ABAC and ABE, providing the overview of the authorization workflow and elaborating on the developed extensions of the AMPLE editor (presented in D3.2) towards policies validation and security awareness.Chapter 6 concludes this deliverable.

2 Relevant SOTA in authorization for medical data access

2.1 Authorization Background – Theoretical Concepts

ASCLEPIOS framework is a security and privacy-by-design framework for accessing medical data. To this end, one of the core aspects that such a framework should deal with is logical access control to generated data. It should be clarified that authorization **should not be confused with authentication**. **Authentication** deals with the problem of proving the **identity** of a natural person or a system entity that aims to interact with a resource while **authorization** deals with the problem of providing **logical access control** to these resources. The logical access control is always bound **to specific actions** that the natural-person/systemic entity (hereinafter **subject**) wishes to perform i.e. discovering, reading, creating, editing, deleting, and executing.

These resources belong administratively to organizational entities that are responsible for providing **proper policies** that dictate the **allowance or disallowance** of an action as mentioned above. If the subject that requires to perform an operation with the resource satisfies the **authorization policy** established by the resource owner, then the subject is authorized to perform the desired operation on that resource.

Taking into consideration all the above, a set of questions are raised. These are:

- Is there a **universal way** of describing the aforementioned policies that dictates allowance/disallowance?
- If not such a universal way exists, **how allowance/disallowance works** in different systems?

In order to answer these questions, we have to trace back to the theoretical concepts of authorization. The “problem statement” of an authorization request entails some **concepts that are common** in all approaches. These include: the **subject** (a.k.a. requestor), the **object** (a.k.a. resource), the **action** (to the resource), the **environmental context**, the **defined policy (or policies)** and the **policy-evaluation business logic (a.k.a. Policy Enforcement)**. Unfortunately, **there is no universal authorization system** that covers modelling-wise and implementation-wise all functional requirements of all systems. Such functional requirements may be contradictive e.g.:

- Dynamic authoring of policies by several actors simultaneously (versus static definition of policies)
- Separation of enforcement business logic with the policy-authoring environment
- Dynamic expansion of policies during the operation of the authorization system without disruption of the resource

As a result, it should be clarified that, in absence of a one-size fits all solution, there are **many proposals that attempt to provide “holistic” solutions**. “Holistic” refers to the fact that they **try to cover** many corner-cases; yet they compromise one feature in favor of another based on their area of applicability. This is justifiable by the fact that some functional and non-functional requirements are more crucial than others in specific systems. For example, **the decision of allowing or disallowing a system-call of a process to the Linux kernel has completely different requirements from a decision of allowing or disallowing a doctor to access a medical record on a specific system**.

Practically, when we refer to an Access Control Mechanism (a.k.a. ACM) we refer to **three distinct aspects**:

- The way the concepts subject, object, action, environmental context and authorization policy are **modelled and serialized**
- The way the **policy evaluation** of allowance/disallowance works (i.e. the signaling that has to be performed)
- The **functional and the non-functional requirements** that are promoted by the specific ACM (i.e. central access control entity).

Throughout the last 30 years, several Access Control Mechanisms have been proposed, with each having its advantages and limitations. However, for the sake of completeness, the four more dominant ACMs will be listed:

- **MAC: Mandatory Access Control** refers to a type of access control by which a central entity (e.g. an operating system kernel) constrains the ability of a subject or initiator to access or generally perform some sort of operation on an object or target by referring to the object per se (directly).
- **DAC:** In contrast to MAC, **Discretionary Access Control** (DAC) **allows users to make policy decisions and/or assign security attributes**. An example of DAC is the traditional Unix system with the users, groups, and read-write-execute permissions. MAC-enabled systems allow policy administrators to implement organization-wide security policies that users cannot override or modify. DAC allows security administrators to define central policies to be enforced for all users
- **IBAC/ACLs**: **Identity Based Access Control** (IBAC) employs mechanisms such as access control lists (ACLs) to capture the identities of those allowed to access the object. If a subject presents a credential that matches the one held in the ACL, the subject is given access to the object. Individual **privileges** of the subject to perform operations (read, write, edit, delete, etc.) **are managed on an individual basis by the object owner**. Each object needs to have ACL and set of privileges assigned to each subject. As result, in IBAC the authorization decisions are made prior to any specific access request; for each subject to be placed on an ACL, the object owner must evaluate identity, object, and context attributes against policy governing the object and decide whether to add the subject to the ACL. As this decision is static a notification process is required for the owner to reevaluate and perhaps remove a subject from the ACL to represent subject, object, or contextual changes. Failure to remove or revoke access over time leads to users accumulating privileges.
- **RBAC (Role Based Authorization & Control)** employs **pre-defined roles** that carry a specific **set of privileges associated with them** and to which **subjects are assigned**[5]. In RBAC, access is predetermined by the person assigning the roles to each individual and explicitly by the object owner when determining the privilege associated with each role. At the point of an access request, **the access control mechanism evaluates the role assigned to the subject requesting access and the set of operations this role is authorized to perform** on the object before rendering and enforcing an access decision.
- **ABAC (Attribute-Based Access Control)** uses attributes, and policies that express boolean rule sets that can **evaluate** many different **attributes before allowing access**. ABAC, therefore, avoids the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made. IBAC and RBAC can be seen as special cases of ABAC, with IBAC using the attribute of “identity” and RBAC using the attribute of “role”

As already mentioned, different ACMs are accompanied by different architectural merits and as such a crucial question looms: **which is the best-fit ACM for ASCLEPIOS**?

2.2 ABAC as a best-of-breed Access Control Mechanism

In order to identify the best-fit ACM we must trace back to the functional and non-functional requirements of ASCLEPIOS, defined in deliverable D1.2[4]. In medical systems, the most crucial requirements are **dynamicity in policy creation** and **separation of concerns** among **policy definition** and policy enforcement. There requirements imply that an access policy can be defined by more than one policy-makers (e.g. a patient, a doctor). Such policies **shall be able to update dynamically based** on specific constraints that are **not a-priori related to requestors** (e.g. i share my medical record to all doctors of hospital or to all doctors of hospital X that belong to department Y).

"**Dynamicity**" is the key factor that drives our decision. More specifically, a Mandatory Access Control scheme is **inappropriate** since the policy definition point is unique and the **resources have static properties** based on which allowance and disallowance is provided. In an analogous manner DAC schemes are not capable to fit in our requirements. Although they allow the objects to define their own values for the policy properties; **the properties per se are totally static** (i.e. they cannot be extended).

Furthermore, IBAC and Access Control Lists suffer from a severe drawback. Although they support extensible attributes for subjects & objects the exhaustive list of **authorization decisions** should be defined **prior to any request**. Hence, we pay a **huge penalty for the offered dynamicity**.

On the other hand, RBAC and ABAC can satisfy our needs. As already discussed, RBAC is a specialization of ABAC (according to which one subject attribute is addressed as Role) and as such we will adopt the generalized ABAC.

In general, using ABAC removes the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made[5]. Instead, when a subject requests access, an ABAC-compliant engine makes an access control decision based on a) the assigned attributes of the requester, b) the assigned attributes of the object, c) the environment conditions, and d) a set of policies that are specified in terms of those attributes and conditions.

Furthermore, policies in ABAC are created and managed **without direct reference to users and objects**, while also users and objects can be provisioned without reference to policy. For the interest of ASCLEPIOS Framework this separation of policies forom users and objects is important as it allows the support of advanced real life scenarios (e.g. the emergency access to a patient's data). In contrast, in any non-ABAC multi-organizational access method example, the access to an object (e.g. patient's medical data) outside of the subject's originating organization would require the **subject's identity to be pre-provisioned** in the target organization and prepopulated on an access list.

For the sake of clarity, we will **differentiate** hereinafter in the deliverable the concepts of **ABAC** and **ABAC Reference Implementations**. Following the strict definition of NIST[5] we will refer to ABAC as the *"access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions"*. Following this definition, the terms "attribute", "subject", "object", "operation", "policy" and "environmental condition" will be unambiguously used based on the following definitions:

- **Attributes** are characteristics of the subject, object, or environment conditions. Attributes contain information given by a name-value pair.
- A **Subject** is a human user or a systemic entity, such as a device that issues access requests to perform operations on objects. Subjects are assigned one or more attributes.
- An **Object** is a system resource for which access is managed by the ABAC system, such as devices, files, records, tables, processes, programs, networks, or domains containing or receiving information. It can be the resource or requested entity, as well as anything upon which an operation may be performed by a subject including data, applications, services, devices, and networks.
- An **Operation** is the execution of a function at the request of a subject upon an object. Operations include read, write, edit, delete, copy, execute, and modify.

- **Policy** is the representation of rules or relationships that makes it possible to determine if a requested access should be allowed, given the values of the attributes of the subject, object, and possibly environment conditions.
- **Environment conditions** represent operational or situational context in which access requests occur. Environment conditions are detectable environmental characteristics. Environment characteristics are independent of subject or object, and may include the current time, day of the week, location of a user, or the current threat level.

Any ABAC system should implement the conceptual flow that is depicted on Figure 2. According to this flow any subject can perform an access request for a specific operation regarding a specific medical record (step 1).

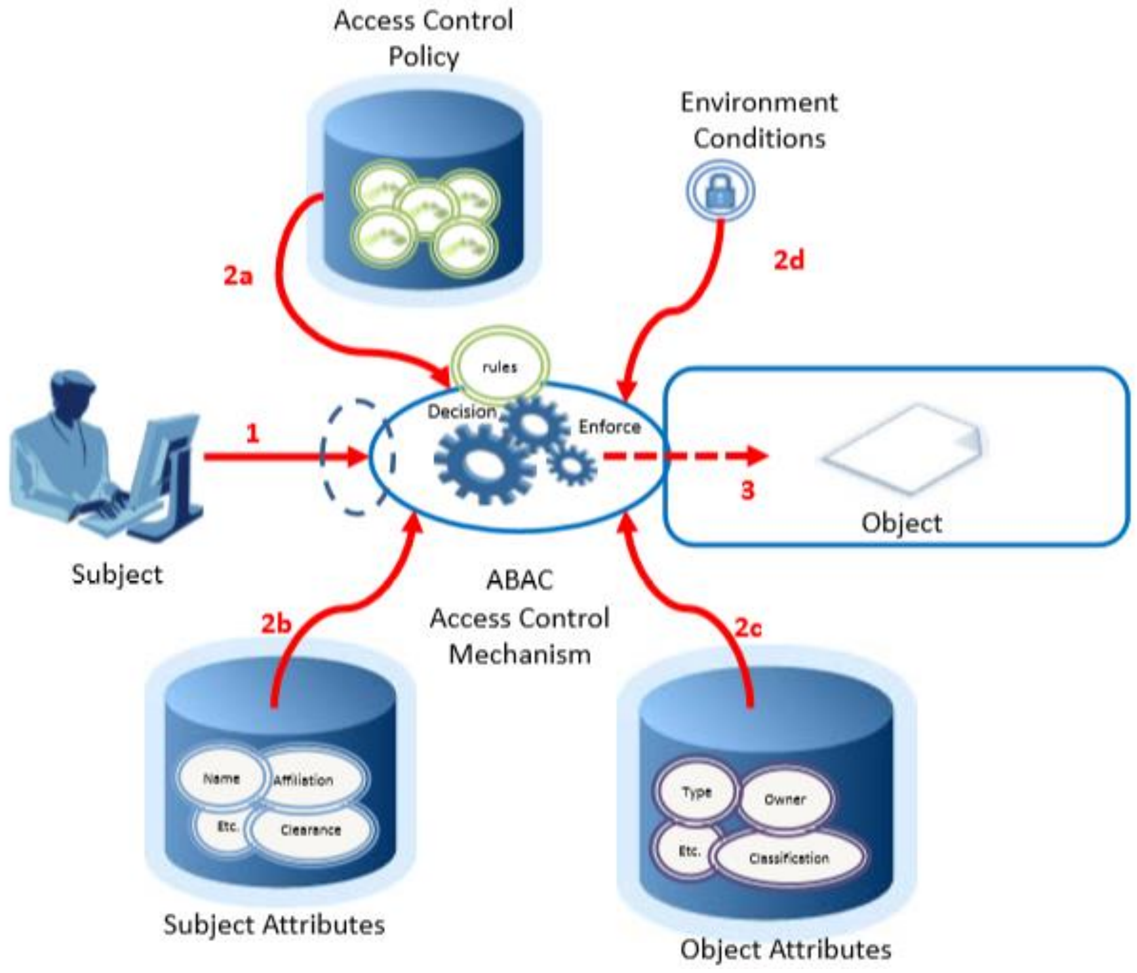


Figure 2 ABAC Indicative Information Flow

The access request is handled by the ABAC reference implementation engine, which consults a policy repository (step 2a) in order to come to the set of attributes that have to be examined in order to reach a decision of “allow” or “deny”. The attribute examination phase checks subject attributes (step 2b), object attributes (step 2c) and environmental attributes (step 2d) in order to perform the actual assessment (step 3).
As a result, ABAC is our final decision. However, it should be clarified that ABAC is a theoretical framework not a standard. In the next section we will elaborate on which standard will be used for the ASCLEPIOS Authorization engine.

3 ABAC Reference Implementations

As already discussed, there are many reference implementations of the ABAC model. One example of an access control framework that is consistent with ABAC is the eXtensible Access Control Markup Language (XACML) [6]. Another example is the Next Generation Access Control (a.k.a. NGAC) standard [7]. These two are considered to be the most notable ones. The in-detail comparison between the two reference implementations super-exceeds the scope of this deliverable. However, we will provide a brief overview of the two proposals in order to justify our selection.

3.1 XACML in a Nutshell

XACML is an OASIS [8] standard that describes both a policy language and an access control decision request/response language. Both languages use XSD [9] notations; hence policy definition and request/response elements are serialized as XML elements. The policy language details the general access control requirements and provides extension points for new functions, data types, combining logic, etc. In the other hand, the request/response language allows forming a query that asks whether a given action should be allowed. The response shall include an answer on the request allowance using one of the fololwing four values:

- Permit
- Deny
- Indeterminate (an error occurred or some required value was missing, so a decision cannot be made)
- Not Applicable (the request can't be answered by this service).

The specification defines five main components (see Figure 3) that handle access decisions; namely Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a Context Handler.

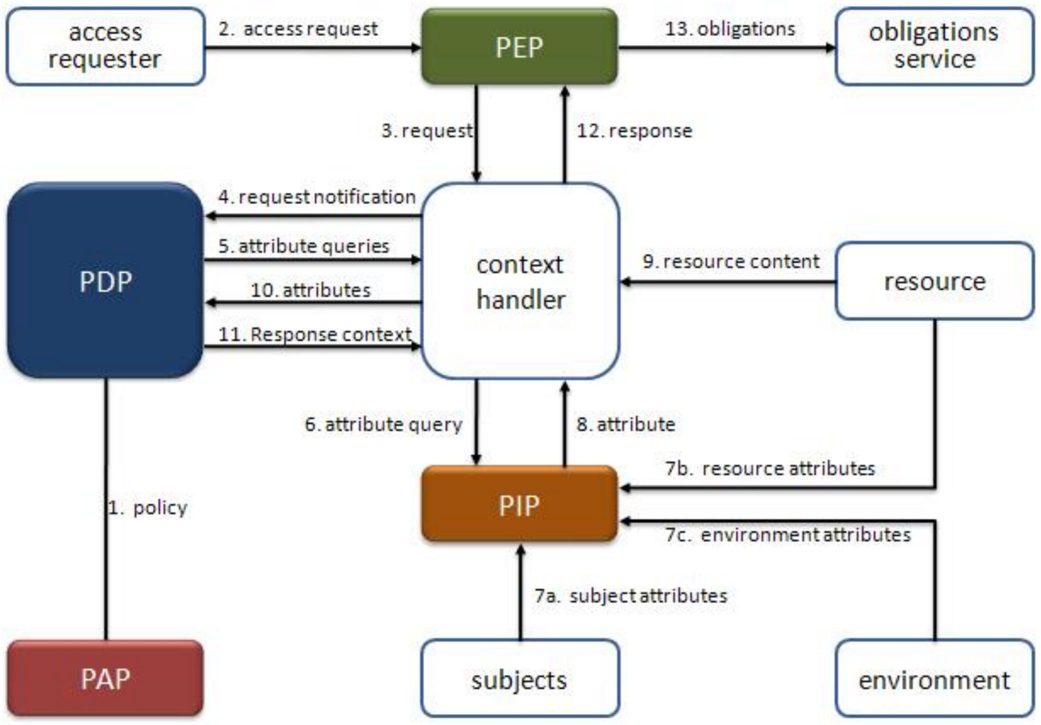


Figure 3 XACML Flow & Architectural Components

The functional purpose of the main components is:

- The Policy Administration Point (PAP) is the repository for the policies and provides the policies to the Policy Decision Point (PDP);
- The Policy Enforcement Point (PEP) is the interface of the whole environment to the outside world. It receives the access requests and evaluates them with the help of the other actors and permits or denies the access to the resource;
- Policy Decision Point (PDP) is the main decision point for the access requests. It collects all the necessary information from other actors and calculates a decision;
- Policy Information Point (PIP) is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g. time), subjects, and so forth.

As already mentioned, XACML uses XSD notation in order to model the three basic artefacts which are required i.e. the policy, the request and the response. Thus, as depicted on Figure 4 three types of XMLs are required by an XACML engine in order to judge upon a decision i.e. the Policy.xml which serializes an actual policy, the Request.xml which serializes an authorization request and the Response.xml that serializes the output of the engine.

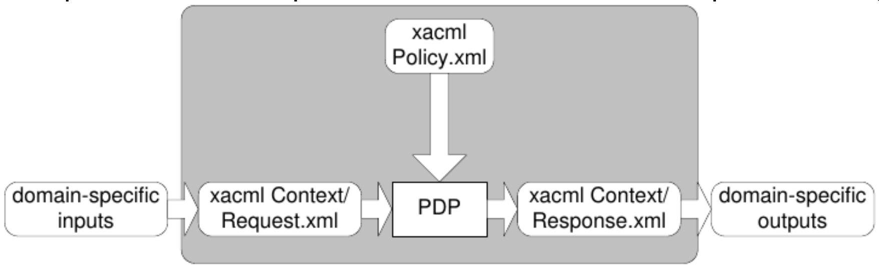


Figure 4 Usage of XML Artefacts

Part of the XSD schema of the policy is depicted on Figure 5. As it is depicted, one policy contains some informative elements (description, issuer etc.) and some elements that relate to the definition of variables and their usage in policy expressions.

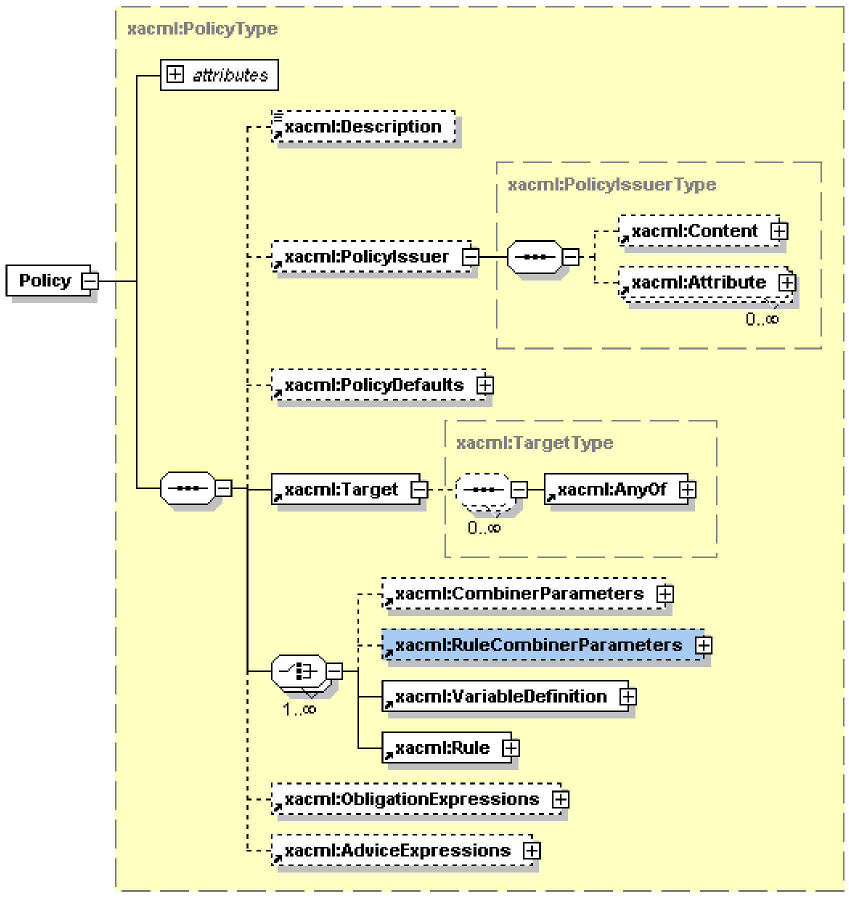


Figure 5 Sample XSD of Policy

In an analogous way, Figure 6 depicts part of the XSD specification of the request and Figure 7 part of the XSD specification of the response.

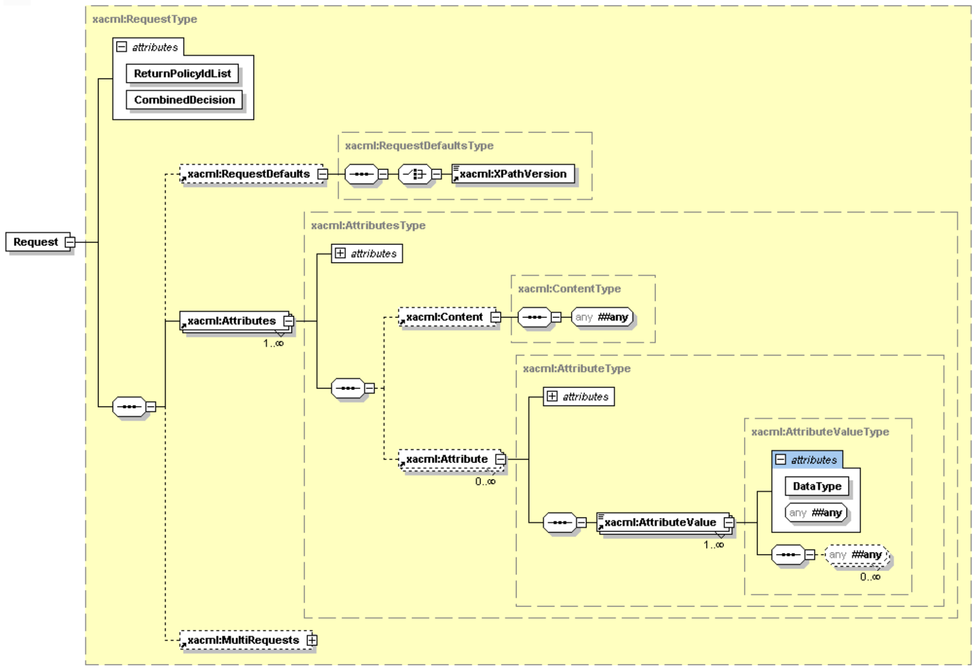


Figure 6 Sample XSD of Request

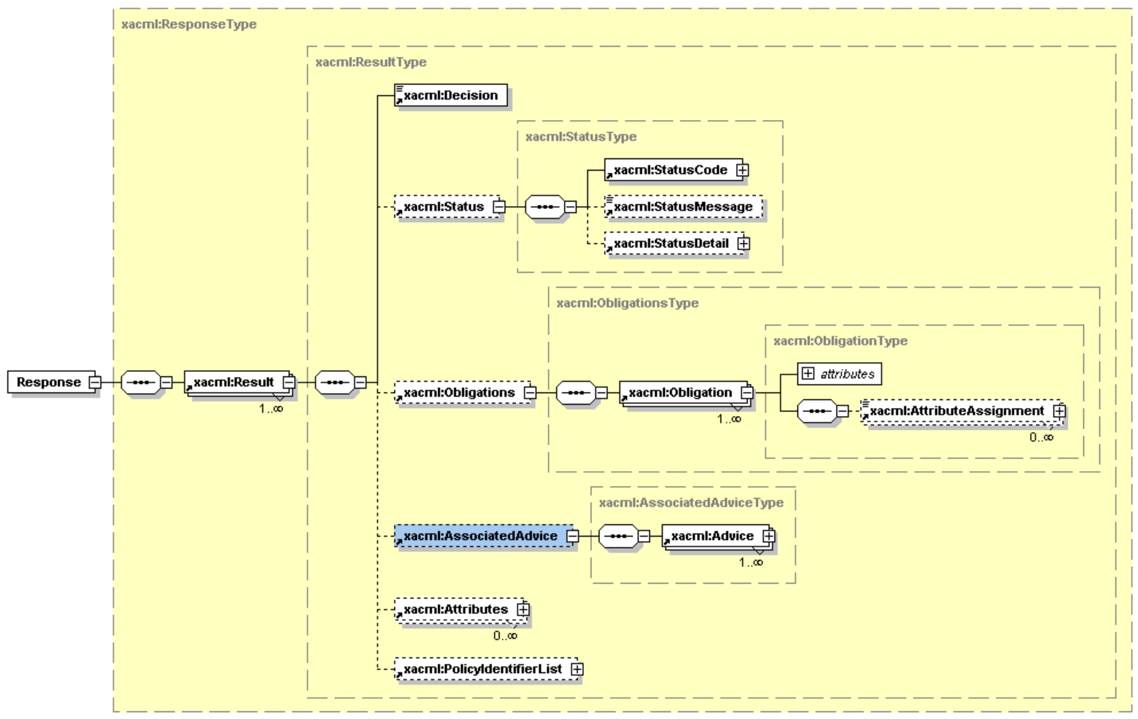


Figure 7 Sample XSD of Response

Delving into the details of the elements is outside the scope of this document. The reader may consult the specification for more details.

3.2 NGAC in a Nutshell

Next Generation Access Control [7] is a fundamental reworking of traditional access control into a form that suits the needs of the modern distributed interconnected systems. NGAC diverges from traditional approaches to access control in defining a generic architecture that is separate from any particular policy or type of policy. NGAC is not an extension of, or adaption of, any existing access control mechanism, but instead is a redefinition of access control in terms of a fundamental and reusable set of data abstractions and functions. NGAC provides a unifying framework capable without extension of supporting not only current many access control approaches, but also novel types of policy that have been conceived but never implemented due to the lack of a suitable enforcement mechanism.
The set of NGAC standards specifies the architecture, functions, operations, and interfaces necessary to ensure interoperability between conforming NGAC implementations. It contains an abstract functional description of architecture to be implemented and also gathers the entities comprising the architecture on the basis of their function. Conforming implementations may employ any design technique that does not violate interoperability. NGAC's access control data is comprised of basic elements, containers, and configurable relations. NGAC uses the terms user, operation, and object with similar meanings. In addition to these, NGAC includes processes, administrative operations, and policy classes[10].
NGAC does not express policies through rules as XACML based ABAC, but instead used the following relation types:

- **assignments** (define membership in containers),
- **associations** (to derive privileges),
- **prohibitions** (to derive privilege exceptions),
- **obligations** (to dynamically alter access state).

NGAC specifies the **assignment** of element x to element y through a tuple (x, y), or using the notation x → y . The assignment relation always implies containment (x is contained in y). The set of entities used in assignments include users, user attributes, object attributes (which include all objects), and policy classes.
In NGAC, to carry out an operation, one or more access rights are required. Two types of access rights apply, non-administrative and administrative, and **the access rights are acquired through associations**.
An association is a triple, denoted by ua---ars---at, where ua is a user attribute, ars is a set of access rights, and at is an attribute, where at may comprise either a user attribute or an object attribute. The meaning of the association ua---ars---at is that **the users contained in *ua* can execute the access rights in *ars* on the policy elements referenced by *at***[10].
Figure 8 illustrates assignment and association relations depicted as graphs with two policy classes—Project Access, and File Management. User attributes are on the left side of the graphs, and object attributes are on the right. The arrows represent assignment or containment relations, while the dashed lines denote associations.

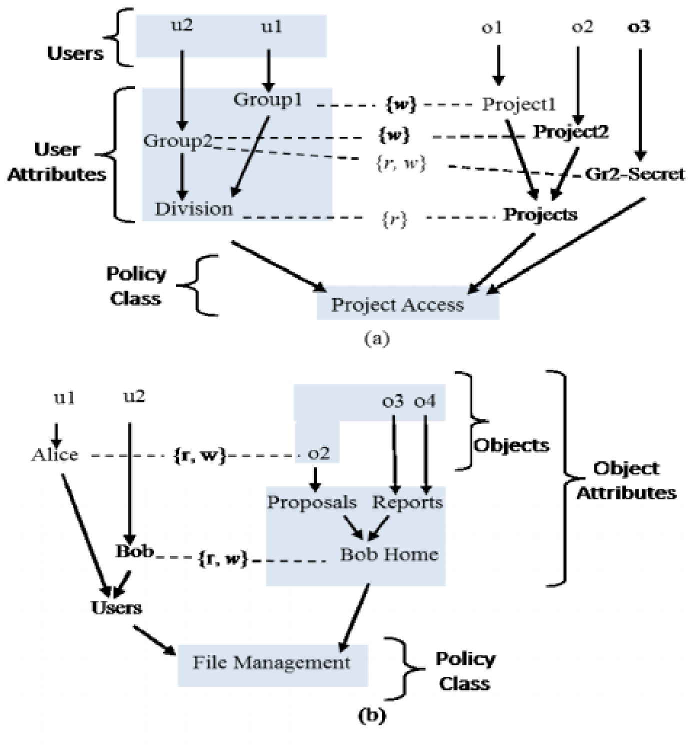


Figure 8 Two Example Assignment and Association Graphs (source:[10])

The associations and assignments in NGAC indirectly specify the privileges in the form (u, ar, e). In simple words, this means that user u is permitted (or has a capability) to execute the access right ar on element e, where e can represent a user, user attribute, or object attribute.

NGAC specification also provides an algorithm that can determine privileges with respect to one or more policy classes and associations. It is important to highlight again that in NGAC the capabilities to perform administrative operations on policy elements and relations are also defined in terms of associations.

3.3 Comparative analysis

As already mentioned, the scope of the deliverable is not to elaborate on the differences that the two most prominent reference implementations have. This elaboration is complicated and requires deep understanding of the foundational principles of both approaches. Out of

the two approaches we have selected XACML mainly because of its architectural clarity and tool support.
However, for the sake of justifying our decision we will provide a high-level overview of a detailed comparison. It could be argued that XACML is similar to NGAC as they both provide **flexible, mechanism-independent representations of policy rules that may vary in granularity, and they employ attributes in computing decisions**. However, XACML and NGAC differ in the way they **formulate policies and treat of attributes, the way the requests are represented and the way that decisions are made**. The following list summarizes the similarities and differences with respect to **a)** separation of authorization functionality from the proprietary operating environment; **b)** attribute and policy management and **c)** operational efficiency.

3.3.1 Separation of Authorization Functionality from Proprietary Operating Environment

Both on XACML and NGAC separation of access control from the proprietary operating environments is used. However, this is done to **different degrees**. An XACML deployment may consist of **multiple operating environments**, each hosting one or more applications and implementing its own method of authentication. For this reason, a common authorization infrastructure shall be used, and an XACML-enabled application uses an operating environment PEP; requests are issued from, and decisions are returned to, an operating environment-specific PEP.

In contrast, NGAC includes a PEP with an API that supports a set of generic, operating environment-agnostic operations (read, write, create, and delete policy elements and relations). An NGAC deployment can include a PEP that recognizes operating environment-specific operations, as the generic operations may not meet the requirements of every application.

3.3.2 Attribute and Policy Management

Although XACML and NGAC have the ability to combine policies, their motivations are different. **XACML's motivation is to resolve conflicts**. That is, policies and rules may have different Effects (Permit or Deny), which must be resolved during evaluation by selectively applying one of several combining algorithms. NGAC's **motivation is to ensure the adherence of combinations of multiple policies when computing a decision** (e.g. DAC and RBAC)[10].
Furthermore, both XACML and NGAC offer a delegation mechanism in support of decentralized administration of access policies and both allow an authority delegation. .

3.3.3 Operational Efficiency and Policy Combination

While XACML and NGAC are similar in that they selectively identify and evaluate policies and conditions that pertain to a request, they differ significantly in their approach. An XACML request is a collection of attribute name-value pairs for the subject (user), action, resource, and environment that must be translated to an XACML canonical form for PDP consumption. XACML identifies applicable policies and rules within policies by matching attributes to Targets. The entire process involves collecting attributes and matching Target conditions through all policies and all rules in applicable policies, issuing administrative requests (for determining a chain of trust for applicable untrusted access policies). If the attributes are not sufficient for the evaluation of an applicable policy or rule, the PDP may search for additional attributes. The access process involves searching at least two data stores (PIP and PDP)..

In NGAC, for responding to an access request, the decision is computed using access control data stored in one database, thus making NGAC more efficient. NGAC identifies relevant policies and attributes directly through assignment relations. Like XACML, NGAC combines policies.
Finally, NGAC and XACML follow a different approach when combining multiple policies. XACML computes and then combines multiple local decisions for creating the final decision,

while NGAC takes multiple policies into consideration when determining the existence of an appropriate privilege. If such a privilege does exist and no exceptions (prohibitions) exist, the request is granted, otherwise it is denied.

4 ABAC Tool Suites

4.1 Balana

Balana[1] was the first open-source reference implementation of the XACML protocol, and is a widely adopted solution. It supports the entire lifecycle of authorisation processing. It is tightly integrated into the WSO2 Identity Server [11]. Balana, as XACML engine of the WSO2 Identity Server has two major components, the Policy Administration Point (PAP) and Policy Decision Point (PDP**).** Figure 9 presented the component architecture of the PDP that is our main interest.

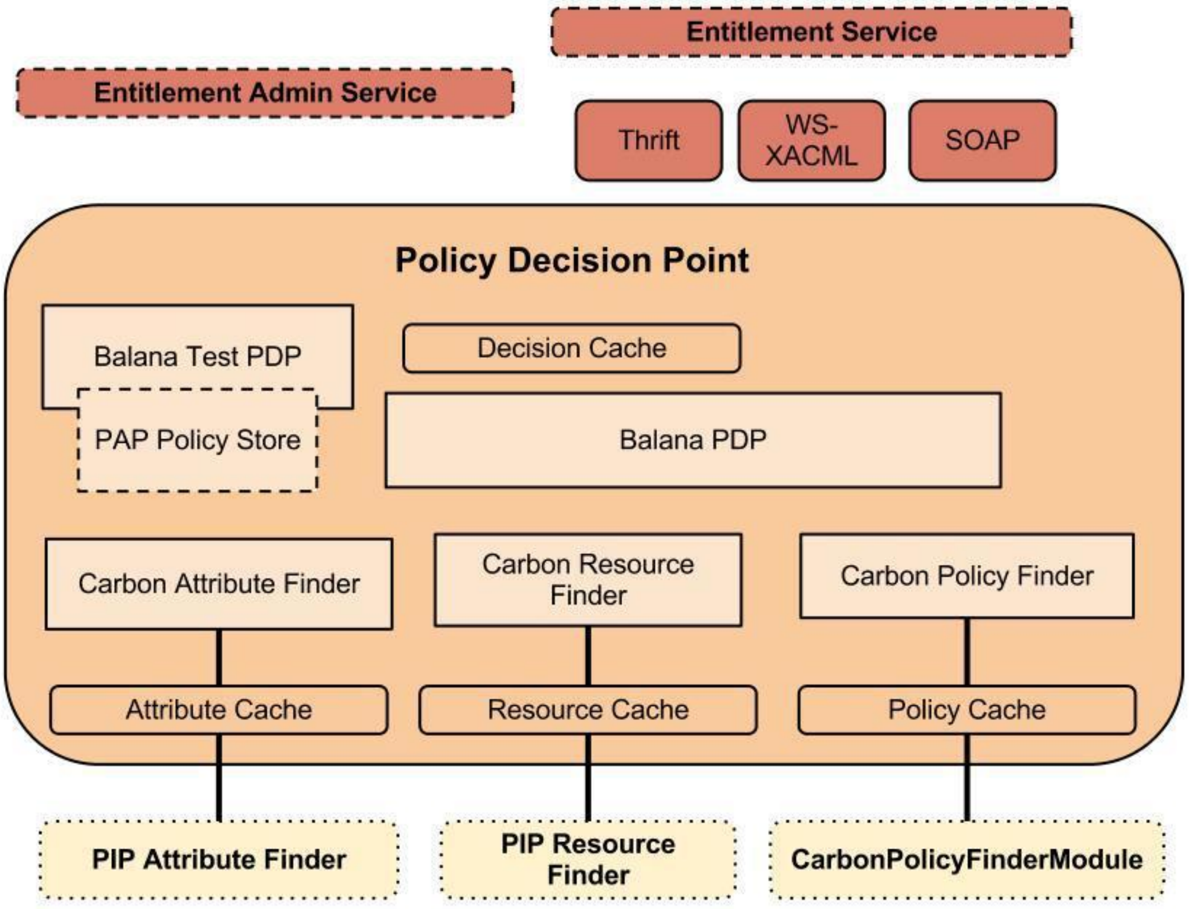


Figure 9 Balana PDP

More details on the components of in the PDP architecture are presented below.
Entitlement Admin Service provides an API that is used to expose all PDP configurations, such as:

- Invalidating caches
- Refreshing policy, attribute, resource finder modules
- Retrieving PDP configurations
- Testing the PDP

Entitlement Service provides XACML authorization API that supports the following three communication methods with PEP.

- SOAP-based Web service
- Apache Thrift binary protocol[12]
- WS-XACML

Balana PDP is the core of the engine of Balana
Balana Test PDP is a duplication of Balana PDP can be only used for testing policies.

[1] https://github.com/wso2/balana

Carbon Policy Finder is a module that finds policies from different policy stores to evaluate an XACML request. Figure 10 presents a high-level diagram of the usage of the carbon policy filter for the collection of the policies to be evaluated.

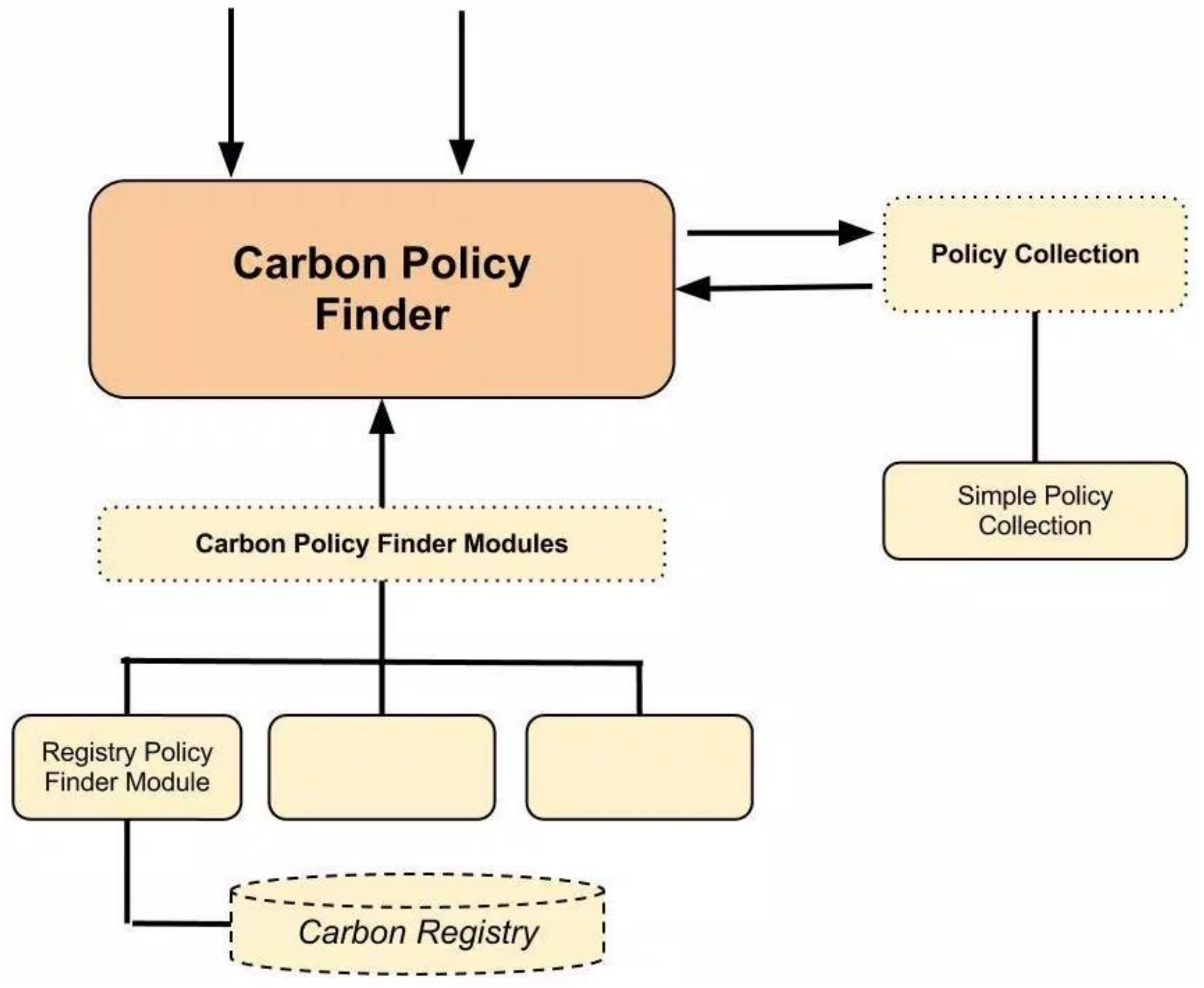


Figure 10 Carbon Policy Filter

Policy finder modules implementing the CarbonPolicyFinderModule interface should be registered and plugged with the Carbon policy finder. WSO2 Identity Server provides by default a Carbon registry-based policy finder module that can retrieve policies from a registry collection. Carbon policy finder finds XACML requests and creates the creates an effective policy. When an update in the policy store happens, Carbon policy finder can be re-initialized automatically by the module, or it can be re-initialized using the API of the Entitlement Admin Service.

Carbon Attribute Finder is a module that is responsible for finding missing attributes for a given XACML request, using the underlying PIP attribute finders. Figure 11 provides a high-level diagram for both the Carbon attribute finder and resource finders.

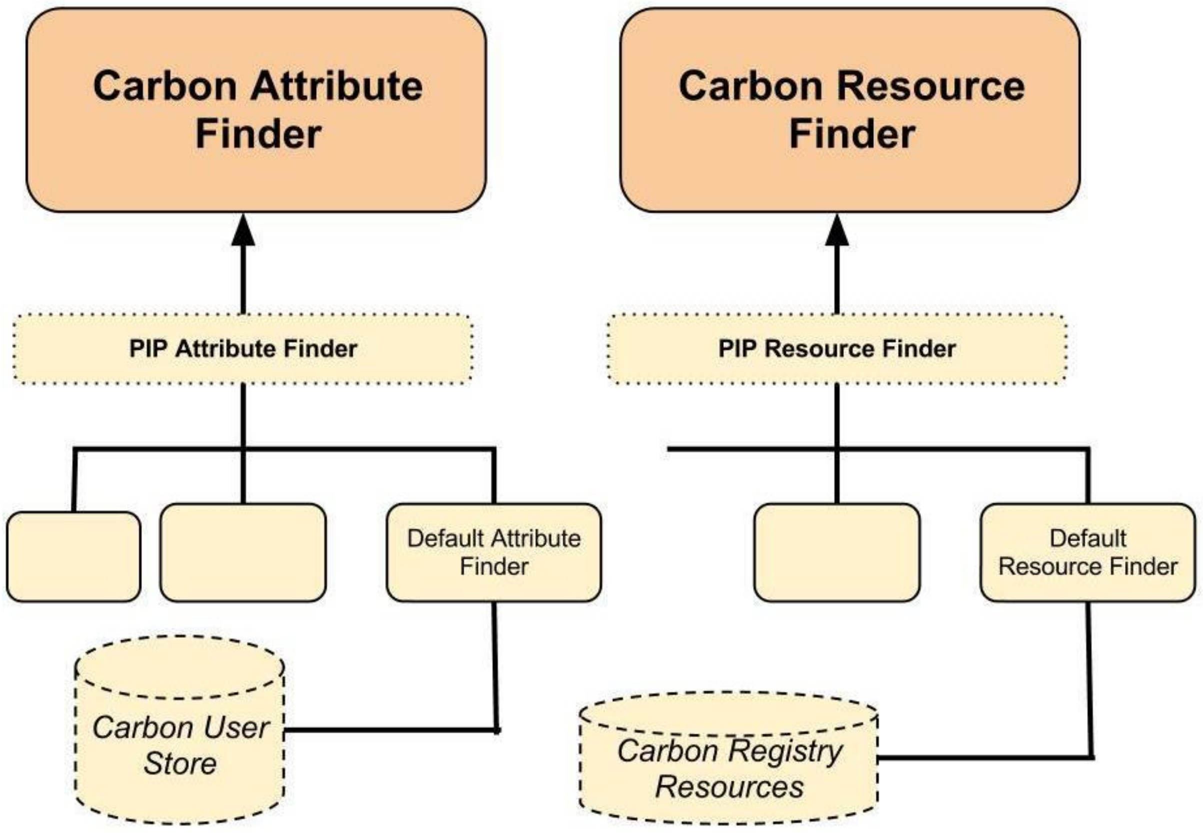


Figure 11 Carbon Attribute Finder

A PIP attribute finder module should implement the PIPAttributeFinder interface, and register it using the entitlement properties configuration file to the Carbon attribute finder. WSO2 Identity Server by default communicates with the underlying user store of the Identity Server that is built with ApacheDS[13].
On runtime, Carbon attribute finder checks for the attribute Id and hands it over to the proper module to handle, while caching mechanism (provided by Carbon attribute finder) is used for caching the findings when possible.

Carbon Resource Finder is used to retrieve children or descendant resources of a given root level resource value, used to fulfil requirements for a multiple decision profile. Similarly to the PIP attribute finder module, it has to implement the PIPAttributeFinder interface.

In general, we consider Balana a highly extensive open source solution, suitable for the needs of ASCLEPIOS.

4.2 PaaSword

PaaSword[14] is an XACML implementation that emphasizes in the **semantic evaluation** of attributes when compared to Balana, as in Balana attributes are exactly matched irrelevant if they belong to a taxonomy. When compared to Balana PaaSword offers **three differentiation features; namely:**

- **Feature-1:** The ability **to harmonize the attribute creation process through the usage of the extensible Context Model.** In traditional XACML the creation of attribute values and subject and resource assignments to those attributes is typically performed in disperse venues **without any notion of coordination or governance**. This is an inherent drawback of the XACML standard. The PaaSword engine alleviates this drawback through the usage of the developed and **extensible Context Model**.

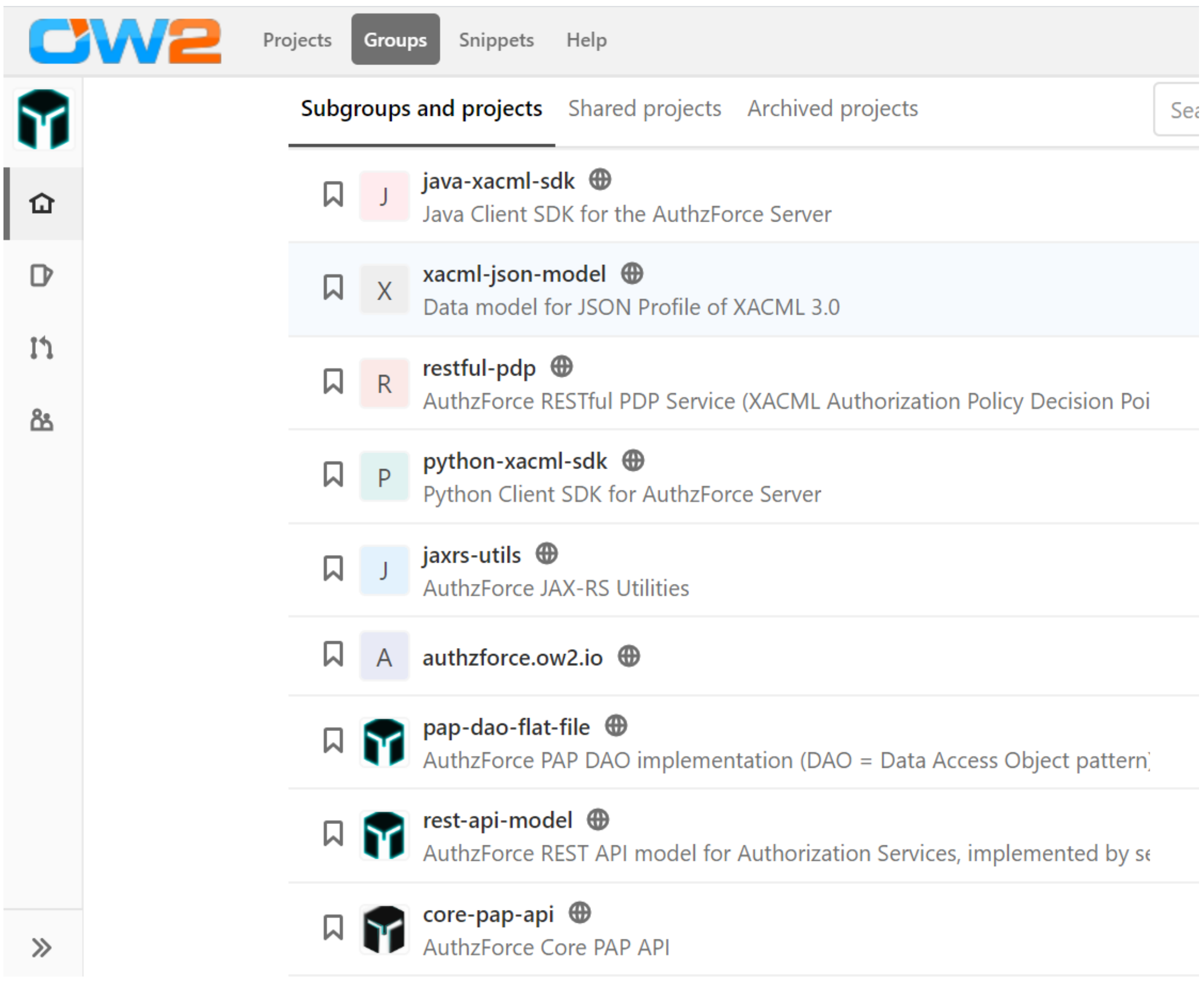
- **Feature-2:** The ability **to decouple the level of granularity of attributes that are used to define policies with the attributes that characterize ‘subjects’, ‘objects’ and the ‘environment’:** The existing mechanism for evaluating the access or deny decision for a given request relies on the expression evaluation of the ‘exact’ attributes that are defined in the provided policies. Exact evaluation means that the relationship between the attributes per se is ignored, i.e. if a policy is defined using a location instance (e.g. FloorA) then an environmental variable that refers to a room that is encapsulated in the specific floor (e.g. RoomA) would never be matched. This mismatch is also alleviated by the PaaSword engine.
- **Feature-3:** The ability **to provide design-time conflict resolution for provided policies**: Since XACML is ‘distributed’ by its nature, the provided policies may be conflicting. The standard is accompanied by a conflict resolution strategy; however, given that policies are expressed ontologically, the framework offers design-time guarantees regarding conflict resolution.

The PaaSword ecosystem does not have the industry adoption of Balana; yet it has more advanced and intuitive interface for managing the rules.

Unfortunately, the disadvantage of PaaSword is the overhead in terms of time for evaluating the XACML policies. The semantic evaluation of policies requires the ontological inferencing of rules. This inferencing has a “severe” performance penalty and as such PaaSword **cannot be used in mission-critical applications**.

4.3 AuthzForce

AuthzForce[15] project provides an ABAC framework compliant with the OASIS XACML v3.0, that mostly consists of an authorization policy engine and a RESTful authorization server. It was primarily developed to provide advanced access control for Web Services or APIs, but is generic enough to address all kinds of access control use cases. AuthzForce is highly modular and as such, can be used in diverse application scenarios.



java-xacml-sdk
Java Client SDK for the AuthzForce Server

xacml-json-model
Data model for JSON Profile of XACML 3.0

restful-pdp
AuthzForce RESTful PDP Service (XACML Authorization Policy Decision Poi

python-xacml-sdk
Python Client SDK for AuthzForce Server

jaxrs-utils
AuthzForce JAX-RS Utilities

authzforce.ow2.io

pap-dao-flat-file
AuthzForce PAP DAO implementation (DAO = Data Access Object pattern)

rest-api-model
AuthzForce REST API model for Authorization Services, implemented by se

core-pap-api
AuthzForce Core PAP API

Figure 12 AuthZForce core components

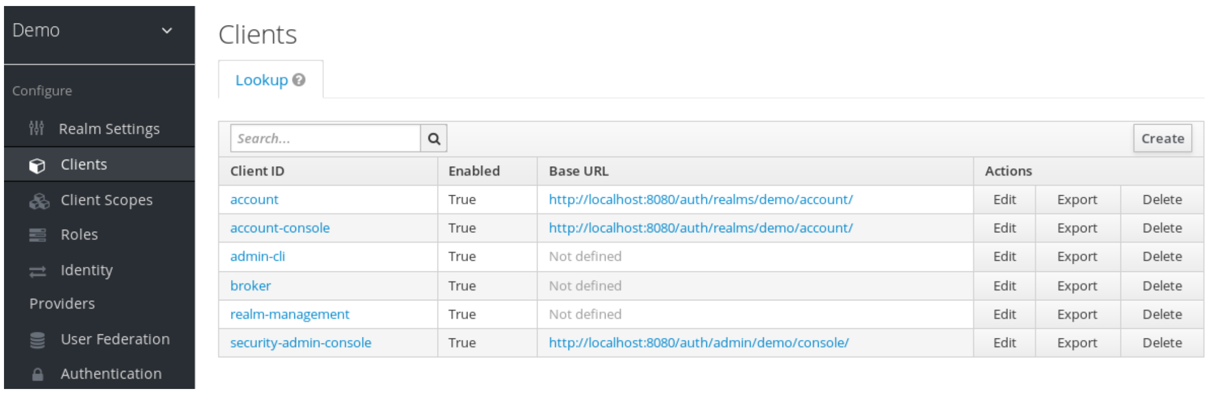
AuthzForce can be used in two ways depending on the needs:

- **Through a Java API**: AuthZForce provides an XACML Policy Decision Point (PDP) engine provided as Java library. Applications can be instantiated and used with the embedded XACML PDP, that is using the API provided by AuthzForce Core.
- **Through a Web API**: AuthZForce provides a multi-tenant HTTP/REST API to PDPs and PAPs (Policy Administration Points). This API, API is provided by AuthzForce Server, can be used by web clients for managing policies, request authorization decisions, etc..

Although AuthzForce does not have the peripheral utilities of Balana (i.e. some administration interfaces) it is functionally equivalent with Balana.

4.4 Keycloak

Before we discuss about Keycloak[16] we have to clarify one thing; Keycloak **is not an ABAC implementation**. In fact, it is **not even an authorization server at all**. Yet is, probably the most **powerful authentication proxy for micro-services** and legacy systems. As such, it **abstracts the functionality of identity extraction and identity verification** for different systems and for different protocols. In parallel, it is able to map users and roles from existing legacy systems in what it calls **authentication realms**.



Client ID	Enabled	Base URL	Actions		
account	True	http://localhost:8080/auth/realms/demo/account/	Edit	Export	Delete
account-console	True	http://localhost:8080/auth/realms/demo/account/	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
realm-management	True	Not defined	Edit	Export	Delete
security-admin-console	True	http://localhost:8080/auth/admin/demo/console/	Edit	Export	Delete

Figure 13 Keycloak Realm Administration

Through configured realms, Keycloak is able to centralize the login-process of various systems through the implementation of many protocols such as oAuth2.0[17] and OpenIDConnect[18] (a.k.a. OIDC). The OpenIDConnect signalling is presented in Figure 14.

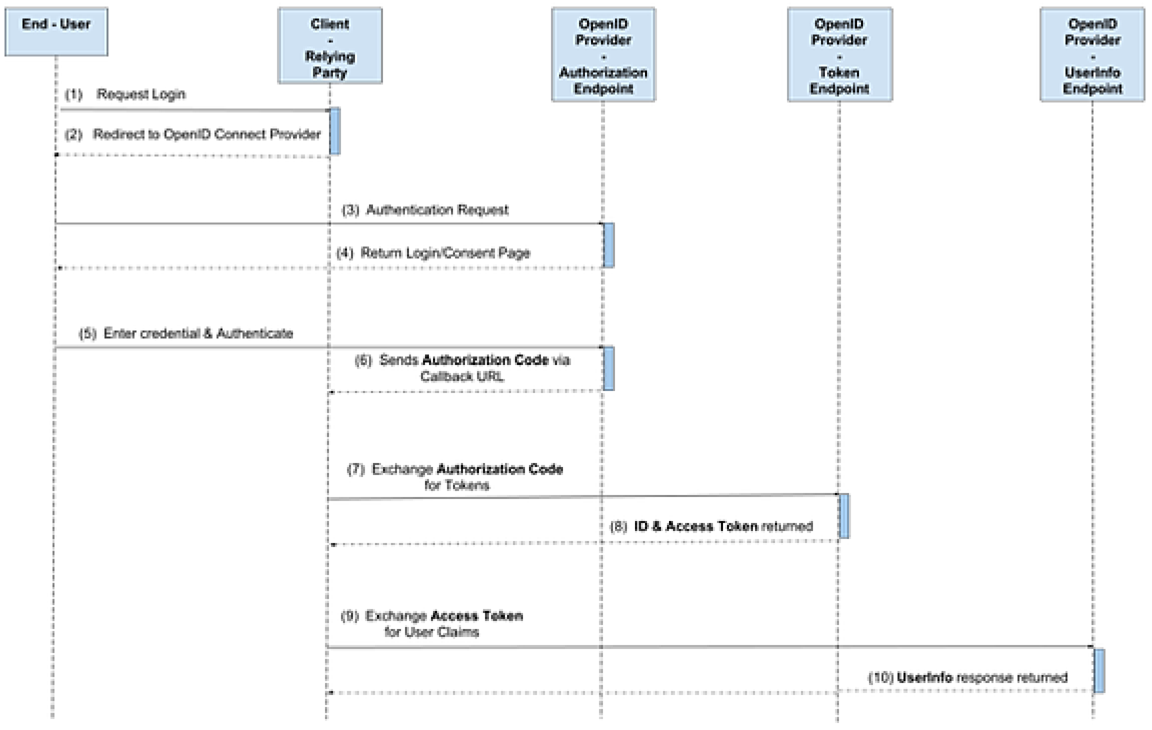

End - User
Client - Relying Party
OpenID Provider - Authorization Endpoint
OpenID Provider - Token Endpoint
OpenID Provider - UserInfo Endpoint
(1) Request Login
(2) Redirect to OpenID Connect Provider
(3) Authentication Request
(4) Return Login/Consent Page
(5) Enter credential & Authenticate
(6) Sends Authorization Code via Callback URL
(7) Exchange Authorization Code for Tokens
(8) ID & Access Token returned
(9) Exchange Access Token for User Claims
(10) UserInfo response returned

Figure 14 OIDC Signalling

According to the flow diagram, a user is attempting to connect to a service (e.g. health-care service X) which supports OIDC. The health care service is redirecting the user to an OIDC provider that is configured to authenticate users based on **a health-care-service-id and a user-role mapping**. The **combination** of the health-care-service-id and the user-role-mapping is addressed as a **realm**.
The OIDC server is “challenging” the user to authenticate based on various methods (username/password, X509 certificate etc.). Upon successful login a distinct set of claims are serialized as a token back to the user in order to use it in his/her interaction with the health-care-service. **These claims contain <u>electronically signed attributes</u> that can be used by the ABAC authorization engine**.

As a result, the OIDC signalling (and Keycloak in general) is extremely crucial for ASCLEPIOS ABAC even if it is not an ABAC engine per se. It acts as an **enabler of verifier attributes**.

5 ASCLEPIOS Authorization mechanism

5.1 The synergy of ABAC and ABE

Before we delve into the details of the ASCLEPIOS authorization mechanism, **we must clarify the boundaries** between ABAC and ABE. ABE is a cryptographic primitive that allows multiple users to encrypt and decrypt files based on their attributes and encryption policies. A crucial question that is raised is the following: **Is ABE an Access Control Mechanism per se?**
The answer is **"leaning to yes" but with severe limitations; so many limitations that someone could claim even no**. On the one hand, **ABE has some inherent authorization properties** since it allows the definition of (encrypted) policies that mandate **whether or not a user-key** can decrypt a cypher or not.

As depicted on Figure 15, an ABE scheme initiated upon a proper initialization phase (step-1) according to which a public file and a master-key-generator are produced.

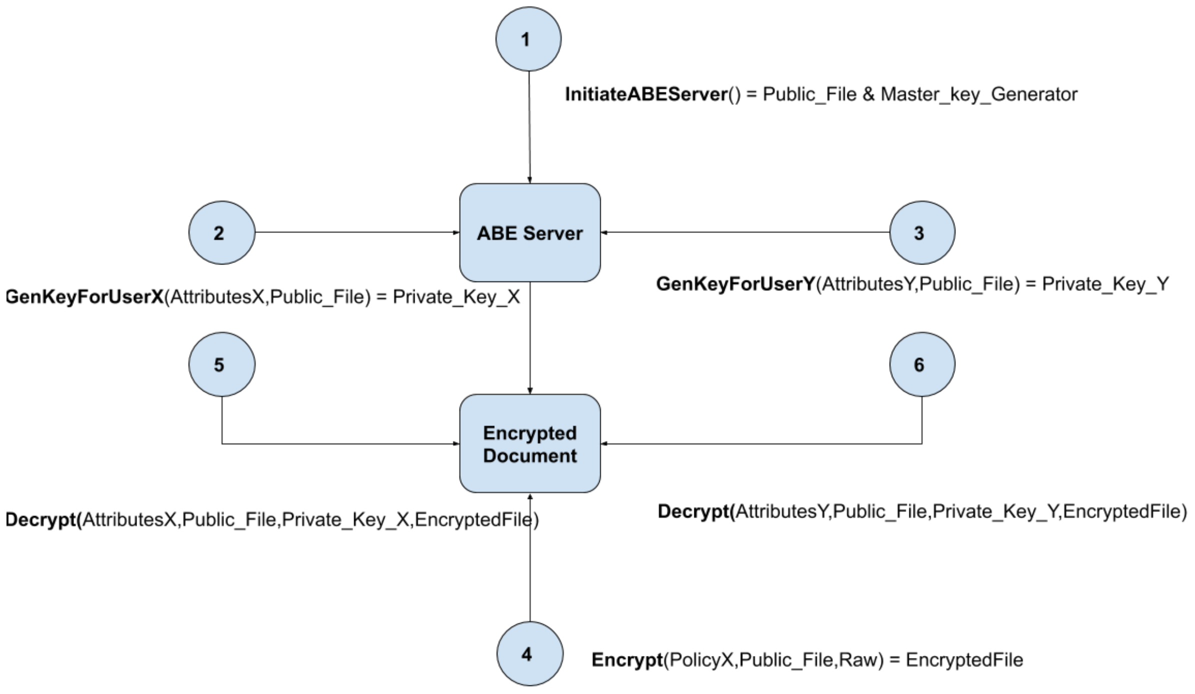

1
InitiateABEServer() = Public_File & Master_key_Generator
2
ABE Server
3
GenKeyForUserX(AttributesX,Public_File) = Private_Key_X
GenKeyForUserY(AttributesY,Public_File) = Private_Key_Y
5
6
Encrypted Document
Decrypt(AttributesX,Public_File,Private_Key_X,EncryptedFile)
Decrypt(AttributesY,Public_File,Private_Key_Y,EncryptedFile)
4
Encrypt(PolicyX,Public_File,Raw) = EncryptedFile

Figure 15 ABE Flow

Upon initiation, many users can ask the ABE server (that holds in a protected zone the master key) to issue a private key based on a set of attributes that are verifiable (e.g. firstname: x1, organization:org1) (steps 2-3).
Each party can encrypt a document using the ABE-Server's public key and a set of attributes; the attributes should match in order to decrypt a file. A policy can be {firstname:y1 or organization:org1}. The policy is encrypted along with the rest of the raw file. Two independent users can attempt to decrypt the file based on their key without having proper knowledge whether their keys can perform decryption or not

Yet, these policies are "**hardcoded**" during the encryption process and cannot be altered upon generation. Moreover, a predicted behaviour of policy enforcement cannot be achieved since a user cannot know a-priori whether or not his key is able to access a protected resource. Above all, ABE scheme supports **only one action i.e. decrypt**. A fully fledged ACM has to support **any type of action on protected resources**.

5.1.1 Sample Flows - Authorization based on ABAC and ABE Policies

In the context of ASCLEPIOS, we propose **the combined use of ABAC and ABE policies for authorization**. We consider a two-step process where ABAC policy is first applied on access attempts to resources (either data or functionality) and subsequently if an ABAC permit is granted, **ABE policy is applied in order to recover the resource symmetric decryption key,** as depicted in Figure 16. Naturally, ABE is applicable only for symmetric keys persisted in an encrypted form. Dynamically generated data or software functionality (e.g. offered through APIs) cannot be protected with ABE.

The following BPMN diagram depicts the aforementioned authorization process.

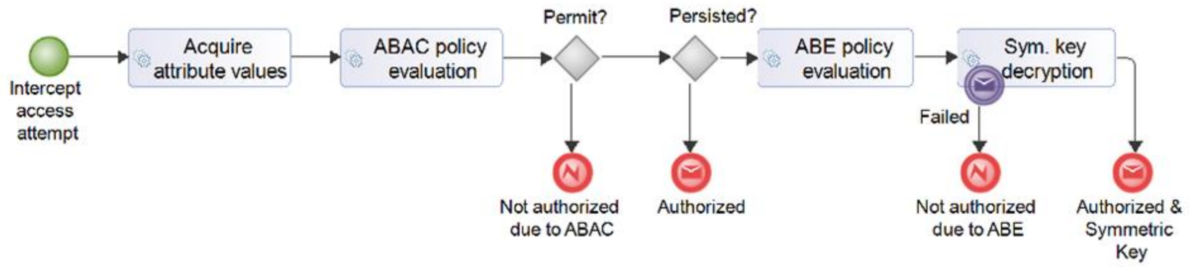
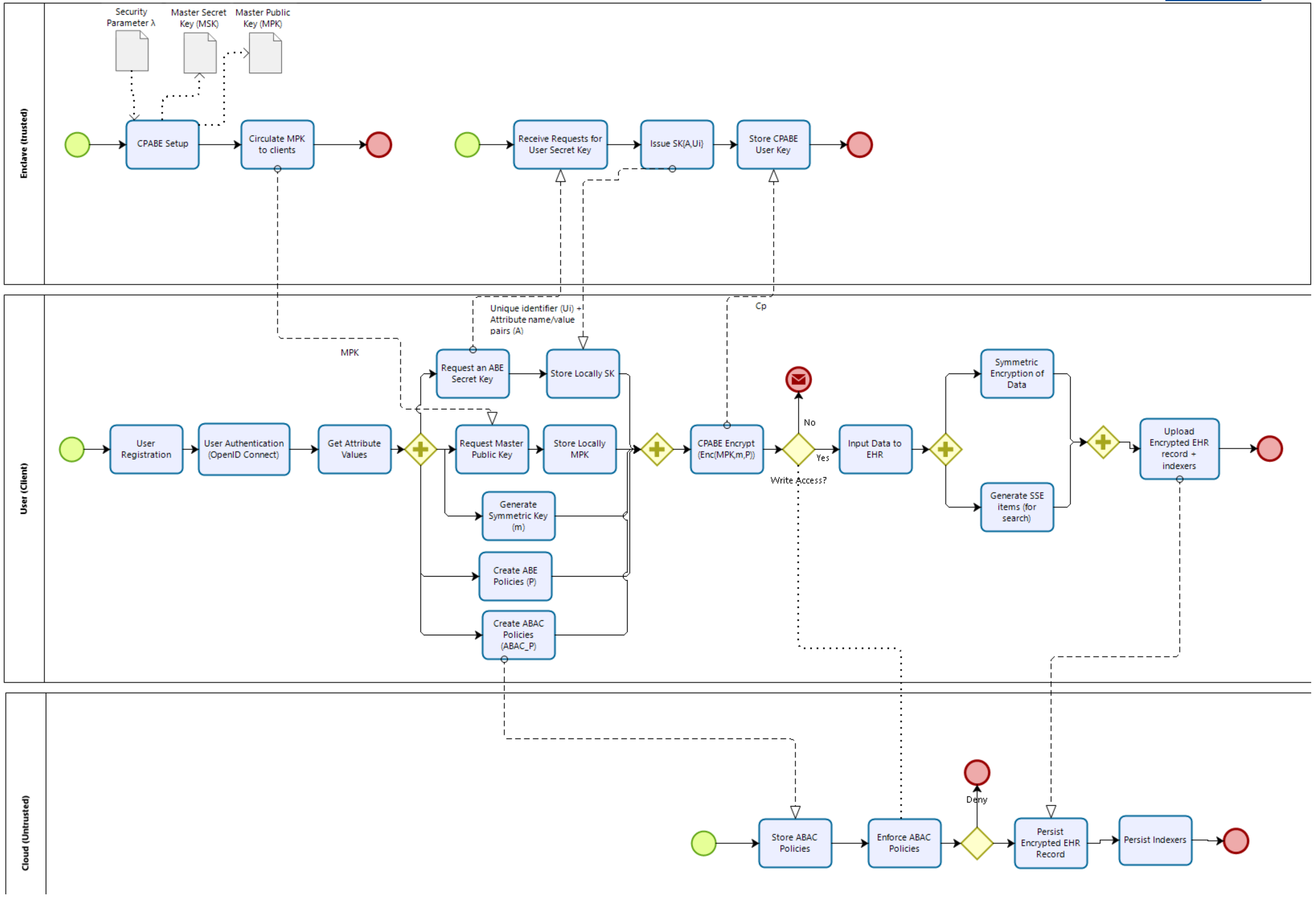


Figure 16 – Generic Authorization process (BPMN view)

Below we provide detailed BPMN diagrams for the complete flow of write action using ABAC and ABE, as it could be used in the scope of ASCLEPIOS.


Enclave (trusted)
Security Parameter λ
Master Secret Key (MSK)
Master Public Key (MPK)
CPABE Setup
Circulate MPK to clients
Receive Requests for User Secret Key
Issue SK(A,Ui)
Store CPABE User Key
User (Client)
MPK
Unique identifier (Ui) + Attribute name/value pairs (A)
Cp
User Registration
User Authentication (OpenID Connect)
Get Attribute Values
Request an ABE Secret Key
Store Locally SK
Request Master Public Key
Store Locally MPK
Generate Symmetric Key (m)
Create ABE Policies (P)
Create ABAC Policies (ABAC_P)
CPABE Encrypt (Enc(MPK,m,P))
No
Yes
Write Access?
Input Data to EHR
Symmetric Encryption of Data
Generate SSE items (for search)
Upload Encrypted EHR record + indexers
Cloud (Untrusted)
Store ABAC Policies
Enforce ABAC Policies
Deny
Persist Encrypted EHR Record
Persist Indexers

ASCLEPIOS is supported by the H2020 Programme under contract no. 826093

Figure 17 – Complete Write Flow

In Figure 18 we present the BPMN diagram for the complete flow of read actions in the scope of ASCLEPIOS using ABAC and ABE.

European Commission

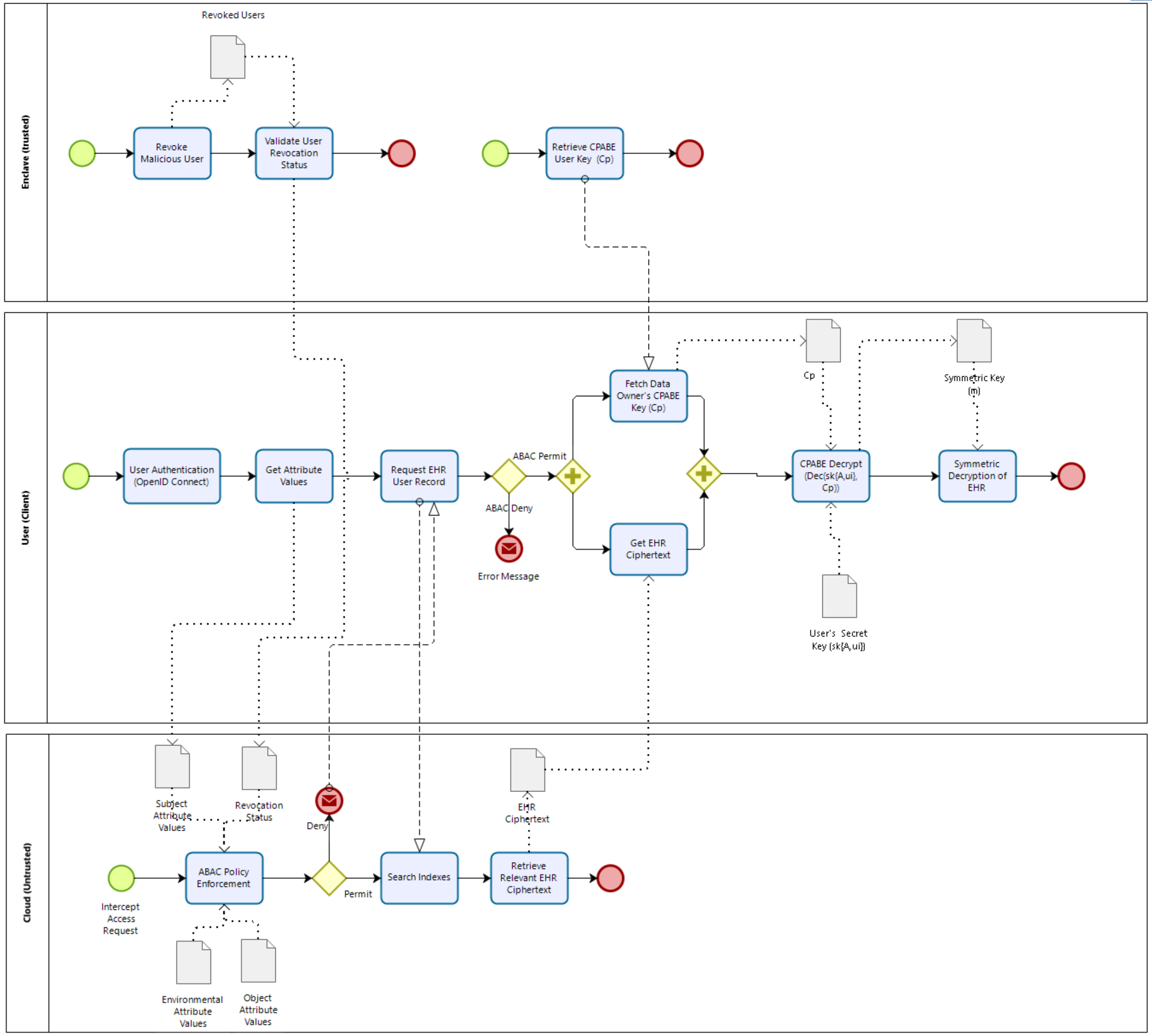


Figure 18 – Complete Read Flow

5.2 Components & Interfaces of ASCLEPIOS Authorization Engine

ASCLEPIOS Authorization engine consists of multiple components as depicted on Figure 19.

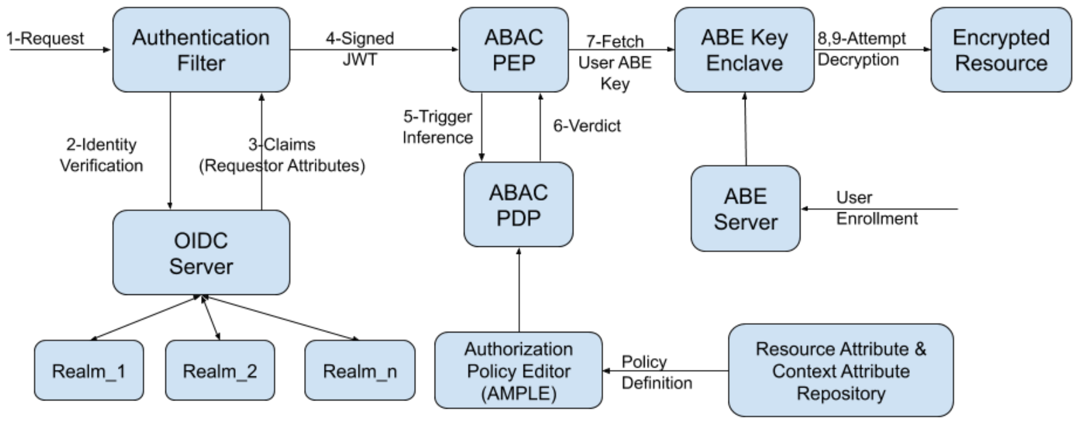


Figure 19 Components & Steps for the Authorization process

Initially, a user is performing a request in order to access a specific encrypted resource. Before the evaluation of whether or not the user is allowed to access the resource the request per se will be intercepted by an authentication filter that is bound to the application.It goes without saying that authentication is always preceding authorization. The role of the authentication proxy is not only to extract the identity (many times called principal-id) but to fetch a set of claims (i.e. attributes) that are verifiably associated with the user per se.

These attributes are electronically signed by the OIDC server. Upon identity extraction, the signed claims along the initial request are intercepted by the PEP element. The implementation of the PEP element is also bound to the code base of the application. The PEP will forward the request and the extracted attributes to the PDP, which is the actual inference engine.

The inference engine, is probably the most complex, in terms of business logic component, since it undertakes the task of identifies which policies are relevant for this request, which attributes are relevant for these policies and finally evaluates the policy expressions based on the values of these policies.

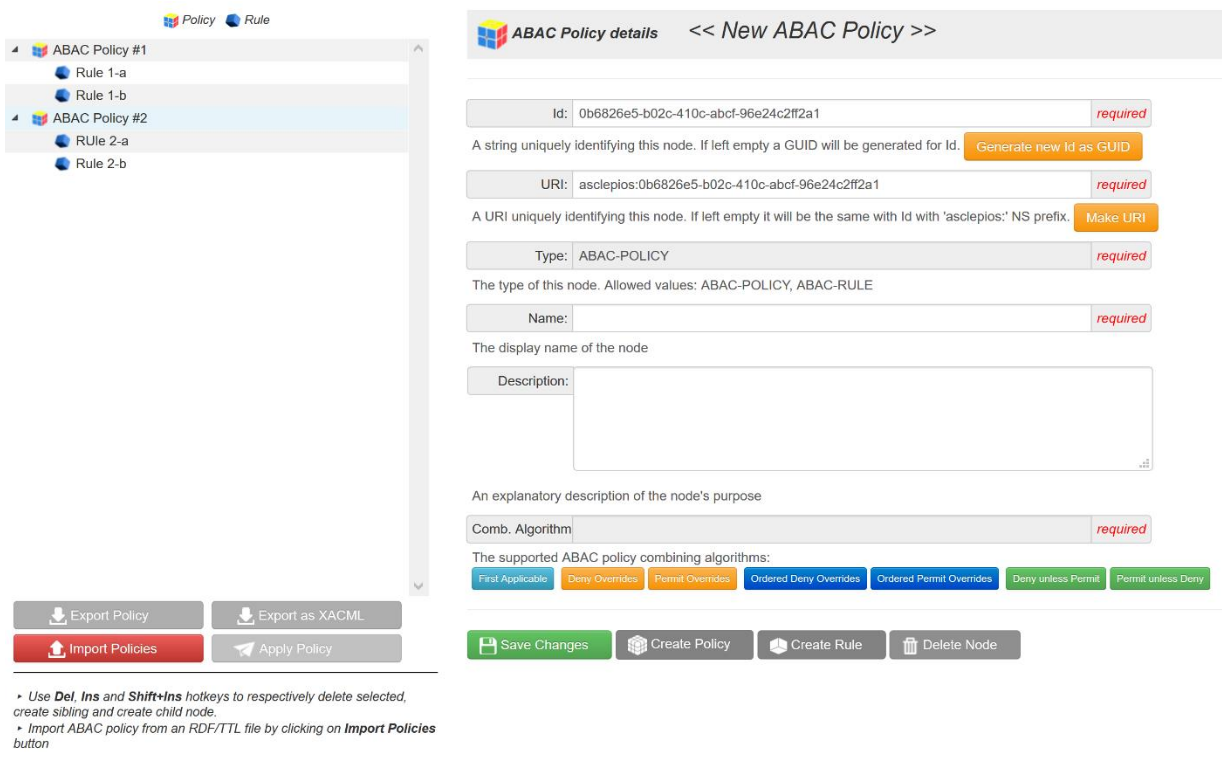
During the evaluation of the expression, some rules may advise towards allowing and some others towards denying the requests. The PDP will consult its conflict resolution policy in order to generate the final verdict. Upon allowance, the application will open the enclave where the user-ABE-key is stored. This key is used in order to “attempt” to open the resource. We refer to the term “attempt” because the attributes that have been used for the encryption policy of the resource may indicate that this user although s/he can access the resource s/he cannot decipher it.

In Figure 19 we also highlight the steps of the authorization process. The steps are presented here:

1. User is attempting to access **a protected and encrypted resource of EHR system-X**.
2. Intercept (and temporarily block) an attempt to access a protected resource. Since the access attempt is an HTTP request the interceptor is evaluating the **OIDC signed claims** that **may exist in the request** (in the form of JWTs).
3. If signed claims do not exist the Interceptor is sending the HTTP-request to the OIDC server with a callback-url that corresponds to the users' initial request (https://systemX/access/resourceY).
4. The user is successfully authenticated (with a single factor or multiple factor schemes) and is landing to the interceptor of system-X
5. The interceptor is triggering an **ABAC PEP signalling**. As such a **collection of attributes pertaining to the requestor, the data owner, the resource being accessed, the attempted action as well as other environmental attributes** is performed. Requestor's attributes are extracted from the signed claims of OIDC server and sent to PDP for evaluation.
6. The **ABAC PDP is performing the evaluation of the access attempt** against ABAC policy (or policies).
   a. If evaluation yields **Deny** then the **access attempt is permanently blocked** and an error is returned to the user.
   b. If evaluation **yields Permit** then
      i. If the protected resource is a service or functionality, then the access attempt is allowed to proceed, and the authorization process completes.
      ii. If the protected resource **is a persisted (and encrypted) dataset** then the authorization process **continues to ABE**.
7. Use of the ABE policy of the resource, along with the **user's secret key** (which is associated to specific attribute values), in order to recover the resource ABE decryption key.
8. **ABE decryption** key is used to decipher the encrypted symmetric key that decrypts the dataset.
   a. If decryption fails, then an error is returned to the user, and access attempt is essentially blocked.
   b. If symmetric key decryption succeeds, the user can continue with dataset decryption.
9. The symmetric key is used to decipher the protected dataset.

5.3 Policy Authoring Environment

As the *ASCLEPIOS Models and PoLicies Editors* (AMPLE) in it whole has been presented in D3.2[2], the way to create policies that are enforced by the ASCLEPIOS Authorization mechanism has been covered in extend. In general, a user can use the AMPLE Policy Editor to create a new ABAC Policy, as depicted in Figure 20.



Id: 0b6826e5-b02c-410c-abcf-96e24c2ff2a1 *required*
A string uniquely identifying this node. If left empty a GUID will be generated for Id.

URI: asclepios:0b6826e5-b02c-410c-abcf-96e24c2ff2a1 *required*
A URI uniquely identifying this node. If left empty it will be the same with Id with 'asclepios:' NS prefix.

Type: ABAC-POLICY *required*
The type of this node. Allowed values: ABAC-POLICY, ABAC-RULE

Name: *required*
The display name of the node

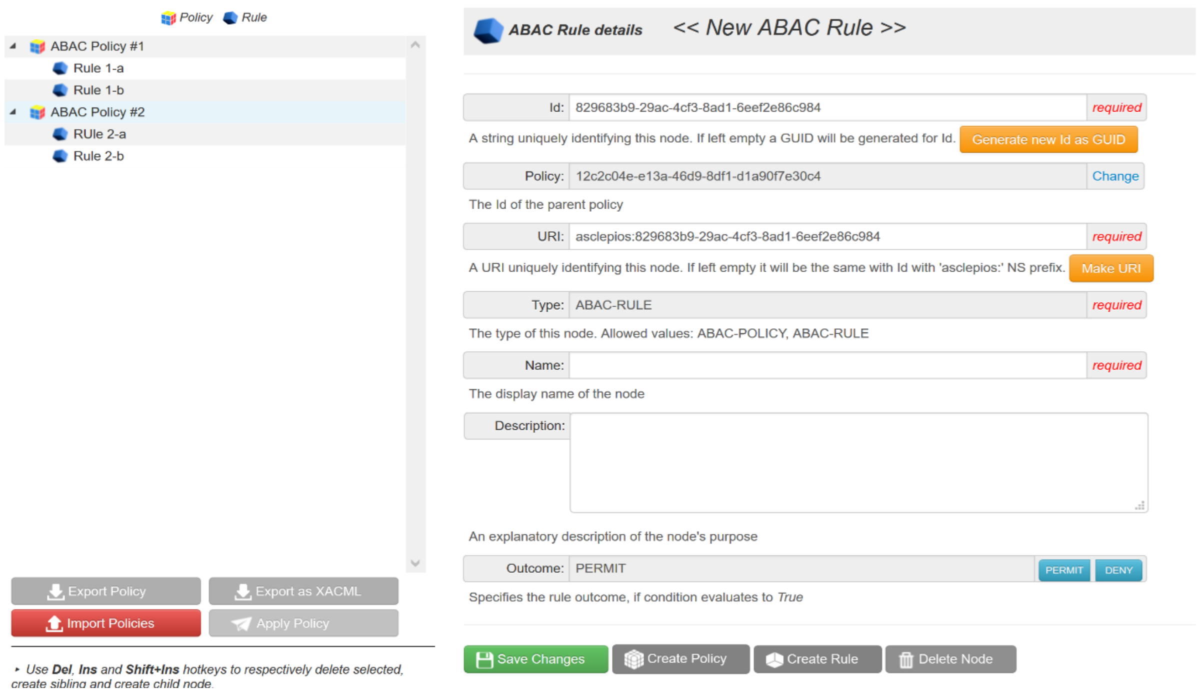
Description:
An explanatory description of the node's purpose

Comb. Algorithm *required*
The supported ABAC policy combining algorithms:

- *Use **Del**, **Ins** and **Shift+Ins** hotkeys to respectively delete selected, create sibling and create child node.*
- *Import ABAC policy from an RDF/TTL file by clicking on **Import Policies** button*

Figure 20 –ABAC Policy creation

And as depicted in Figure 21 user shall used the AMPLE editor to also create the ABAC Policy Rule.



Id: 829683b9-29ac-4cf3-8ad1-6eef2e86c984 *required*
A string uniquely identifying this node. If left empty a GUID will be generated for Id.

Policy: 12c2c04e-e13a-46d9-8df1-d1a90f7e30c4
The Id of the parent policy

URI: asclepios:829683b9-29ac-4cf3-8ad1-6eef2e86c984 *required*
A URI uniquely identifying this node. If left empty it will be the same with Id with 'asclepios:' NS prefix.

Type: ABAC-RULE *required*
The type of this node. Allowed values: ABAC-POLICY, ABAC-RULE

Name: *required*
The display name of the node

Description:
An explanatory description of the node's purpose

Outcome: PERMIT
Specifies the rule outcome, if condition evaluates to *True*

- *Use **Del**, **Ins** and **Shift+Ins** hotkeys to respectively delete selected, create sibling and create child node.*

Figure 21 –ABAC Policy rule creation

For createing the rule properly and allow ABAC engine to validate the rule, the corresponding set of attributes, instances, properties, have to be added. While this has been also coverd in D3.2, in the following of this section we cover again the procedure for definition of attributes and creation of an ABAC and ABE policy, but this time elaborated with the policy validation process.

5.3.1 Policy Validation

Policy validation is the process of checking a given ABAC or ABE policy to verify it meets certain requirements imposed by legislation or by corporate guidelines, in order to reach a minimum level of quality (see D3.2 [2]). It can be performed when developing a policy or before putting it into effect.

Policy validation checks the given policy against one or more sets of rules. Each rule (also termed as Guideline) examines a certain feature of the given policy, for instance checks whether a policy includes an expression of the IP address of the request. For more advanced checks several rules must be used in conjunction. Such rules are grouped into sets.

In more detail, each rule comprises a validation condition and an outcome that is returned if the condition is met. Conditions check for the existence (or non-existence) of specified attributes, attribute properties or whole expressions (including attribute, property, operator and value). Outcomes can be Error messages, which block policy from being put into effect, Warnings, which issue messages but allow policy being put into effect, and Informational messages (which again don't block policy from being put into effect). Moreover, a policy type filter can also be applied for narrowing rule applicability to ABAC or ABE policies only. The rule template is:

[OUTCOME] when [POLICY TYPE] [contains / does not contain] [Attribute / Property / Expression]: "Specification of ATTRIBUTE / PROPERTY / EXPRESSION"

Few indicative examples are:

- [Error] *when* [ABE Policy] [does not contain] [Expression] : "NetworkLocation.hasZone = 'Zone1' "
- [Warn] *when* [Any] [contains] [Property] : "NetworkLocation.hasPort"
- [Info] *when* [ABE Policy] [contains] [Attribute] : "NetworkLocation"

5.3.1.1 Types of Validation

In accordance to deliverable D3.1[1], two types of policy validations are considered. The first is called *Policy Inspection* and its purpose is to spot policy deficiencies and deter their usage by issuing errors. The latter is called *Security Awareness* and its purpose is to identify lacks in policy checks that might result in security issues. Security awareness checks should follow the CAPEC classification of security threats and should issue warning (i.e. not blocking policy to be put into effect).

In technical terms both policy validation types are implemented by a similar approach of rule-based policy checking. In AMPLE editor's user interface, the two validation types are distinguished by using different diacritic icons, terms and different set hierarchies, but apart from that the validation process follows the same steps.

5.3.1.2 Policy Validation set / rule creation

Policy validation sets and rules are defined using the AMPLE editor presented in deliverable D3.2. The policy validation editor can be loaded either by clicking on *"Policy Validation management"* button in AMPLE welcome page or the corresponding menu item.

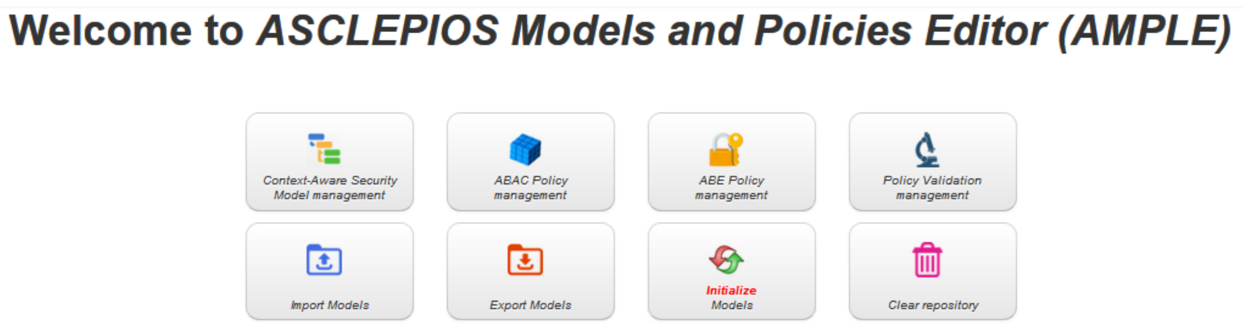
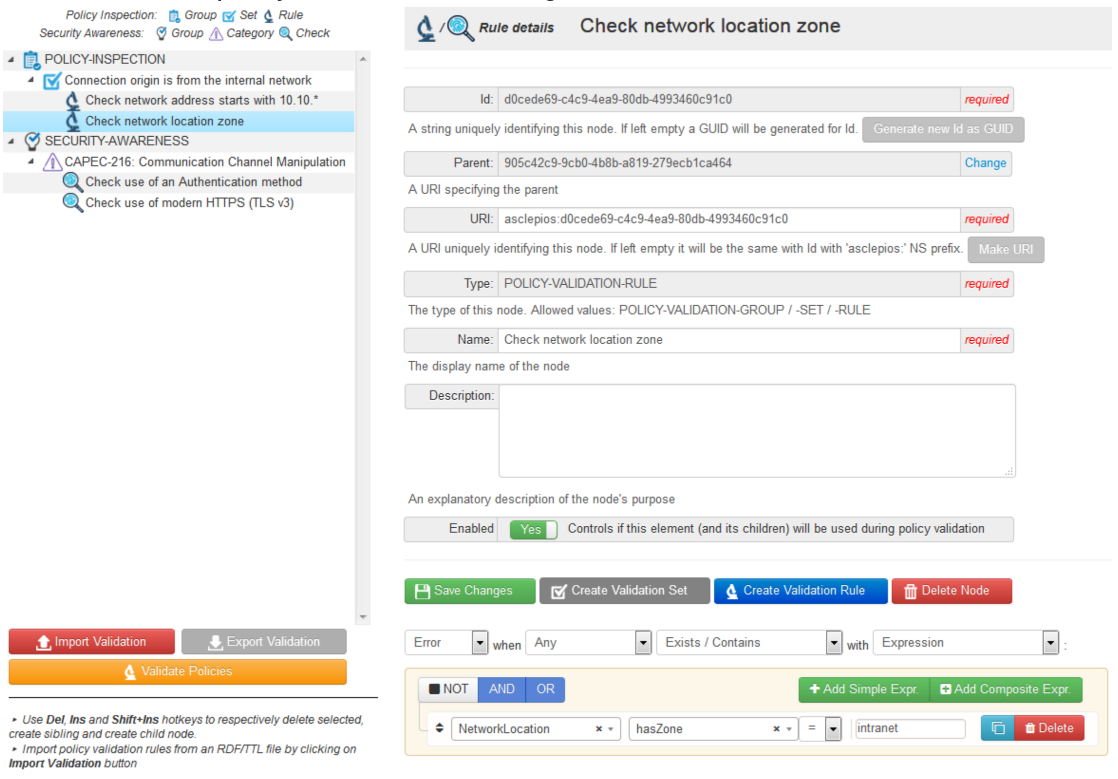

Welcome to ASCLEPIOS Models and Policies Editor (AMPLE)
Context-Aware Security Model management
ABAC Policy management
ABE Policy management
Policy Validation management
Import Models
Export Models
Initialize Models
Clear repository

Figure 22 AMPLE welcome page

A screenshot of the policy validation editor is given next.


Policy Inspection: Group Set Rule
Security Awareness: Group Category Check
POLICY-INSPECTION
Connection origin is from the internal network
Check network address starts with 10.10.*
Check network location zone
SECURITY-AWARENESS
CAPEC-216: Communication Channel Manipulation
Check use of an Authentication method
Check use of modern HTTPS (TLS v3)
Import Validation
Export Validation
Validate Policies
Rule details
Save Changes
Create Validation Set
Create Validation Rule
Delete Node
NOT
AND
OR
Add Simple Expr.
Add Composite Expr.
Delete

▸ *Use **Del**, **Ins** and **Shift+Ins** hotkeys to respectively delete selected, create sibling and create child node.*
▸ *Import policy validation rules from an RDF/TTL file by clicking on **Import Validation** button*

Check network location zone

Id: d0cede69-c4c9-4ea9-80db-4993460c91c0 required
A string uniquely identifying this node. If left empty a GUID will be generated for Id. Generate new Id as GUID

Parent: 905c42c9-9cb0-4b8b-a819-279ecb1ca464 Change
A URI specifying the parent

URI: asclepios:d0cede69-c4c9-4ea9-80db-4993460c91c0 required
A URI uniquely identifying this node. If left empty it will be the same with Id with 'asclepios:' NS prefix. Make URI

Type: POLICY-VALIDATION-RULE required
The type of this node. Allowed values: POLICY-VALIDATION-GROUP / -SET / -RULE

Name: Check network location zone required
The display name of the node

Description:
An explanatory description of the node's purpose

Enabled Yes Controls if this element (and its children) will be used during policy validation

Error when Any Exists / Contains with Expression :

NetworkLocation hasZone = intranet

Figure 23 Policy validation editor

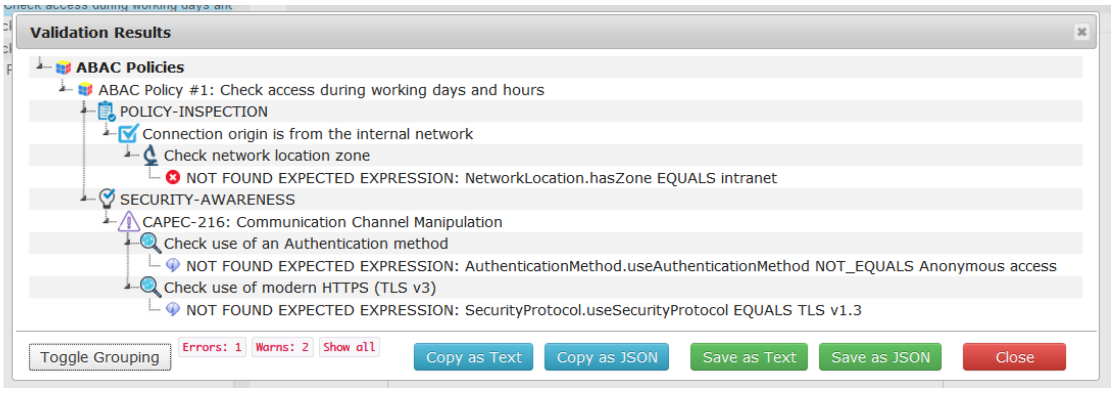
In order to create a new validation set, the user must select either POLICY-INSPECTION or SECURITY-AWARENESS node in the left-hand side tree. Subsequently, user needs to click the "Create Validation Set" button, fill in the validation set data (name, description, URI) and click "Save Changes" button. After successful persistence of the data, the page will reload and an item for the new validation set will be available in the left-hand side tree.
In order to create a new validation rule, the user must select the parent validation set node in the left-hand side tree. Subsequently, she needs to click the "Create Validation Rule" button, fill in the validation rule data (name, description, URI) as well as the data of the rule condition (i.e. rule output, policy type filter, contains/not contains flag, expression type etc.) Eventually, she must click "Save Changes" button. After successful saving, the page will reload and an item for the new validation rule will be available in the left-hand side tree, under the parent validation set node.
Selecting a node in the left-hand side tree will load its details into the details form in the main area of the page. The user can modify this information and store them by clicking on "Save Changes" button. Selecting "Delete Node" the user can delete the selected rule or set. Moreover, it is possible to disable (and re-enable) certain validation rules or sets. This is

controlled by “Enabled” switch button at the last line of the set/rule details form (at the main area of the Policy validation editor page).

5.3.1.3 Use of Policy Validation sets / rules

The policy validation sets and rules are used to check policies in two ways:

- When the user clicks on “Apply Policy” button, in ABAC or ABE policy editor. In this case the policy is checked against all applicable policy validation rules. If any errors, warnings or informational messages are issued during validation, a dialog will pop up listing all of them. If no messages are issued then ABAC or ABE policy will be sent to Policy Enforcement mechanism in order to put it into effect.
  If at least an error message is issued, the policy cannot be put into effect. If there are only warnings or informational messages (no errors) the user has the possibility to review the messages and click the “Continue” button in order to send policy to the Policy Enforcement mechanism, or click “Close” button and correct the policy.



```
ABAC Policies
  ABAC Policy #1: Check access during working days and hours
    POLICY-INSPECTION
      Connection origin is from the internal network
        Check network location zone
          NOT FOUND EXPECTED EXPRESSION: NetworkLocation.hasZone EQUALS intranet
    SECURITY-AWARENESS
      CAPEC-216: Communication Channel Manipulation
        Check use of an Authentication method
          NOT FOUND EXPECTED EXPRESSION: AuthenticationMethod.useAuthenticationMethod NOT_EQUALS Anonymous access
        Check use of modern HTTPS (TLS v3)
          NOT FOUND EXPECTED EXPRESSION: SecurityProtocol.useSecurityProtocol EQUALS TLS v1.3
```

Figure 24 Policy validation results

- When the user clicks on “Validate Policies” button, in Policy validation editor. In this case all ABAC and ABE policies are checked against the selected rule or set. When validation completes a dialog will pop and list validation results sorted by validation set and rule. This can be changed to sort messages by ABAC or ABE policy and then by validation set and rules. This variation of policy validation is provided for checking the validation rules while creating them. For this reason, the results dialog does not provide the possibility to put a policy into effect.

5.3.1.4 Policy Validation Example

In this section, a complete policy validation example is provided to highlight the use of policy validation with regards to ABAC and ABE policies. Let us suppose that a hospital’s policies require that accessing patients’ records can only be allowed from the internal network (Intranet). Moreover, the *Attack Pattern 216* of CAPEC classification, dictates that communications must be encrypted to mitigate *Communication Channel Manipulation* attacks (only one CAPEC pattern will be considered in this example).

- An error is issued when authorization policies defined for protecting EHR records don’t investigate whether the connection origin comes from the internal network.
- CAPEC-216: Communication Channel Manipulation[2]. The corresponding audit issues a warning when communication encryption is not considered in authorization policies.

[2] CAPEC-216 attack pattern: https://capec.mitre.org/data/definitions/216.html

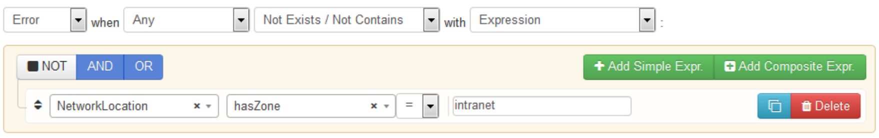
We assume the necessary means for collecting the needed data (communication origin and encryption) are available to the Policy enforcement mechanism through the proper interceptors.

5.3.1.4.1 Sample policy validation rules

The first validation requirement could be implemented in Policy validation editor, as a Policy Inspection check with the following rule:

[Error] when [Any] [does not contain] [Expression]: “NetworkLocation.hasZone = ‘intranet’ ”

In the policy validation editor the corresponding validation rule is defined as presented in Figure 25.



Error when Any Not Exists / Not Contains with Expression :

NOT AND OR | Add Simple Expr. | Add Composite Expr.

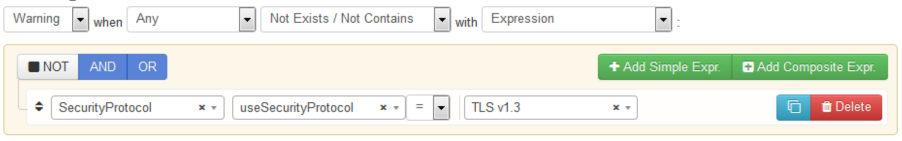
NetworkLocation | hasZone | = | intranet | Delete

Figure 25 Policy inspection rule: Check network location zone

CAPEC-216 attack pattern mitigation section specifies the following: “Encrypt all sensitive communications using properly-configured cryptography. Design the communication system such that it associates proper authentication/authorization with each channel/message.” These specifications could be implemented with the next Security Awareness rules:

[Warning] when [Any] [does not contain] [Expression]: “SecurityProtocol. useSecurityProtocol = ‘TLS v1.3’ ” **and** [Warning] when [Any] [does not contain] [Expression]: “AuthenticationMethod.useAuthenticationMethod ≠ ‘Anonymous access’ ”

In policy validation editor the corresponding validation rules are defined as presented in Figure 26 and Figure 27.

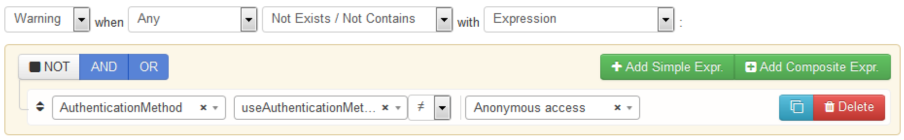


Warning when Any Not Exists / Not Contains with Expression :

NOT AND OR | Add Simple Expr. | Add Composite Expr.

SecurityProtocol | useSecurityProtocol | = | TLS v1.3 | Delete

Figure 26 Security awareness rule: Check the use of modern HTTPS (TLS v3)



Warning when Any Not Exists / Not Contains with Expression :

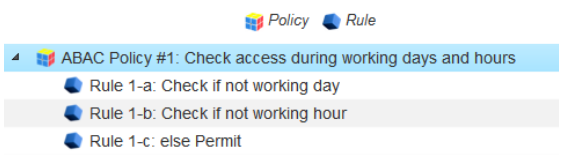
NOT AND OR | Add Simple Expr. | Add Composite Expr.

AuthenticationMethod | useAuthenticationMet... | ≠ | Anonymous access | Delete

Figure 27 Security awareness rule: Check the use of an Authentication method

5.3.1.4.2 Sample authorization policies

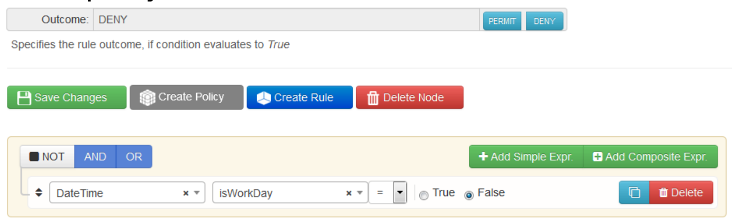
Next, let us assume the following two ABAC and ABE policies to be used. The ABAC policy investigates whether an access request is attempted during working days and hours. Obviously, errors and warnings must be reported during validation, since neither communication origin nor security protocol nor authentication is checked.


Policy Rule

ABAC Policy #1: Check access during working days and hours
- Rule 1-a: Check if not working day
- Rule 1-b: Check if not working hour
- Rule 1-c: else Permit

Figure 28 ABAC policy for checking whether an incoming request detected during working days and hours

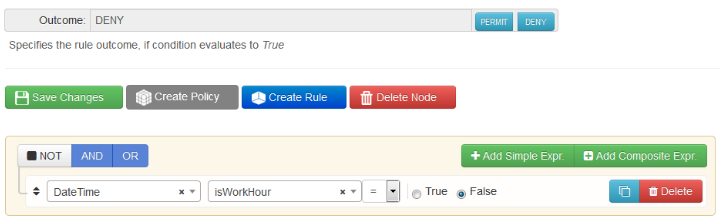
The ABAC policy comprises the following (ABAC) rules. The two rules evaluated first deny access, if it is not working day or working hour. The last one permits access unconditionally. The combining algorithm of the policy is *First Applicable*, meaning that the first rule that matches will provide the policy result.


Outcome: DENY

Specifies the rule outcome, if condition evaluates to *True*

DateTime × isWorkDay × = True [✓ False]

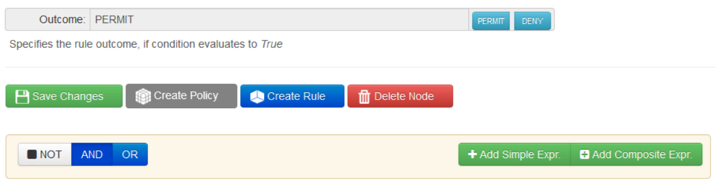
Figure 29 ABAC rule for checking whether it is not a working day


Outcome: DENY

Specifies the rule outcome, if condition evaluates to *True*

DateTime × isWorkHour × = True [✓ False]

Figure 30 ABAC rule for checking whether it is not a working hour


Outcome: PERMIT

Specifies the rule outcome, if condition evaluates to *True*

Figure 31 ABAC rule that permits access if none of the previous rules match

The ABE policy (used to protect certain resources) requires that only Pathologists can access the resource. Again warnings must be issued.

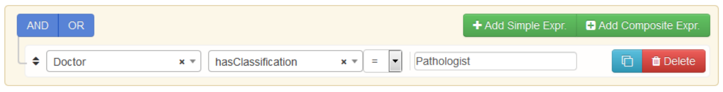
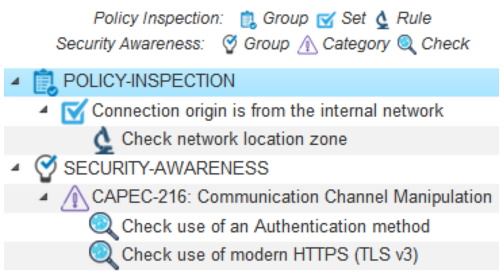

Doctor × hasClassification × = Pathologist

Figure 32 ABE policy requiring that only a Pathologist can access protected resource

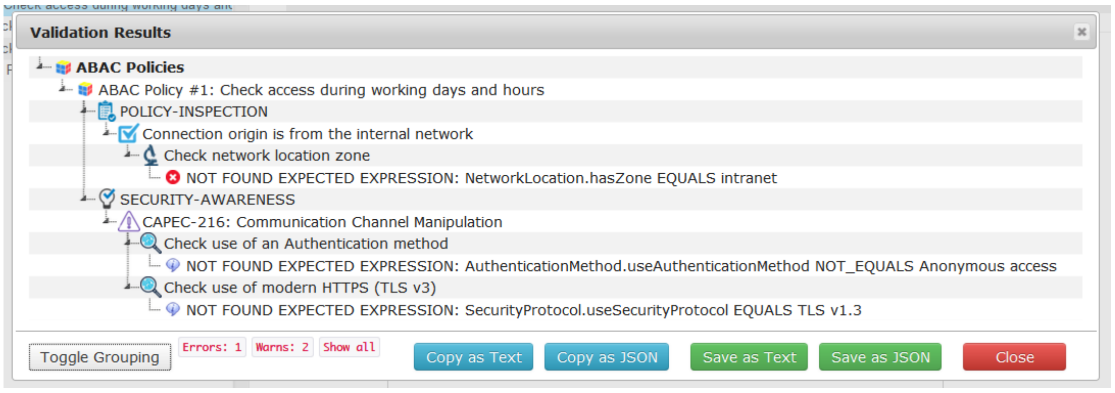
5.3.1.4.3 Policy validation results

Before being put into effect, the authorization policies are checked against the validation rules presented before. The validation results for each policy are presented in the following figures.



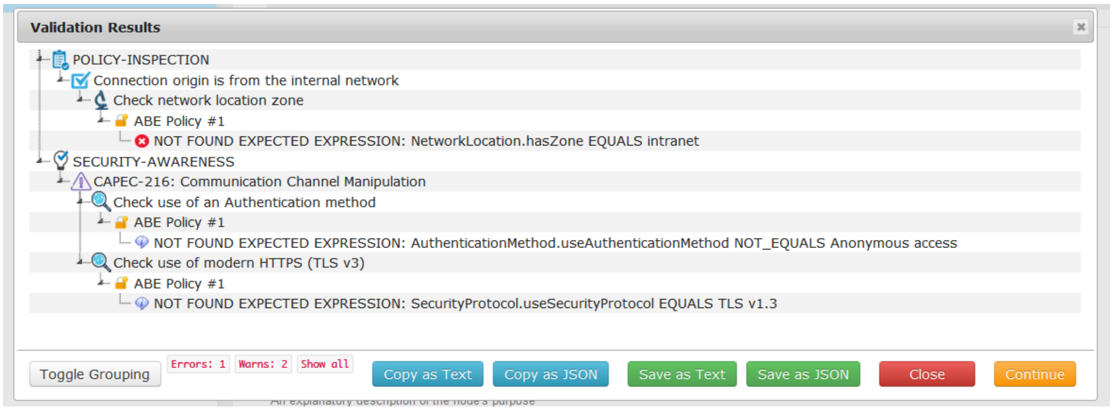
```
Policy Inspection:   Group  Set  Rule
Security Awareness:   Group  Category  Check
◢ POLICY-INSPECTION
  ◢ Connection origin is from the internal network
      Check network location zone
◢ SECURITY-AWARENESS
  ◢ CAPEC-216: Communication Channel Manipulation
      Check use of an Authentication method
      Check use of modern HTTPS (TLS v3)
```

Figure 33 Policy validation sets and rules



```
Validation Results
ABAC Policies
  ABAC Policy #1: Check access during working days and hours
    POLICY-INSPECTION
      Connection origin is from the internal network
        Check network location zone
          NOT FOUND EXPECTED EXPRESSION: NetworkLocation.hasZone EQUALS intranet
    SECURITY-AWARENESS
      CAPEC-216: Communication Channel Manipulation
        Check use of an Authentication method
          NOT FOUND EXPECTED EXPRESSION: AuthenticationMethod.useAuthenticationMethod NOT_EQUALS Anonymous access
        Check use of modern HTTPS (TLS v3)
          NOT FOUND EXPECTED EXPRESSION: SecurityProtocol.useSecurityProtocol EQUALS TLS v1.3
```

Figure 34 Policy validation results for ABAC policy



```
Validation Results
POLICY-INSPECTION
  Connection origin is from the internal network
    Check network location zone
      ABE Policy #1
        NOT FOUND EXPECTED EXPRESSION: NetworkLocation.hasZone EQUALS intranet
SECURITY-AWARENESS
  CAPEC-216: Communication Channel Manipulation
    Check use of an Authentication method
      ABE Policy #1
        NOT FOUND EXPECTED EXPRESSION: AuthenticationMethod.useAuthenticationMethod NOT_EQUALS Anonymous access
    Check use of modern HTTPS (TLS v3)
      ABE Policy #1
        NOT FOUND EXPECTED EXPRESSION: SecurityProtocol.useSecurityProtocol EQUALS TLS v1.3
```

Figure 35 Policy validation results for ABE policy (sorted by validation rule)

Once the policy is considered valid, then it is translated through the Policy-to-XACML Interpreter component of AMPLE and sent to the the Policy Decision Point (PDP) of ABAC Enforcement Mechanism in order to be enforced.

6 Conclusions

The goal of the deliverable was to the present the ASCLEPIOS Context-Aware Authorization Engine. An exhaustive analysis of the concept of authorization was presented. The theoretical principles of generalized “Access Control Mechanisms” were presented along with the similarities and differences of the most prominent proposals. Each proposal is a best-fit for a specific set of use-cases. For example, MAC and DAC are ideal for operating system security aspects (system calls, etc.). ACLs are ideal for perimeter-networking use-cases where all rules are fixed.

However, it could be argued that ACLEPIOS use-cases lean towards the ABAC schemes. The reason for that is that ABAC allows the dynamic formulation of policies based on arbitrary attributes of the Subject, Object and the Environment. Moreover, these policies can be authored in a distributed fashion. Above all, ABAC allows the complete separation of policy formulation with policy enforcement. There is a penalty that has to be paid for these merits. The penalty has to do with efficiency and complexity-of-applicability.

There are several standards that attempt to implement an ABAC scheme. Among the most notable ones we selected XACML because of two reasons a) industrial-adoption and b)open source community that supports these standards.

An ABAC engine is a cornerstone of ASCLEPIOS Context-Aware Authorization Engine; yet it is not the only component. ASCLEPIOS Context-Aware Authorization Engine is using a combination of ABAC with Attribute-Based-Encryption in order to augment the authorization functionality in a distributed cloud environment. These two schemes are complemented by an identity management scheme that is used to abstract the extraction of authentication info.

Hence the Context-Aware Authorization Engine is efficiently combining **OpenIDConnect signalling** for Identity Extraction (user authentication), **ABAC Policy Enforcement** for accessing allowing/disallowing access to an **ABE Server** that issues attribute-based encryption/decryption keys.

The authorization flow is analytically described for the two main use-cases, which are the encryption of a resource and the decryption of the resource. One of the most significant issues that relate to the developed mechanisms is how to lower the barrier of an adopter. The most difficult/error-prone aspect of the context-aware engine relates to the definition of attributes and policies. As such, **a dedicated authoring environment** has been implemented that aims to lower this barrier for adopters.

.

7 References

1. Y., Verginadis et al., 2019. D3.1 ASCLEPIOS Security and Policies Model. ASCLEPIOS Deliverable
2. Y., Verginadis et al., 2020. D3.2 ASCLEPIOS Models Editor and Interpretation Mechanism. ASCLEPIOS Deliverable
3. R., G., Roessink et al., 2020. D2.2 Attribute-Based Encryption, Dynamic Credentials and Ciphertext Delegation and Integration in Medical Devices. ASCLEPIOS Deliverable
4. A., Michalas et al., 2019. D1.2 ASCLEPIOS Reference Architecture, Security and E-health Use Cases, and Acceptance Criteria. ASCLEPIOS Deliverable
5. NIST, 2014, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, https://doi.org/10.6028/NIST.SP.800-162
6. eXtensible Access Control Markup Language (XACML) Version 3.0, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
7. NGAC standard (ANSI499), https://webstore.ansi.org/standards/incits/incits4992018
8. OASIS XACML Technical Committee, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
9. W3C XML Schema Definition Language (XSD), https://www.w3.org/TR/xmlschema11-1/
10. A Comparison of Attribute Based Access Control (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-178.pdf)
11. WSO2 Identity Server, https://wso2.com/identity-and-access-management/
12. Apache Thrift, https://thrift.apache.org/
13. ApacheDS, https://directory.apache.org/apacheds/
14. PaaSword, https://paasword.io/
15. AuthzForce, https://authzforce.ow2.org/
16. KeyCloak, https://keycloak.org
17. OAuth 2.0, https://oauth.net/2/
18. OpenIDConnect, https://openid.net/connect/