Attributed Based Access Control (ABAC) for Web Services

Eric Yuan, Jin Tong
Booz Allen Hamilton, Inc., McLean, Virginia
yuan_eric@bah.com, tong_jin@bah.com

Abstract

For companies and government agencies alike, the emergence of Web services technologies and the evolution of distributed systems toward Service Oriented Architectures (SOA) have helped promote collaboration and information sharing by breaking down "stove-piped" systems and connecting them via loosely coupled, interoperable system-to-system interfaces. Such architectures, however, also bring about their own security challenges that require due consideration.

Unfortunately, the current information security mechanisms are insufficient to address these challenges. In particular, the access control models today are mostly static and coarsely grained; they are not well-suited for the service-oriented environments where information access is dynamic and ad-hoc in nature. This paper outlines the access control challenges for Web services and SOA, and proposes an Attribute Based Access Control (ABAC) model as a new approach, which is based on subject, object, and environment attributes and supports both mandatory and discretionary access control needs.

The paper describes the ABAC model in terms of its authorization architecture and policy formulation, and makes a detailed comparison between ABAC and traditional role-based models, which clearly shows the advantages of ABAC. The paper then describes how this new model can be applied to securing web service invocations, with an implementation based on standard protocols and open-source tools. The paper concludes with a summary of the ABAC model's benefits and some future directions.

1. Introduction

The emergence of Web service technologies has enabled previously "stove-piped" information systems to interoperate in a platform-independent fashion, promoting unprecedented collaboration and information sharing, often across enterprise and network boundaries. In the commercial sector, for instance, web services are increasingly being used for "deep" supply chain integration; in the government sector, web services is a key technology enabler for agencies to increase joint situation awareness in counter-terrorism operations.

Sharing information in such service-oriented, highly dynamic environments, however, poses some daunting security challenges. On one hand, collaborative systems require making information accessible to all who need it; on the other hand, organizations must continue to protect sensitive and confidential information, allowing only authorized personnel to retrieve and manipulate them. Balancing the two competing goals is difficult under today's information security mechanisms and practices, and here are some of the challenges:

- Many systems today still rely heavily on perimeter-based security such as Demilitarized Zones (DMZ), firewalls, and intrusion detection to thwart security threats. In a truly service-oriented architecture (SOA), however, such measures are far from adequate. For instance, allowing access to web servcices via standard Internet protocols such as HTTP opens up servers to potential attacks which may not be detectable by conventional firewall products.

- For traditional information systems, a system provider need only deal with a set of known users, and a user need only access a set of known systems. In a SOA environment, however, the relationship between service consumers and providers is often not established a priori, but more ad-hoc and dynamic in nature – consumers can discovery new providers on-the-fly and make use of their data in real-time. It is impossible to fully predict what data should be shared, when and to whom. Most existing security models are not designed to support this need.

- Most security models currently in place are too simple, static, and coarse-grained to provide the rich semantics for web service level access control

Proceedings of the IEEE International Conference on Web Services (ICWS'05)
0-7695-2409-5/05 $20.00 **IEEE**

policies. For example, many systems simply authorize access based on a small handful of roles such as "user", "admin", and "guest". Few of them take operational contexts into consideration in their access control policies, which may include factors such as the transaction at hand, the current threat level, or the residing "community of interest".

To address these challenges, a number of industry standards have been or are being developed to facilitate the web service level security tasks such as identification, authentication, and authorization. Among these standards:

- The Security Assertions Markup Language (SAML) [1] defines a framework for exchanging security information in XML format. Security information such as authentication artifacts, authorization decisions, and subject attributes are represented in XML constructs called "assertions". The SAML specification also defines the protocol, transport bindings, and usage profiles for exchanging the assertions.

- The XML Access Control Markup Language (XACML) [2] defines a generic authorization architecture and a policy language for expressing and exchanging access control policy information using XML. Like SAML, XACML also provides a request / response semantics for authorization decisions to facilitate access control mechanisms. As described in later sections, XACML serves as a good vehicle for attribute-based access control and fits nicely with our architecture.

- Web Services Security (WSS) [3] serves as the foundation to address the Simple Object Access Protocol (SOAP) level security issues, with three major propositions: (1) use of security tokens in SOAP headers for user identity and authentication, (2) use of XML Digital Signature (XML-DSIG) standard for message integrity and authenticity, and (3) use of XML-Encryption (XML-ENC) for message confidentiality. WS-Security is only the first in a series of emerging standards proposed to provide a broader security framework for Web Services. Additional standards and vendor proposals are forthcoming that address issues such as privacy, policy, trust, secure conversation, and federation.

It is worth noting, however, that these standardization efforts have focused largely on defining the *wire formats* needed for security information exchange. The specifications are yet to address the challenge of defining the service-level semantics by which different parties interface with each other to achieve end-to-end security goals such as authentication and authorization.

In this paper we focus on the authorization and access control aspects of web services, and explain how an attribute based access control model can provide richer, more flexible semantics to protect web services. The rest of the paper is organized as follows:

To set the background, Section 2 provides a brief overview of existing access control models. Section 3 introduces the Attribute Based Access Control model in terms of its architecture and policy formulation. Section 4 and 5 compares ABAC with the traditional RBAC model in details. In Section 6, the ABAC model is applied in the web service security setting. Section 7 concludes the paper.

2. Existing Access Control Models

Before we summarize the existing mechanisms for restricting access to resources, it is worth pointing out that these mechanisms generally fall under one (or sometimes a combination) of two forms, discretionary or mandatory:

- *Discretionary Access Control* (DAC) is a way of restricting access to objects based on the identity and need-to-know of the user, process, and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject [4]. DAC is a common form of access control used in many systems and networks. For example, in the Windows operating system, the owner of a file or directory can grant or deny access to other users or groups of users.

- *Mandatory Access Control* (MAC) was originally formalized by Bell and LaPadula [5]. MAC is a way of restricting access to objects based on fixed security attributes or "labels" assigned to users as well as objects. The controls are mandatory in the sense that they are system-enforced and cannot be modified by users or their programs [6]. The need for MAC arises when the security policy of a system dictates that the resource owner must not make authorization decisions, a requirement often found in government and defense information systems. In practice, a label on a resource / object is usually called a security classification, while a label on a user is called a security clearance. For example, a document labeled as "X" may be readable only by someone with a clearance "Y" that can access X. Ideally, the system will prevent any data transfer outside those constraints, a feature called the ★-property (read as star-property) [7].

Note that MAC and DAC are not mutually exclusive. In fact, many systems use MAC and DAC in conjunction. In this case, mandatory controls are necessary but not sufficient conditions for the indicated access. They can be checked first, followed by a check of the discretionary controls. Alternatively, mandatory controls can kick in after the discretionary checks are satisfied.

There are many access control models that implement DAC and / or MAC. We now briefly describe three models that are used most often, and at the same time point out their deficiencies in meeting today's web service security needs.

2.1. Identity Based Access Control (IBAC)

Under this model, permissions to access a resource is directly associated with a subject's identifier (e.g., a user name). Access to the resource is only granted when such an association exists. An example of IBAC is the use of Access Control Lists (ACL), commonly found in operation systems and network security services. An ACL contains a list of users and their access rights such as read, write, or execute. Albeit easy to implement, the drawbacks of IBAC are obvious: the number of identifiers in the ACL will increase and become difficult to maintain as more users request access, making this approach impossible to scale. Furthermore, access control decisions are not based on any business function or characteristics of the user but solely on the identifiers, making it unsuitable for enterprise level use.

2.2. Role Based Access Control (RBAC)

The RBAC model [8] restricts access to a resource based on the business function or role the subject is performing. The permissions to access a resource are then assigned to the appropriate role(s), rather than directly assigned to subject identifiers. Compared with IBAC, RBAC adds levels of indirection between identities and resources. Because permissions no longer need to be repeatedly assigned to individual users, RBAC scales much better and significantly reduces administration overheads. Furthermore, RBAC maps intuitively to the way business roles and responsibilities are managed, and is therefore easy to understand and use. Although RBAC may take slightly different forms, a common representation as defined in [9] is depicted in Figure 1.

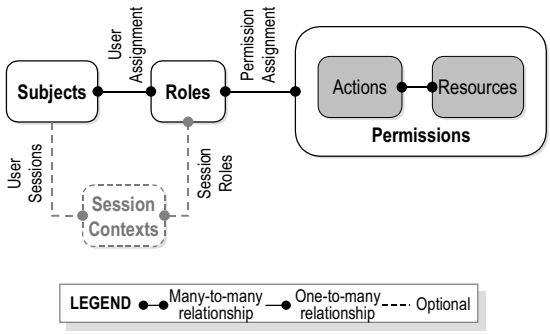

Figure 1: Core RBAC Model

IBAC an RBAC are primarily DAC mechanisms; they are not best suited to address MAC requirements: With IBAC, there is no way to associate security labels with users. With RBAC, unique roles have to be created for all combinations of security labels, which is clearly undesirable.

2.3. Lattice Based Access Control (LBAC)

The lattice-based models were proposed in the 1970s and since then have been implemented mostly in the U.S. defense sector and its allies to address the MAC problem. Under an LBAC, a totally ordered set of security labels (e.g., TS > S > C > U) are combined with a set of categories (e.g., X, Y, Z) to form a "lattice", allowing for the representation of a set of security classes ranging from system-low to system-high using a much smaller number of security labels [7].

Generally speaking, the LBAC model is manageable when there are a relatively small number of static security labels and categories (as the total combinations of labels and categories is potentially n*m), and therefore is only effective for certain coarsely-grained security scenarios and lacks flexibility and scalability.

The above access control models, though effective in certain situations, are not sufficient to describe the complex, fine-grained access control policies in today's collaborative environments. Take RBAC for example: because the core RBAC model limits the abstraction of user function to roles only, it does not consider any other characteristics that a user may demonstrate. Further, RBAC generally doesn't take into account the characteristics of resources (other than their identifiers); nor does it capture any security-relevant information of the environment.

The goal of a well-engineered web services security architecture is to be flexible enough to accommodate a

variety of users, services, actions and policies such that it is extensible to a wide range of business use scenarios. Limitations of existing access control mechanisms suggest the next evolutionary step in access control should work in terms of the relevant characteristics of subjects, resources, and operational contexts, rather than solely in terms of identifiers or domain specific roles. Such an attribute-based approach is introduced in the next section.

3. Attribute Based Access Control

The Attribute Based Access Control (ABAC) model consists of two aspects: the *policy model* which defines the ABAC policies, and the *architecture model* which applies the policies to web services access control. After explaining the concepts of attributes, we first introduce the policy model, and then describe the architecture model.

3.1. Attributes Defined

Unlike IBAC and RBAC, the ABAC model can define permissions based on just about any security-relevant characteristics, known as *attributes*. For access control purposes, we are concerned with three types of attributes:

- *Subject Attributes*. A subject is an entity (e.g., a user, application, or process) that takes action on a resource. Each subject has associated attributes which define the identity and characteristics of the subject. Such attributes may include the subject's identifier, name, organization, job title, and so on. A subject's role, naturally, can also be viewed as an attribute.

- *Resource Attributes*. A resource is an entity (e.g., a web service, data structure, or system component) that is acted upon by a subject. As with subjects, resources have attributes that can be leveraged to make access control decisions. A Microsoft Word document, for example, may have attributes such as title, subject, date, and author. Resource attributes can often be extracted from the "metadata" of the resource. In particular, a variety of web service metadata attributes may be relevant for access control purposes, such as ownership, service taxonomy, or even Quality of Service (QoS) attributes.

- *Environment Attributes* have so far been largely ignored in most access control policies. These attributes describe the operational, technical, and even situational environment or context in which the information access occurs. For example, attributes such as current date and time, the current virus / hacker activities, and the network's security level (e.g., Internet vs. Intranet), are not associated with a particular subject nor a resource, but may nonetheless be relevant in applying an access control policy.

By treating role and identity as characteristics of a principal, ABAC fully encompasses the functionality of both IBAC and RBAC approaches. Moreover, its ability to consider additional resource attributes such as sensitivity information enables it to tackle MAC requirements just as easily. Therefore, we believe that the ABAC is the natural convergence of existing access control models and surpasses their functionality. Policy representation is semantically richer and more expressive, and can be more fine-grained within ABAC because it can be based on any combination of subject, resource, and environment attributes.

3.2. ABAC Policy Formulation

Here we formally define the (basic) ABAC policy model:

(1) S, R, and E are subjects, resources, and environments, respectively;

(2) SA_k ($1 \le k \le K$), RA_m ($1 \le m \le M$), and EA_n ($1 \le n \le N$) are the pre-defined attributes for subjects, resources, and environments, respectively;

(3) ATTR(s), ATTR(r), and ATTR(e) are *attribute assignment* relations for subject s, resource r, and environment e, respectively:

$$ATTR(s) \subseteq SA_1 \times SA_2 \times \ldots \times SA_K$$

$$ATTR(r) \subseteq RA_1 \times RA_2 \times \ldots \times RA_M$$

$$ATTR(e) \subseteq EA_1 \times EA_2 \times \ldots \times EA_N$$

We also use the function notation for the value assignment of individual attributes. For example:

Role(s) = "Service Consumer"

ServiceOwner(r) = "XYZ, Inc."

CurrentDate(e) = "01-23-2005"

(4) In the most general form, a *Policy Rule* that decides on whether a subject s can access a resource r in a particular environment e, is a Boolean function of s, r, and e's attributes:

Rule: can_access (s, r, e) ←

$$f\ (ATTR(s), ATTR(r), ATTR(e))$$

Given all the attribute assignments of s, r, and e, if the function's evaluation is true, then the access to the resource is granted; otherwise the access is denied.

(5) A Policy rule base or *Policy Store* may consist of a number of policy rules, covering many subjects and resources within a security domain. The access control decision process in essence amounts to the evaluation of applicable policy rules in the policy store.

Now we show how different policy rules may be formulated using attributes. One of the simplest yet commonly used forms of policy rules involve subject, resource, and environment attributes that are *independent* of one another. In this case, the policy rules usually match subject / resource / environment attributes to constant values. For example, a rule that dictates "Users with role 'Manager' may access the 'ApprovePurchase' web service" can be written as:

R1: can_access (s, r, e) ←

(Role(s) = 'Manager') ∧

(Name(r) = 'ApprovePurchase')

This example also shows that conventional RBAC rules can be accommodated by the ABAC model just as easily. However, the ABAC model is capable of modeling much richer access control semantics. For example, to enforce that a resource may only be accessed by its owners, we can have a rule as follows:

R2: can_access (s, r, e) ←

(UserID(s) = ResourceOwner(r))

More interesting examples will be given in the next section.

Given attribute assignments, the evaluation of policy rules may be boiled down to the evaluation of First Order Logic (or its simpler variants) expressions, and existing algorithms (e.g., forward chaining) can be readily applied. However, it is beyond the scope of this paper to discuss the actual implementation of policy evaluation.

In this basic definition, we also assume that the attributes are single-valued, and that the policy decision is based on only one rule. The model can be extended to support more flexible features, such as multi-valued attributes and combining algorithms for multiple rules.

3.3. Authorization Architecture

A typical attribute-based authorization architecture is illustrated in Figure 2 below.

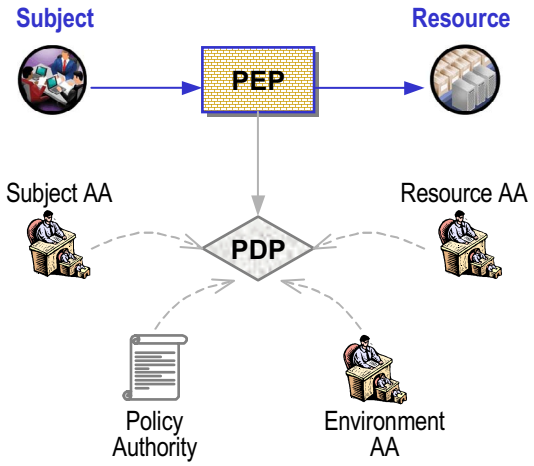

Figure 2: ABAC Authorization Architecture

The diagram reflects the following logical actors involved in an ABAC model:

- The *Attribute Authorities* (AA) are responsible for creating and managing the attributes for subjects, resources, and the environment, respectively. As a logical entity, an AA may or may not store the attributes by itself (e.g., a Subject AA may choose to retrieve attributes from the organization's LDAP directory), but it is responsible for binding attributes to an entity of interest, and plays an important role in the provisioning and discovery of attributes.

- The *Policy Enforcement Point* (PEP) is responsible for requesting authorization decisions and enforcing them. In essence, it is the point of presence for access control and must be able to intercept service requests between information consumers and providers. Although the diagram depicts the PEP as a single point, it may be physically distributed throughout the network. The most important security engineering consideration for the implementation of a PEP is that the system must be designed such that the PEP cannot be bypassed in order to invoke a protected resource.

- The *Policy Decision Point* (PDP) is responsible for evaluating the applicable policies and making the authorization decision (permit or deny). The PDP is in essence a policy execution engine. When a policy references a subject, resource, or an environment attribute that is not present in the request, it contacts the appropriate AA to retrieve the attribute value(s).

- The *Policy Authority* (PA) creates and manages access control policies. The policies may consist of decision rules, conditions, and other constraints for accessing the resources.

This model is agnostic to the actual representation and format of the policies. As mentioned earlier, XACML is an XML-based policy language that is powerful enough to support ABAC.

4. ABAC vs. RBAC

Since many access control solutions today are role based, in this section we compare ABAC with the latest RBAC approaches to illustrate the advantages of this new model.

We use an example that involves a slightly more complex access control scenario to show how RBAC and ABAC attack the problem differently. The scenario is inspired by the Online Entertainment Store example presented in [10].

In our example, an Online Entertainment Store streams movies to users for a flat monthly fee. The store needs to enforce an access control policy that is based on the users' age and the movie content ratings. The movie ratings and the corresponding access control policy is listed as follows:

Movie Rating	**Users Allowed**
R	*Age 21 or older*
PG-13	*Age 13 or older*
G	*Everyone*

In the standard RBAC model, where only user roles are considered, there would be three pre-defined roles created for users, *Adult*, *Juvenile*, and *Child*. Each user of the store would be assigned to one of the three roles, possibly during registration. There would be three permissions created, namely, *Can view R rated movies*, *Can view PG-13 rated movies* and *Can view G rated movies*, each represent the permission to view movies with the respective rating. The A*dult* role gets assigned with all three permissions, where as the *Juvenile* role gets *Can view PG-13 rated movies* and *Can view G rated movies* permissions, and the *Child* role gets the *Can view G rated movies* permission only. Both the user-to-role assignment and the permission-to-role assignment are manual administrative tasks.

In comparison, the ABAC approach in this scenario has no need to explicitly define roles. Instead, whether a user u can access or view a movie m (in a security environment e which is ignored here) would be resolved by evaluating a policy rule such as the following:

$$\begin{aligned} &\text{R3: can\_access}(u, m, e) \leftarrow \\ &\quad (\text{Age}(u) \geq 21 \wedge \text{Rating}(m) \in \{\text{R,PG-13,G}\}) \vee \\ &\quad (\text{Age}(u) \geq 13 \wedge \text{Age}(u) < 21 \wedge \\ &\qquad\qquad \text{Rating}(m) \in \{\text{PG-13,G}\}) \vee \\ &\quad (\text{Age}(u) < 13 \wedge \text{Rating}(m) \in \{\text{G}\}) \end{aligned}$$

Where Age and Rating are the subject attribute and the resource attribute, respectively. The advantage of the ABAC model shown here is that it eliminates the definition and management of static roles, hence also eliminates the need for the administrative tasks for user-to-role assignment and permission-to-role assignment.

Finer-grained access control policies often involve multiple subject and object attributes. In such cases ABAC becomes more manageable and scalable than RBAC. To illustrate this, we extend the example slightly: suppose movies are also categorized either as *New Release* or as *Old Release* based on release dates, and users are further classified as *Premium User* and *Regular User* based on the membership fee paid. Suppose the store would like to enforce a policy such that only premium users can view new releases.

In RBAC, the roles and permissions that need to be created have both doubled:

Standard RBAC Roles
Adult premium user role
Adult regular user role
Juvenile premium user role
Juvenile regular user role
Child premium user role
Child regular user role

Standard RBAC Permissions
Can view R rated new release
Can view R rated old release
Can view PG-13 rated new release
Can view PG-13 rated old release
Can view G rated new release
Can view G rated old release

In general, if there are K subject attributes and M resource attributes, and if for each attribute, Range() denotes the range of possible values it can take, then the respective number of roles and permissions need to be created would be:

$$\prod_{1}^{K} Range(SA_k)$$

and

$$\prod_{1}^{M} Range(RA_m)$$

Therefore, as the policy becomes finer-grained and more attributes are involved, the number of roles and permissions will grow exponentially, making the user-to-role assignment and permission-to-role assignment tasks prohibitively expensive.

In the ABAC model, the previous policy rule R3 still applies; we only need to add a second rule R4, and then combine the two conjunctively:

R4: can_access(u, m, e) ←

(MembershipType(u) = 'Premium') ∨

(MembershipType(u) = 'Regular' ∧
MovieType(m) = 'OldRelease')

R5: can_access(u, m, e) ←

R3 ∧ R4

So far we haven't addressed environment attributes. The RBAC model does not address environment attributes explicitly. For example, it would be even more awkward to implement a policy that states "*Regular users are allowed to view new releases in promotional periods.*" By contrast, in an ABAC model, this policy can be implemented simply by adding conjunctive sub-clauses to check to see if *today's date*, an environment attribute, falls in a promotional period.

5. Other Related Work

Some research work has tried to address various issues in RBAC. The Rule-Based RBAC (RB-RBAC) model [10] tries to amend the tedious manual user-to-role assignment problem in standard RBAC models by specifying user-to-role assignment rules based on user attributes and dynamically assign users to pre-defined roles user attributes. The user-to-role assignment rules for the previous online store example would be expressed as:

If Age >= 21, then assign role { Adult }

If Age >= 13 and Age < 21, then assign role { Juvenile }

If Age < 13, then assign roles { Child }

By evaluating these rules using a particular user's corresponding attributes, the user-to-role assignment software eliminates the manual administrative user-to-role assignment. RB-RBAC, however, does not directly address the explosion of roles and permissions shown in the previous section, because the number of rules will also increase exponentially as the number of attributes increase.

In [11], Barkley et al. presented an approach that includes the relationship between resource and subject in the access policy model. They also included, in parallel, a standard RBAC policy evaluation as part of the overall access decision process. Barkley et al. argued that the inclusion of subject-resource relationship is more scalable than de-composing the relationship into individual roles that need be created for each of such relationship in a traditional RBAC model. For example, an access policy may simply be expressed as *"a patient X's records may be updated by a physician Y who is in the attending-physician relationship with X."* In traditional RBAC models, such a policy will have to involve the creation of an *Attending-Physician* role for each patient. The ABAC model described in this paper, however, is able to accommodate subject-resource relationships in the policy function formulation, similar to the example policy R2 in Section 3.3.

[12] and [13] described a system called *Atenti* where different stakeholders can specify its own policies as use conditions using user attributes as building blocks. These use-condition policies and user attributes are provided in the system using policy and attribute stores. Furthermore, digital certificates are used to establish the trust of the policies and attributes. The system allowed distributed attribute authorities and policy stakeholders to participate in the access decision process, which reflects a real world problem where access decision may not be decided by a single actor. This is a shared motivation for the ABAC model in a widely distributed Web services environment as well.

There are many other research efforts that attempt to extend the RBAC model in various ways to support service-oriented information sharing, ranging from role delegation [14] to role hierarchy certificates [15] to XML-based policies [16]. Most of these efforts, however, are constrained by the inherent limitations of RBAC described above.

6. Securing Web Services with ABAC

6.1. Logical Architecture

In this section we describe how web service access control decisions may be achieved using ABAC policies. Figure 3 depicts a generic architecture for attribute-based web service security:

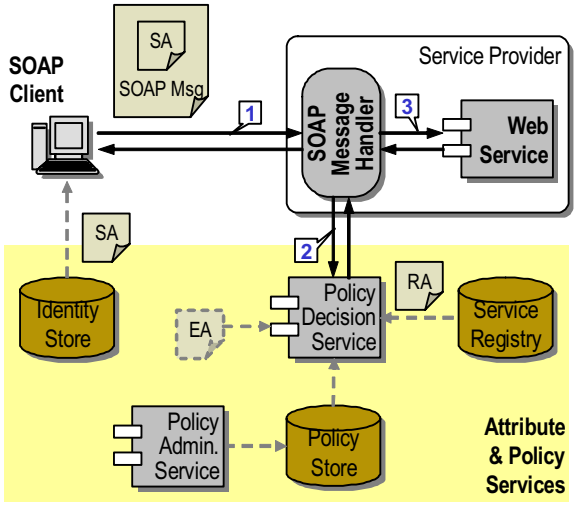

Figure 3: ABAC Service Security Architecture

The diagram depicts a simple web service invocation scenario involving the following steps:

(1) A Web service (SOAP) client application fetches the appropriate subject attributes from a Subject Attribute Authority (e.g., an LDAP-based identity store). The SOAP client application sends the SOAP request message that includes the fetched subject attributes in the SOAP header, e.g., in the form of SAML Attribute Assertions;

(2) A SOAP Message Handler that acts as the PEP picks up the SOAP request message and makes a call to the PDP for an authorization decision. In this case, the PDP is a Policy Decision Service (PDS), which is a Web service in its own right. The PDS fetches needed resource attributes for the target Web service from a Service Registry, which acts as the Resource Attribute Authority. With the subject attributes, resource attributes, and possibly environment attributes, the PDS then evaluates the policy rules it fetched from a policy store. A *permit* or *deny* decision is rendered and sent back to the SOAP Message Handler.

(3) With a *permit* decision, the SOAP message handler passes the original SOAP request to the actual Web service endpoint.

6.2. Implementation

Our implementation of this architecture, developed for a government agency, uses of a set of JAX-RPC message handlers to intercept SOAP request messages. These JAX-RPC message handlers constitute the PEP. The implementation is based on the Apache Axis SOAP engine [17].

The Policy Decision Service is also implemented using Apache Axis. It is envisioned to be used by many Web services protected by their PEPs.

The Service Registry in our implementation is a UDDI registry that holds service metadata, including attributes that are security-relevant such as service type and service provider.

The policy store contains all the ABAC policies. The policies are stored and retrieved in XACML format, and evaluated using an open-source XACML processor.

Each Web Service call (request and response) within the above scenario is digitally signed to provide sender authentication and message integrity.

Caching is implemented at various places to optimize system performance.

7. Conclusion

As shown in this paper, the ABAC model brings out many advantages over traditional identity or role based models:

- It is intuitive to model and manage real-world access control policies;
- It is more flexible and more powerful to describe complex, fine-grained access control semantics, which is especially suitable for the dynamic, ad-hoc environments for Web services;
- Under ABAC, the management of security information is spread over a number of Attribute and Policy Authorities, which can be distributed over the network, even across organizational boundaries – also something that is suited for the service-oriented architectures of future enterprises;
- Similarly, this "divide and conquer" approach significantly reduces overall system complexity, allowing different system actors (User Directory, Service Registry, Policy Server, etc.) to focus on their respective administrative needs.

The ABAC model, though very powerful, only focuses on authorization of requests from information consumers to providers. An end-to-end security architecture, however, is concerned not just with the access control model. Rather, it covers the whole set of components, interfaces, and data that allows authorization decisions to be made and enforced. To utilize the ABAC model to its full potential, many other aspects of the entire attribute management "life

cycle" needs to be considered, such as attribute provisioning, cryptographic binding of attributes to entities, attribute discovery, and the feedback loop on attribute usage. These are all potential research and engineering topics to be explored.

8. References

[1] OASIS, The Security Assertions Markup Language (SAML) OASIS TC Homepage, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[2] OASIS, The XML Access Control Markup Language (XACML) OASIS TC Homepage, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[3] OASIS, Web Services Security (WSS) OASIS TC Homepage, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[4] National Computer Security Center (NCSC), "Glossary of Computer Security Terms" (NCSC-TG-004), October 21, 1988, http://csrc.nist.gov/secpubs/rainbow/tg004.txt

[5] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", *Mitre Corp. Report No. M74-244*, Bedford, MA, 1975

[6] William Stallings, *Network and Internetwork Security: Principles and Practice*, Prentice Hall, 1995

[7] Ravi S. Sandhu, "Lattice-Based Access Control Models", *IEEE Computer*, November 1993, pp. 9-19

[8] Ravi S. Sandhu et al., "Role-Based Access Control Models", *IEEE Computer*, February 1996, pp. 38-47

[9] Rick Kuhn et al., "Information Technology - Role Based Access Control", ANSI Standard, Document Number: ANSI/INCITS 359-2004, Feb. 03, 2004

[10] Mohammad A. Al-Kahtani and Ravi Sandhu, "Induced role hierarchies with attribute-based RBAC", *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, ACM Press, Como, Italy, 2003, pp. 142-148

[11] J. F. Barkley, K. Beznosov, and J. Uppal, "Supporting Relationships in Access Control Using Role Based Access Control", *ACM Workshop on Role-Based Access Control*, ACM Press, 1999, pp. 55-65

[12] W. Johnston, S. Mudumbai, and M. Thompson, "Authorization and Attribute Certificates for Widely Distributed Access Control", *Proceedings of the IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998, p. 340

[13] M. Thompson, W. Johnson, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiar, "Certificate-based Access Control for Widely Distributed Resources", *Proceedings of 8th USENIX UNIX Security Symposium*, USENIX Association, Washington D.C., 1999, pp. 215-227

[14] G. Ahn and B. Mohan, "Secure Information Sharing Using Role-based Delegation", *International Conference on Information Technology: Coding and Computing (ITCC'04) Proceedings*, August 2004

[15] G. Denker and J. Millen, "Cross-Domain Access Control via PKI", *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02) Proceedings*, April 2002

[16] R. Bhatti, E. Bertino, and A. Ghafoor, "A Trust-based Context-Aware Access Control Model for Web-Services", *IEEE International Conference on Web Services (ICWS'04) Proceedings*, March 2004

[17] Apache Axis Project Home Page, http://ws.apache.org/axis