Access Control for the Services Oriented Architecture

Jun Li and Alan H. Karp
Hewlett-Packard Laboratories
Palo Alto, CA, USA
{jun.li, alan.karp}@hp.com

ABSTRACT
Federated Identity Management (FIdM) is being applied to Services Oriented Architecture (SOA) deployments that cross enterprise boundaries. Though federation is essential in order to address the distributed nature of SOA, these FIdM solutions have been found to be inflexible, unscalable, and difficult to use, manage, and upgrade. We contend that a major reason for these difficulties is that FIdM addresses the wrong aspect of the problem. Specifically, FIdM does not address the federation of access policies. What is needed is a system for Federated Access Management (FAccM). This paper demonstrates the benefits of FAccM over FIdM for SOA deployments and shows how FAccM can be implemented using the existing web services standards.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services– *Web-based Services.*

General Terms
Security, Management, Standardization

Keywords
Services Oriented Architecture; SOA; web services; access control; Federated Identity Management; FIdM

1. INTRODUCTION
The Services Oriented Architecture (SOA) [28, 35] may yet deliver on the promise of loosely coupled application development that didn't materialize from earlier attempts, such as CORBA [39, 15]. SOA can be based on the Web Services standards - UDDI for service discovery [38], WSDL for interface definition [40], and SOAP for invocation [33], all of which use XML [10] as the communications format. These standards remove many dependencies on application servers, operating systems, and machine architecture, making composition of independently developed components far easier.

One of the things holding back the widespread use of cross-domain SOA is the delay in reaching consensus on how to secure the service between different organizations. There are a number of aspects of securing web services, such as encryption, message integrity, authentication, authorization, etc., and there appears to be at least one standard for each of them, XML DSIG [41], XACML [9], etc. The relevant standard for a discussion of access control is the Security Assertion Markup Language (SAML) [27]. The goal of SAML is to provide a means for exchanging security information across organizational boundaries, a requirement if the SOA is to reach its full potential.

The SAML specification is quite general in the kind of assertions that can be made, but most of the examples include a specification of the user's identity. For example, the SAML Technical Overview [27] includes the statement, "At the heart of most SAML assertions is a ***subject*** (a principal – an entity that can be authenticated – within the context of a particular security domain) about which something is being asserted." The Liberty Alliance [22], which is developing a framework for distributed identity management, has adopted SAML 2.0, another indication of the importance of identity assertions in SAML.

It is no surprise, then, that most implementations based on the SOA tie access control decisions to the identity of the requester. This approach is spelled out in the introduction to the SAML specification [27], which states,

> "For example, a typical assertion from an identity provider might convey that 'This user is John Doe, he has an email address of john.doe@company.com, and he was authenticated into this system using a password mechanism.' A service provider could choose to use this information, depending on its access policies, to grant access to local resources."

Judging by the preponderance of talks at security conferences, such as RSA 2007 [31], most implementers assume that a Federated Identity Management (FIdM) framework is needed to associate an access policy with a given identity when crossing organizational boundaries. Based on the problems reported [18], organizations implementing these solutions are learning that federating access policies is much harder than federating identities.

The SAML specification does not indicate how the service provider is to use the identity of the requester to make access control decisions. Typically, the service uses the identity to look up the appropriate policy in some local database and bases the access decision on that information. So, it appears that the identity of the requester isn't the critical information; it is the authorization information in the database that matters. If that is indeed the case, why not have the request convey the authorization information instead of or in addition to the requester's identity? This paper shows that managing access policies using explicit authorizations with Federated Access

SWS'07, November 2, 2007, Alexandria, Virginia, USA.

Management (FAccM) is simpler than managing access policies using FIdM.

In the remainder of this paper, we'll explain why Identification-Based Access Control (IBAC) with FIdM is not a good solution, show the advantages of FAccM using Authorization-Based Access Control (ABAC), introduce the FAccM architecture, and show how to express FAccM with SAML assertions.

2. Current Approach

Today, most web services in a SOA implementation use Identification-Based Access Control (IBAC), often with Federated Identity Management (FIdM). It's quite familiar, but there are a number of problems.

Manageability: There is substantial management overhead. When employees change jobs, their companies must make sure their policies are updated. The result should be the employee losing some rights and gaining others. When using services provided by other organizations, they must request that the corresponding ACLs be updated. Enterprises can have10,000 or more business partners. Even small companies often have hundreds of customers. The cost of updating the access lists can be substantial.

The approach is unmanageable for users, too. Employees may work with dozens of business partners. They may well end up with a different credential for each of them. Worse, each might be based on a different technology, perhaps as similar as SAML 1.1 and SAML 2.0 or as different as X.509 and Kerberos. The employee will have to learn how to use each of them. The number of tools being developed to simplify things for users is proof that the problem is real. These tools include a variety of Single Sign-On (SSO) products [1, 30], and identity management tools, such as Card Space [24]. The problem is getting all the business partners each employee deals with to agree on a single approach.

Ambient authorities: Every request is accompanied by proof of the user's identity. The service searches its policy database entries for him looking for a match with the request. If a match is found, the request is honored. It is very hard for users to give processes running on their behalf a subset of their rights. That means a virus running in a user's browser can do anything that user is allowed to do. Single Sign-On exacerbates the problem by giving each process the user runs even more authority.

Delegation: Employees often need to delegate some tasks. Since the access control lists are tightly controlled, they must ask some party in authority to do the delegations on their behalf. If the delegation includes services controlled by other organizations, a responsible party in each organization needs to be involved. Since access is controlled by authentication, the employee receiving the rights being delegated needs credentials for each organization. This process is so onerous that people often share credentials, such as passwords and private keys [29].

Revocation: Delegated rights often need to be revoked. We can't simply revoke an employee's identity. It's used for too many things. Instead, we need to remove entries from the relevant access control lists. As with delegation, this process involves administrators from several organizations. One risk with this approach is that the employee whose right is being revoked might have had that right for a reason independent of the delegation. Simply removing the ACL entry will inappropriately revoke that user's right.

Responsibility tracking: We need to know who to hold accountable for service invocations for billing and to deal with misuse. The employee's company can audit the fact that the service was invoked. Depending on the configuration, they can track who made the request. However, tracking why the employee was granted the right is harder.

Confused deputy: This problem arises when rights from two parties need to be exercised in the same request [14]. The classic example is a service that compiles programs and updates a billing file (a log file). The service invocation takes two parameters, the name of the input file and the name of the output file. The attack involves specifying the log file where the service is expecting the name of the output file. The consequence is that the billing file gets corrupted. Variants of this attack are not rare [34].

Transitivity: It is common to invoke a service that needs to invoke a second service in order to satisfy the original service request. The question is whose rights get used for that second service request. In some cases, there is no right answer. Consider a service that copies files using the implementation outFile.write(inFile.read()). The original invocation provides the name of the input file. The subsequent (second) invocation provides in addition the name of the output file. The service request will fail unless there is one party with the permission to both read the input file and write the output file.

Policy compliance: Access policies are complicated and frequently change. Hence, security can not rely on all employees being aware of all the policies all the time. With IBAC, granting a right involves a system administrator who can deny requests that violate policy. Unfortunately most such requests are simple delegations that would not violate policy. People end up avoiding the delays of involving an administrator by sharing credentials [29], and the policy is violated.

3. Federated Access Management

In a conventional system based on Identification-Based Access Control, users present their identities along with their requests. The invoked service presents these identities to a policy engine and honors the request if the policy engine returns an authorization. Because authentications cross organization boundaries, Single Sign-On (SSO) [1, 30] and Federated Identity Management (FIdM) [22] solutions are often proposed to reduce the management burden.

Federated Access Management (FAccM) is based on Authorization-Based Access Control (ABAC) [18]. In ABAC users authenticate to their respective identity servers. Each identity server presents the identity to a policy engine that returns the authorizations appropriate for that user. Users present these authorizations along with their requests, and the service honors the request if the authorization is valid, e.g., has not expired or been forged. Since authentications need not cross organizations, there is no need for SSO or FIdM.

FAccM with ABAC has a number of advantages.

Manageability: Management overhead is reduced. There is no need to set up accounts at a variety of business partner sites. There is no need to manage the identities of employees of those

business partners. Each organization is free to grant and revoke privileges on its own.

Ambient authorities: There are none. Users may give the processes acting on their behalf exactly the subset of their rights that they want them to have.

Delegation: Delegation is the key to simplifying distributed policy management. Users manage their rights without needing to bother people who know nothing about their organizations. Easy delegation encourages users to delegate subsets of their rights to processes acting on their behalf, reducing their vulnerability to errors and malware.

Revocation: Revocation only involves the two end-points of the request. That reduces the delay in revoking access and simplifies the mechanism. Further, there is no danger that revoking the rights delegated to one party will affect the authorization that party received from someone else.

Confused deputy: Deputies cannot become confused because the authorization and the designation are combined. Each resource named is tied to exactly the intended set of rights.

Transitive access: Invoking a service involves using one's rights and delegating rights to the arguments. The example of outFile.write(inFile.read()) works because the original invocation delegates the right to read the input file, and the subsequent invocation delegates the authority to write the output file.

Policy Compliance: Delegation is easy, even if it violates policy. In the minority of cases where it is important, the request can be directed to a forwarder that can ensure policy is followed at an early stage, instead of being enforced later at the service. Since users delegate all rights in the same way, policy is enforced.

4. FAccM Architecture

Authorization-Based Access Control (ABAC) applies an organizing principle that has long been used in human interactions, delegation of rights and responsibilities. The characteristic of this principle that ABAC most closely embodies is making the chain of trust explicit. Federated Access Management (FAccM) extends ABAC across organizations. In this case, each link in the chain of trust can be viewed as a contract between two organizations.

The overall architecture is shown in Figure 1. We will first discuss ABAC within an organization, which refers only to the top half of the figure, and then show how extending it across organizations leads naturally to Federated Access Management.

4.1 Intra-organization

Within an organization O, there is a Domain Access Right Controller (DARC) and a policy engine. Authorizations are represented by certificates of the form Cert(A,B,R), where the issuer B is authorizing the recipient A to use the rights R. When a service S is deployed in O, the owner of S creates a self-issued certificate, Cert(S,S,*), which grants all rights to the service owner. The service owner then registers the service with the DARC, along with a certificate Cert(O,S,*) that delegates all operations on S to O. Later, when a subject U, which can be either an end-user or a service, asks the DARC for access to S, the DARC will check its policy engine. If access is approved, the DARC will delegate the appropriate subset of the rights in a certificate to U, Cert (U, O, R_U). Hence, the DARC represents the organization for issuing authorizations.

$$\text{Cert (O, S,*)} = \text{Delegate (O, Cert(S, S,*), S,*)}$$

$$\text{Cert (U,O,}R_U\text{)} = \text{Delegate (U, Cert (O, S,*), O, }R_U\text{)}$$

In the Delegate function above, the four parameters represent the delegatee, the source certificate, which serves as proof of the right to delegate, the delegator, and the subset of the rights in the source certificate that are being delegated. Restricting the rights being delegated allows the delegation of certificates authorizing different rights to different subjects (users). This approach decouples the service's granting of the right to use all of its operations from the enforcement of the organization's policy.

The user presents Cert (U,O,R_U) when invoking S. Note that the innermost certificate in the delegation chain, Cert(S,S,*), is what identifies S as the service being authorized. S, or a Policy Enforcement Point (PEP) acting on behalf of S, is responsible for verifying the validity of the authorization. This validation includes checking that the certificate is properly signed and that the signing key is still valid, that the request has been made during the validity period of the authorization, and that the delegations are proper. This last step includes verifying the above facts in addition to the fact that the rights being delegated are a (proper) subset of the rights of the delegator.

Many service invocations take arguments. Some of these arguments are pure data, but others include references to other services, say S_1, which S can invoke later. As a result, the full invocation path is U→S→S_1. Since the user does not know how S is implemented, the argument is expressed as a delegation to S of a (proper) subset of the user's rights to S_1, Cert(S,U,R_S). U gets the necessary information to delegate to S from the certificate authorizing U to use S. S now has the rights needed to invoke S_1. If S invokes S_1 with the delegation from U, the invocation cannot exceed U's authority to S_1. The rights that U delegates to S_1 are called the *transitive rights* to respond to the invocation of S.

Users can be less vulnerable to the software they use by taking advantage of this transitive rights pattern. They need not just start a process that has access to the user's private key, and thereby all the user's rights. Instead, the user creates a new public/private key pair, delegates the subset of the user's rights that the process needs to do the job the user wants done, and starts the process, passing in the new key pair and the delegated certificates. As a result, the process becomes a subject that works on behalf of the user with the limited rights specified in the delegated certificates.

Revocation is straightforward. When a user U leaves the organization, U's public key can be revoked using conventional means, such as an organization-wide Certificate Revocation List. That doesn't work well when U is changing jobs because only some of U's rights change. Instead, we need to revoke those rights U is not longer entitled to. With ABAC, any subject on the delegation chain can send a message to S revoking access to some Cert(U,*,R_U) granted to U.

Each service exposes a *revoke()* operation. The right to revoke the certificate is embedded in the certificate granted to the subject. A subject makes a request directly to the service specifying a certificate that the subject has delegated. The revocation request is honored if the requesting subject's authorization is in the delegation chain. Once the revocation request is validated, the revocation involves recording the certificate in the service's

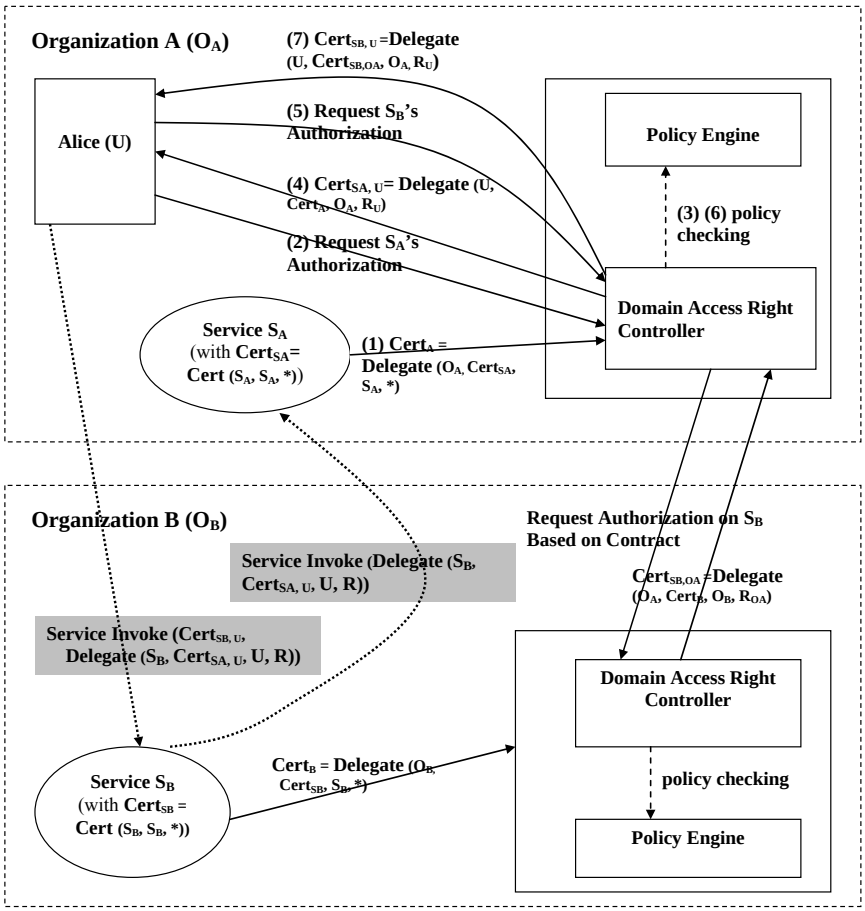

Figure 1. Certificate granting and service invocation in FAccM architecture.

revocation list, indexed either by its Globally Unique Identifier (GUID) or the cryptographic hash of the certificate. There is no need to propagate revocations to other services.

Delegation and revocation with ABAC follows the organizational structure. The DARC delegates to the department head; the department head delegates to the manager; the manager delegates to the user who will do the work; the user delegates to the process invoking the service. A manager reassigning work can simply revoke and delegate certificates as needed. The rest of the organization need not be involved.

4.2 Inter-organization

When crossing organizations, the pattern described above for granting access within a single-organization still holds. When a contract is signed granting organization O_A the right to use aspects of service S provided by organization O_B, the DARC in O_B delegates to the DARC in O_A the right to use those aspects of S covered by the contract.

$$\text{Cert}(O_A,O_B,R) = \text{Delegate}(O_A,\text{Cert}(O_B,S,*),O_B,R)$$

The DARC in A will then store this cross-organizational delegated certificate as if the certificate was issued by a service hosted in O_A, albeit with one extra layer of delegation in the certificate. The DARC in O_A issues subsets of the rights received from the DARC in O_B to subjects in O_A as if these were authorizations to local services.

O_B may have a policy on the classes of subjects it wishes to have access to S, e.g., users who have taken a training course. For subjects in O_B, this policy is controlled by the policy engine of O_B. That won't work for subjects in O_A because only the policy engine in O_A is consulted when deciding what rights to delegate. The contract between O_A and O_B must specify the policies O_A is expected to enforce.

No technology can force O_A to respect O_B's policy because O_B does not control how O_A identifies its subjects. Even using the policy engine in O_B doesn't help because subjects in O_A are free to share credentials. Users in O_B may also share credentials, but O_B can take the appropriate action when they are caught. O_B has no such control over users in O_A.

There is no difference between service invocation and revocation of local and remote services. U's authorization certificate to S, Cert(U,U',R_U), where U' is either an end user or the service S or the DARC, is used to invoke the service. Service parameters, such as to S_1, are delegated to S by U, Cert(S,U,R_S). Figure 1 shows how authorization certificates are passed along service

invocations, for a user U in organization O_A to invoke a service S_B in organization B, with S_B further invoking service S_A in organization O_A. By default, the delegation of parameters involves only U and S. In particular, the DARC of neither organization is involved. If there is a concern that such delegations might violate some policy of the user's organization, when U invokes S, the DARC in U's organization can establish a proxy service that will only forward requests that follow policy. Only the proxy service, but not the actual service, is visible to U in U's organization. Authorizations are revoked by sending a revocation request directly to S. Organizations that do not wish to rely on another organization for revocation have the option of proxying requests to those services. Having the proxy in the invocation path adds one more level of indirection, but it doesn't impact the manageability of rights.

5. Using Standards to Implement ABAC

We have shown how to implement Federated Access Management using Authorization-Based Access Control in the abstract. In this section, we show how to use SAML assertions [19] as authorization tokens. We show how to construct a SAML certificate delegation chain, how to specify constrained delegation, and how to dynamically construct authorization certificates to represent transitive rights that encode full responsibility tracking. More detail is available with the description of our reference implementation [19].

5.1 SAML Certificates as Authorizations

A SAML assertion, defined by the OASIS consortium [27], consists of three types of assertion statements. The authentication statement shows how and by whom a subject has been authenticated. The attribute statement states properties of a subject. The authorization decision statement for a particular resource states whether access is granted or denied.

The SAML standard includes the following fields in an authorization decision statement, which we use to grant rights:

(1) **Decision**: Whether access is denied or permitted. We always set this field to denote that access is permitted.

(2) **Resource**: What resource the authorization decision applies to. In our use of the certificate, the resource is a web service, and this field is used to encode the URL of the web service endpoint reference.

(3) **Subject**: Which subject the authorization applies to. The subject can be either a user or a web service. Each subject is represented by a X.509 public key certificate. The SOAP request to the web service must be signed by the private key corresponding to the public key in this field.

(4) **Action**: Which actions on the resource are being authorized. In our certificate, an action is a method provided by the web service specified as the Resource. The namespace attribute of the Action includes the web service's URL.

(5) **Evidence**: Information to support the claim that the authorization is valid. In our certificate, this field contains a copy of the certificate that represents the rights being granted to the delegator.

Using the evidence field this way lets us reconstruct the delegation chain, even across subjects from different organizations. The delegation chain starts with the innermost certificate, which was signed by the owner of the web service. Each subsequent delegation is embodied by the next outer SAML Assertion. Each Assertion is signed by the private key corresponding to the X.509 public key certificate listed in the subject field in the next inner certificate. We show this structure in Figure 2. Entities specified in boxes are XML elements and entities specified in angle brackets are XML attributes. Detailed SAML-based authorization certificates can be found in [19].

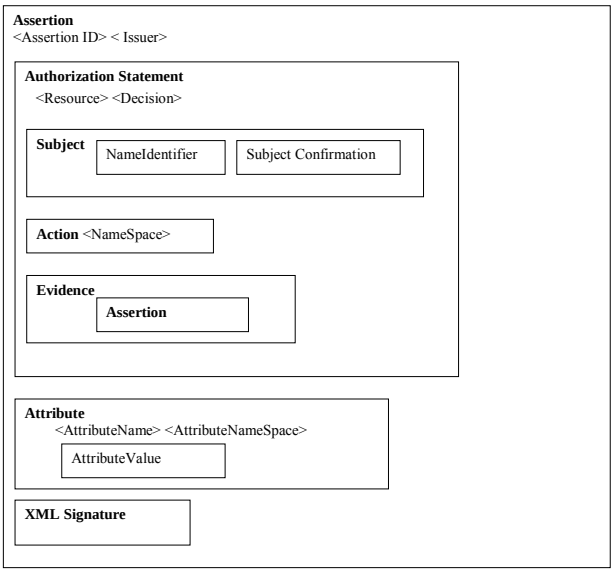

Figure 2. SAML certificate layout for authorization-based access control.

Although an auditor in O_A can show that some user U in O_B took some action or delegated to some other user, that auditor may not have the means to associate the public key of U with a particular responsible party. However, the auditor does know the responsible party in O_A. Hence, by following the delegation chain the responsible party can be identified.

5.2 Constrained Delegation

Often in the delegation chain, the delegator is only willing to delegate a subset of its right to the delegatee. In the web service environment, a simple constrained delegation is at the level of the web service method. The delegator allows only a subset of the web service methods to be delegated by simply listing Action fields containing only the methods being authorized.

A more sophisticated form of constrained delegation is to put limits on the range of parameter values being authorized. We use SAML attribute statements to express such constraints. The constraint is specified by name, encoded in the AttributeName field, and the associated value, encoded in the AttributeValue. Since the Attribute statement is distinct from and in parallel to the authorization statement, we need to establish the association between the constraint and the corresponding web service method. Making the namespace of the Attribute the concatenation of the web service namespace and method name serves this purpose.

5.3 SAML Authorization Certificate Validation

The authorization certificate encoded in the SAML assertion is validated by the web service for each invocation from the client. The validation is separated into a service-independent part and a service-dependent part. The service-independent checking includes:

(1) every certificate in the delegation chain has a valid XML signature;

(2) no certificate in the delegation chain has expired;

(3) no certificate in the delegation chain is in the service's revocation list;

(4) the issuer of every certificate in the delegation chain (embedded in the XML signature) must match the subject (explicitly expressed in the authorization statement) in the certificate encapsulated in the Evidence field;

(5) the web service method being invoked needs to be explicitly stated as "permit" in the all certificates in the delegation chain;

(6) each invocation needs to have a verifiable message digest, and the X.509 public key certificate associated with the message digest needs to match what is stated in the subject field at the outmost certificate.

Checking of Step 6 prevents a subject from using an authorization certificate issued to a different subject. Aside from Step 6, all the steps can be done by using XPath to traverse the certificate hierarchy in the XML document containing the SAML Assertion. Step 6 requires that the web service connect the signature on the SOAP request to the Subject specified in the SAML Assertion. This procedure is explained in Section 6.

Service-dependent checking makes sure that no delegation grants more rights than appear in the Evidence. Such constraint checking depends on the semantics of the individual services.

6. Message Interception on Service Invocation

SAML certificates are assumed to be public documents. When used as an authorization, the Assertion specifies the delegatee Subject and is signed by the issuer, which is the delegator Subject. Hence, merely verifying the integrity of the certificate does not tell if the certificate was presented by someone else. In this section we show how to use message interception to address this issue. Our solution uses the .NET web services framework. A reference implementation is available for download [20, 21].

6.1 .NET Messaging Layer Basics

In the .NET web service environment, service requests and responses are carried in a SOAP message, which consists of the header and the body. For a web service call, a SOAP body encodes the call parameters, while the header encodes out-of-band parameters such as security tokens [32]. .NET web services provide support for application-defined SOAP headers as extensions from a generic SoapHeader class [12]. A web service can define such a SOAP header instance via a custom attribute annotated to the web service method, and allow the header to be accessed in the method's implementation.

.NET web services also provide an application-defined message level interceptor. Each interceptor is defined as SOAP extension class, extended from the generic SoapExtension class [12]. The extension provides message level interception points along the call/return path – before/after serialization and before/after deserialization, on both the client and the service side. Figure 3 shows the two interception points we use, after serialization on the client side and before deserialization on the service side.

6.2 Subject Verification

We solve our validation problem by verifying that the SOAP message was signed by the private key corresponding to the X.509 public key certificate in the Subject field of the authorization assertion. The requester signs the SOAP message on the request side, and the server checks that signature on the receiving side. There is a slight complication. The SOAP signature is at the messaging layer; the SAML assertion is at the application layer. We need to pass the necessary information between these layers.

To this end, we define a *MessageSigningHeader* as a SOAP Header. On the client side, before service invocation, the service proxy is provided with a SOAP header that includes credentials needed to access the user's certificate store. The header also has three other fields defined: a public key (with type of byte array), an XML signature (with type of XmlElement), and a Boolean used to hold the message integrity signature checking result, which are initialized with the default values of null or false.

We also defined a *WebServcSoapExt* as a Soap Extension. On the client side, we intercept the service call after serialization. At this time, the whole call invocation request has been serialized as an XML SOAP message. First, we use the user credentials to retrieve the public/private key pair from the local certificate store. Next, we construct an XML signature over the Soap body. The MessageSigningHeader is also represented as an XML tree within the Soap Header section. We directly manipulate the XML tree of this MessageSigningHeader to set the public key and the computed XML signature. At the end, we remove the credentials from the XML tree.

On the server side, we intercept before deserialization. At this time, the whole call invocation request is still in the form of a SOAP message. We take the XML signature and public key out of the MessageSigningHeader, and verify them against the SOAP body. The verification result is then assigned to the boolean message integrity field in the MessageSigningHeader.

Finally the flow of control reaches the service method being called. In the method implementation, the MessageSigningHeader is retrieved, and the stored message integrity result is extracted. If this value is True, the public key is retrieved from the SOAP header. The service then pulls out the authorization token, traverses the SAML certificate, and extracts the outermost certificate's Subject field. The Subject field contains the X.509 public key certificate belonging to the delegatee. If the public key from the header and the public key extracted from the subject field are the same, we know the requester is the delegatee. (Actually, we only know that the requester knows the delegatee's private key, but that's the best we can do.)

```
public class MessageSigningHeader {
   public string CertStoreName;
   public string CertStorePassword;
   public byte[] PublicKey;
   public XmlElement XmlSignature;
   public bool CheckingResult;
}
```

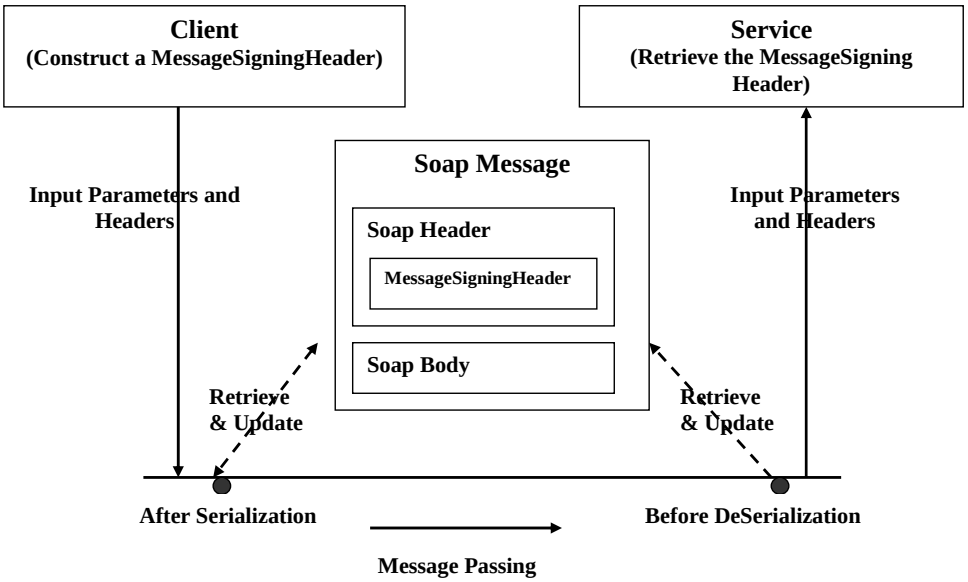

Figure 3. Using Soap extension and Soap header to facilitate subject verification.

In summary, the application-defined Soap Header provides the link between an application (both client invocation and service implementation) and the generic message level interception. The required information provided by message interception to the application is transported via the Soap Header. By combining these two mechanisms, we are able to enforce that the authorization token comes from the right client and that the authorization token has not been forged.

7. Prior Work

Access control became necessary when users started storing their data on computers they shared. Two forms were developed, capabilities [6] and access control lists, ACLs [5]. ACLs use the subject's identity to make access decisions and are the most common form of Identification-Based Access Control (IBAC). Capabilities, which combine designation with authorization, implement Authorization-Based Access Control (ABAC). ACL systems became dominant after claims were made that capabilities were unable to support some security properties of interest to the intelligence community [3, 23]. These criticisms have recently been addressed [25, 26], but our install base is almost entirely ACL based.

It is widely known that managing an ACL system can overwhelm a system administrator [36, Chapter 8], since a large number of changes may be needed when a user changes responsibilities. Role-Based Access Control (RBAC) [11], which assigns access rights to roles and subjects to roles, reduces this burden. In practice, the rights of a role often depend on which subject is in that role, which leads to role explosion. Policy-Based Access Control (PBAC) [2] was developed to address this problem. Each subject is assigned a set of attributes. A policy engine checks these attributes and the request against some policy, authorizing the request if it does not violate the policy. SAML certificates were designed to support these access control models with their Authentication, Attribute, and Authorization fields.

While RBAC and PBAC are effective at solving the rights management problem of IBAC that they were designed to solve, they do not address the other problems discussed in Section 2. In practice, by requiring administrator involvement in every delegation, they lead to environments where rights are granted in bulk, making it harder to enforce least privilege. Akenti [37] is an alternative that uses X.509 certificates to express rights. Although the authors briefly discuss an approach similar to what we present here, which they call the *push* model, they focus on a *pull* model. In that model, the user authentication accompanies the request, and the service (gatekeeper in their terminology) asks Akenti to retrieve the user's authorization certificates. This approach suffers from several of the problems discussed in Section 2. Delegation, which is key to FAccM, is only mentioned in passing, but it appears that the Akenti server must be involved, at least in the pull model.

Although the term is new [18], ABAC has a long history. The original capability paper [6] described specialized hardware for managing hardware resources. KeyKOS [13] later implemented capabilities for general resources types on commodity hardware. Capabilities were later extended over the network [7].

ABAC is more general than capabilities. Client Utility [17] used "split capabilities", which partially separated designation from authorization. The work presented here most closely follows the way e-speak [16] used SPKI [8] certificates. In that case and in our work, the authorization certificates are capabilities if they are used as the service parameters. They are not if they are used in addition to some other form of designation. We believe that it is better to use them as capabilities, but legacy systems may not support having the certificate as the argument. In these cases, a non-capability form of ABAC is better than the other options.

8. Conclusions

ABAC encourages delegation, allowing subjects to delegate to other subjects and processes they run only the rights needed to do the job. Since delegations can be further delegated, they follow the pairwise trust relations. When a delegation crosses an organizational boundary, the responsible party in that organization controls its use. This step is the key to Federated Access Management, allowing organizations to manage their own rights without needing to manage users in other organizations

We have shown how to use the web services standards, in particular SOAP and SAML, to implement ABAC. While this use fully conforms to the published standards, it is clear we are using these components in a way not envisioned by the standards committees. Further, none of the existing toolkits supporting web services development implement what we propose. Nevertheless, we feel the benefits are compelling enough to warrant a change.

To illustrate the FAccM mechanism detailed in this paper, we completed a reference implementation in the .NET web service environment. The implementation is available for public download [20, 21].

9. REFERENCES

[1] ActiveIdentity, Single Sign-On, http://www.actividentity.com/solutions/technology/esso__overview.php

[2] Blaze, M.; Feigenbaum, J.; Lacy, J., "Decentralized trust management," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 164-173, 1996.

[3] Boebert, W. E., On the Inability of an Unmodified Capability Machine to Enforce the *-property. In *Proc. 7th DoD/NBS Computer Security Conference*, pages 291–293, Gaithersburg, MD, USA, September 1984. National Bureau of Standards.

[4] Computer Associates, Single Sign-On, http://www.ca.com/us/products/product.aspx?id=166

[5] Daley, R. C. and Neumann, P. G., A general-purpose file system for secondary storage, *Proceedings of the Fall Joint Computer Conference*, 1965.

[6] Dennis, J. B. and Van Horn, E. C., Programming Semantics for Multiprogrammed Computations, *Comm. of the ACM*, 9, #3, 1966.

[7] Donnelley, J. E., A Distributed Capability Computing System. In *Proc. Third International Conference on Computer Communication*, pages 432–440, Toronto, Canada, 1976.

[8] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T., "SPKI Certificate Theory", IETF RFC 2693. http://www.ietf.org/rfc/rfc2693.txt

[9] Extensible Access Control Markup Language (XACML) V1.1, http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf

[10] Extensible Markup Language (XML), http://www.w3.org/XML/

[11] Ferraiolo, D. F. and Kuhn, D. R, "Role Based Access Control" *15th National Computer Security Conference*, 1992.

[12] Ferrara, A. and MacDonald, M., *Programming .NET Web Services*, O'Reilly Media, Inc., 2002.

[13] Hardy, N., KeyKOS Architecture. *SIGOPS Operating Systems Review*, 19(4):8–25, 1985.

[14] Hardy, N., "The Confused Deputy: (or why capabilities might have been invented)", *ACM SIGOPS Operating Systems Review*, Volume 22, Issue 4 (October 1988).

[15] Henning, M. and Vinoski, S., *Advanced CORBA Programming with C++*, Addison-Wesley, 1999.

[16] Hewlett-Packard, *e-speak Architectural Specification*, Release A.03.14.00, 2001.

[17] Karp, A. H., Gupta, R., Rozas, G., and Banerji, A., The Client Utility Architecture: The Precursor to E-Speak. Technical Report HPL-2001-136, Hewlett Packard Laboratories, 2001.

[18] Karp, A. H., "Authorization Based Access Control for the Services Oriented Architecture", *Proc. 4th Int. Conf. on Creating, Connecting and Collaborating through Computing (C5 2006)*, Berkeley, CA, IEEE Press, January (2006), http://www.hpl.hp.com/techreports/2006/HPL-2006-3.html. Some of the introductory material comes from this paper.

[19] Li, J. and Karp, A., "Zebra Copy: A Reference Implementation of Federated Access Management", HP Labs Technical Report HPL-2007-105, http://www.hpl.hp.com/techreports/2007/HPL-2007-105.html

[20] 20. Li, J. and Karp, A., "Zebra Copy sample code", http://www.hpl.hp.com/Alan_Karp/ZebraCopy.zip

[21] 21. Li, J. and Karp, A., "Zebra Copy sample code with SOAP interception", http://www.hpl.hp.com/Alan_Karp/ZebraCopyExtension.zip

[22] Liberty Alliance, http://www.projectliberty.org/.

[23] Mayfield, W. Traditional capability-based systems: An analysis of their ability to meet the trusted computer security evaluation criteria. Technical report, National Computer Security Center, Institute for Defense Analysis, 1987.

[24] Microsoft, "Introducing Windows CardSpace", http://msdn2.microsoft.com/en-us/library/aa480189.aspx

[25] Miller, M. S. and Shapiro, J. S. Paradigm Regained: Abstraction Mechanisms for Access Control. In *Proc. Eighth Asian Computing Science Conference*, pages 224–242, Tata Institute of Fundamental Research, Mumbai, India, 2003.

[26] Miller, M. S, Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control, Ph. D. Thesis, Johns Hopkins University, Baltimore, Maryland, USA, 2006.

[27] 27. OASIS, "Security Assertion Markup Language (SAML) 2.0 Technical Overview, Working Draft 05", 10 May 2005, http://www.oasis-open.org/committees/download.php/12549/sstc-saml-tech-overview-2%5B1%5D.0-draft-05.pdf

[28] 28. Papazoglou, M.P and Georgakopoulos, D., "Service-Oriented Computing," *Communications of the ACM*, Vol. 46, No. 10, pp. 25-8, Oct. 2003.

[29] Ping Identity, "Reducing Account Sharing with Federated Single Sign-On", Webinar, http://www.pingidentity.com/p/03yVcBqM?elq=F993B4D596D54D5B91838E8F7ECD6DE6

[30] Ping Identity, Single Sign-On, http://www.pingidentity.com/resources/88

[31] RSA Conference 2007, http://www.rsaconference.com/2007/US/.

[32] 32. Security Token, see http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, and http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf

[33] Simple Object Access Protocol (SOAP) 1.1, W3C Note, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[34] Stoll, C., *The Cuckoo's Egg*, Pocket Books, New York, 1989.

[35] Stojanovic, Z. and Dahanayake, A. (eds), *Service-Oriented Software System Engineering: Challenges and Practices*, Idea Group Publishing, 2005.

[36] The Open Group, CDSA Explained, http://www.opengroup.org/bookstore/catalog/g905.htm, 2001.

[37] Thompson, M. R., Essiari, A., and Mudumbai, S., Certificate-Based Authorization Policy in a PKI Environment, *ACM Trans. Information System Security*, Vol. 6, No. 4, Nov. 2003, pp. 566-588.

[38] Universal Description, Discovery, and Integration (UDDI), http://www.uddi.org/.

[39] Vinoski, S., "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol.35, no.2, pp. 46-55, Feb. 1997.

[40] Web Services Description Language (WSDL) 1.1, W3C Note, http://www.w3.org/TR/wsdl.html.

[41] XML-Signature Syntax and Processing, W3C Recommendation, http://www.w3.org/TR/xmldsig-core/