Authorization-Based Access Control for the Services Oriented Architecture

Alan H. Karp
HP Laboratories Palo Alto
HPL-2006-3
January 3, 2006*

Services Oriented Architecture, Web Services, security, access control

Several attempts at using the Services Oriented Architecture have failed to achieve their goals of scalability, security, and manageability. These systems, which base access decisions on the identity of the requester, have been found to be inflexible, don't scale well, and are difficult to use and to upgrade. This paper shows that identity-based access control is a key contributor to these failures and proposes another way to approach the problem. Basing access control decisions on authorizations presented explicitly by the requester leads to a more securable and more robust architecture.

* Internal Accession Date Only
Published in the Fourth International Conference on Creating, Connecting, and Collaborating through Computing (C^5), 26-27 January 2006, Berkeley, CA, USA Approved for External Publication

Authorization-Based Access Control for the Services Oriented Architecture

Alan H. Karp
Hewlett-Packard Laboratories
Alan.Karp@hp.com

Abstract

Several attempts at using the Services Oriented Architecture have failed to achieve their goals of scalability, security, and manageability. These systems, which base access decisions on the identity of the requester, have been found to be inflexible, don't scale well, and are difficult to use and to upgrade. This paper shows that identity-based access control is a key contributor to these failures and proposes another way to approach the problem. Basing access control decisions on authorizations presented explicitly by the requester leads to a more securable and more robust architecture.

1. Introduction

The Services Oriented Architecture (SOA) may yet deliver on the promise of loosely coupled application development that didn't materialize from earlier attempts, such as CORBA. The SOA is based on Web Services standards - SOAP for invocation, WSDL for interface definition, and UDDI for service discovery, all of which use XML as the communication format. These standards remove any dependence on machine architecture and operating system, making composition of independently developed components far easier.

One of the things holding back the widespread use of the SOA is the delay in reaching consensus on how to secure the services. 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, XACML, SAML, *etc*.

The relevant standard for a discussion of access control is the Security Assertion Markup Language (SAML) [1]. 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 are based on the user's identity. For example, the SAML Technical Overview [1] states,

> 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, 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 [1], 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.

Left unspecified is how the service provider uses 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 makes the access decision with 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 just have the request convey the authorization information instead of or in addition to the requester's identity? As shown in Section 6, passing authorizations has a number of important advantages over passing authentications.

The advantages of using authorizations are illustrated by the example scenario sketched in Section 2. Sections 3 and 4 outline how identification based access control (IBAC) can be applied to this scenario. An implementation based on authorization based access control (ABAC) is sketched in Sections 5 and 6. Section 7 outlines some implementation options.

2. Sample Scenario

While the SOA can be useful within a single organization, its real promise comes from the ability to federate services provided by different organizations. This federation exposes some of the problems that arise from basing access control decisions on identities.

In order to illustrate these problems, we use the simple scenario shown in Figure 1. Actual deployments will probably involve more complex arrangements, but it is unlikely that an architecture that makes it hard to deal with this simple case will be able to deal with more realistic configurations.

While this example may appear contrived, it was provided by a group at the US Navy to a team of contractors as a test of their web services middleware. These contractors concluded that they could not make the example work, specifying identity management as a key problem.

In this sample scenario, we have a user, Alice, invoking a service, which we'll call A6, for the sixth service of service provider A. That service uses service B5; B5 uses A3 and B2; B2 uses C3 and C5; C5 uses D5. Alice only invokes service A6, which invokes other services that may be unknown to Alice. Indeed, Alice may not have the right to invoke these other services herself. This kind of cross-domain service composition is the source of the power of the SOA.

For this discussion, we say that Domain 1 provides weather prediction services; services in Domain 2 access satellite images, and services in Domain 3 return topographic data. In order to be clear, we distinguish between Alice as the *originator* of the request and the intermediaries that are *senders* of requests.

We assume that each domain has a single administrator. Thus, it is relatively easy for the owner of

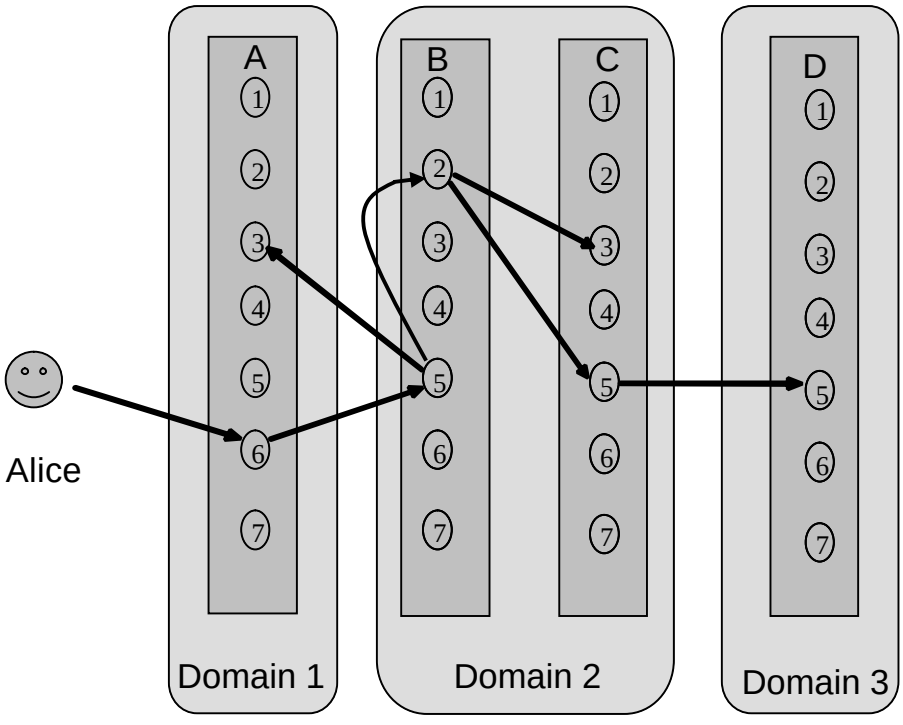

Figure 1. Sample scenario of cross domain service use. The arrows show service invocations.

Domain 2 to allow service B2 to use service C3. The interesting problems arise when a principal in one domain needs to use a service in another. Sections 3 and 5 contain descriptions of two approaches to solving this access control problem.

3. Identity-based Access Control

Let's follow the advice quoted above from the SAML specification and have the web services base their access decisions on user authentication. Clearly, service A6 bases its access decision on Alice's identity, but what about services A3 and B2? There are two options. Either they use Alice's identity or that of service B5. For the purposes of this discussion, we assume that sender authentication is used except for service D5, which requires originator authentication.

We need to populate the policy databases before accepting requests. Some cases are relatively easy. Each domain can start by entering rules for its principals. For example, the administrator for Domain 1 enters a rule stating that Alice may access service A6 if she used two-factor authentication to log in. The administrator of Domain 2 states that service B5 may use service B2 if the request comes from inside the firewall.

These rules may be stored in a variety of ways. If we use Access Control Lists (ACLs), each service has an entry for each user who may access it. That entry specifies what that user may do. In our example, the policy database for Domain 1 has an entry for service A6 listing Alice and stating that she may invoke the service if she used two-factor authentication.

Next, we need to add the rules for cross domain access. Domain 1 informs Domain 2 that service A6 needs to access service B5, and Domain 2 puts the appropriate entry into its policy database. For example, the rule might state that A6 can use B5 if A6 has an X.509 certificate and is using AES 256-bit encryption. Likewise, Domain 2 informs Domain 1 that service B5 needs to access service A3, and Domain 1 updates its policy database accordingly. Domains 2 and 3 do a similar exchange.

Alice needs to find the services she wants to use. In the SOA, she discovers services registered in some UDDI repository [2]. We ignore the fact that UDDI doesn't provide a means to restrict discovery and assume that all three domains register their services in a common repository. Such an approach is clearly unacceptable in a final architecture that requires tighter control and better privacy guarantees. Something can be layered on top of UDDI to meet these and other requirements, such as rich query, that UDDI doesn't provide.

At this point, Alice looks up services related to weather prediction in the UDDI repository and finds service A6. The tModel ([2], Section 1.6.4) in that entry points her to the WSDL specification and provides the SOAP invocation information. This specification notes that providing a topographic service as a parameter produces better results, so Alice searches the UDDI repository and finds service D5.

Alice then invokes service A6 with a reference to D5 as a parameter via a SOAP request that includes a SAML assertion provided by Domain 1. Service A6 checks the Domain 1 policy engine to determine if it should honor the request. If the document received from the policy engine asserts that access is allowed, service A6 starts processing. As part of its processing, service A6 invokes service B5, submitting its own SAML assertion. Alice's assertion must be carried along even though B5 won't use it because service D5 requires the originator's authentication.

4. Problems with IBAC

While using identification to make access decisions seems straightforward, there are a number of problems with doing access control this way. In this section we look at why the Navy consultants had so much trouble.

4.1. Trust relationships

In our example, Domain 2's trust relationship is with Domain 1, the signer of the SAML assertions, not Alice, the user of Domain 2's services. In particular, Domain 2 has no way to enforce restrictions on principals in Domain 1. If Domain 1 wants to state that strong authentication was used when it wasn't, there's nothing Domain 2 can do to detect this fact. Moreover, there's nothing to stop Domain 1 from creating fictitious identities. The only thing preventing such violations is the trust relationship between the two domains.

Even when the parties obey the trust relationship's requirements, there is the problem of propagating such information through intermediaries. Service D5 requires originator authentication, but its domain has no trust relationship with Domain 1. In this case, Domain 2 must attach its own SAML assertion to Domain 1's request and forward it to Domain 3. Domain 3 makes an entry in its policy database relying on the assertion from Domain 2. Domain 1's assertions are worthless to Domain 3.

Since Domain 3 has no trust relationship with Domain 1, it can do no better than merely copying the information provided by Domain 1. So, Domain 3's policy database has an entry stating that Alice in Domain 1 may use service D5 if the request has the appropriate SAML assertion signed by Domain 2. Domain 3 may choose to attach other conditions, such that Alice use two-factor authentication. At the end of this exchange, Domain 3's policy database has an entry for Alice, even though Domain 3 has no trust relationship with either Alice or her domain.

4.2. Changing personnel

Consider what must be done when Alice changes jobs within Domain 1, and Bob takes over her old duties. Domain 1 can't just revoke Alice's identity certificates; she still needs them to carry out her new duties. Instead, the policy databases must be updated. Not only must the Domain 1 administrator update all its ACLs, the administrator must inform the other domains of the change, and they must update their databases accordingly. Complicating the matter is the lack of a direct relationship between Domains 1 and 3. These problems are magnified if Domain 1 reorganizes, since many changes are likely to be needed. The overhead can be substantial if changes are frequent, so substantial that something "better than a straight ACL" ([3], page 57) is recommended.

4.3. Role explosion

Operating systems use group permissions to reduce the overhead incurred when people change jobs, and Role-based Access Control (RBAC) is the recommended approach for the SOA. These approaches do help reduce the overhead, but the responsibilities of groups and roles are not completely static.

There is often less than perfect alignment between job duties and roles, particularly when crossing domain boundaries. For example, Domain 1 may have a role "Weather predictor" which is known to need access to topographic data. Domain 3 may have a role that needs access to topographic data that it calls "Driving Directions" and a role that does not require topographic data that it calls "Weather predictor". These inconsistencies lead to an explosion in the number of roles needed to correctly map roles to authorizations.

4.4. Information leakage

Another problem with identity-based access control is the information leakage inherent in this approach. Domain 3 is given organizational information about Domain 1. For example, Domain 3 knows that Domain 1 has a user Alice and that Alice has certain properties. While some of these properties are relevant to Alice's use of services in Domain 3, others are not. In addition, Domain 3 must be informed when Alice's permissions change and told that Bob in Domain 1 is to be given those permissions.

4.5. Delegation and revocation

Even if Alice doesn't change jobs, there may be times when she needs to delegate some of her authorities. For example, if Alice is sick when a weather prediction is needed, she'd like to assign a subordinate to do the task. She must ask the Domain 1 administrator to update the policy database to allow Bob to invoke service A6 and inform Domain 3 to add an entry to service D5's entry for Bob. Bob won't be able to take over from Alice until the databases have been updated, and Bob will have that authority until the entries are removed. This requirement of consistency across databases severely limits the scalability of this approach.

This example illustrates another problem with IBAC, the misalignment of incentives. There are a couple of cases of interest. Say that Domain 1 is at risk if Bob continues to use service D5 after Alice returns to work, but Domain 3 is not. There is no incentive for Domain 3 to remove Bob's identity from its ACL promptly if at all. If Domain 3 is at risk, but Domain 1 is not, then Domain 1 has no incentive to inform Domain 3. This discussion of incentives hasn't even noted that all such requests are passed by Domain 2, which is likely to have no incentive at all to make sure the revocation happens.

4.6. System evolution

Another problem involves upgrades to the system. Since Domain 1 issues SAML certificates that Domain 3 must process, any upgrade must be done simultaneously. Backward compatibility isn't too hard; Domain 3 can start using SAML 3.0 formats while still accepting SAML 2.0 assertions. However, Domain 1 can't update to the newer format until Domain 3 is able to parse it. Since many dependences go in both directions, coordinated upgrade is required, a serious problem in an operational system. Also, parties wishing to join the system may have to completely redo their internal processing, a serious barrier to entry.

4.7. Ambient authorities

Yet another problem is that every program invokes services by presenting SAML assertions of a

principal's identity. We say that the granted authorities are *ambient* because they are taken out of the principal's environment. Ambient authorities make a confused deputy attack [4] possible.

Note that in our example service B2 invokes services C3 and C5. Let's say that C5 accesses the satellite images and C3 records the accesses for audit purposes. Alice would like to compromise the audit trail. Let's further assume that the API she uses has her specify the service to produce the output, C5, and the service to receive the output, B5. In normal operation, service B5 uses its permissions to invoke C5 and to modify the audit by invoking C3. However, if Alice specifies C3 to receive the output, service B5 will use its permissions to write image data over the audit trail. Granted, there are ways to prevent such attacks, but they complicate the system, and it's difficult to find all the places they can occur.

Ambient authorities lead to other problems. If the program has an error, or if the program has been subverted by a virus, it can carry out any action that the domain administrators have granted Alice in their policy databases. There is no way for Alice to start a program with the authority to do just weather prediction. Requests from that program to invoke other services will succeed as long as Alice has been granted permission.

This example illustrates a far larger problem, one closely related to that of viruses. One of the motivations behind SAML is Single Sign On (SSO), a means of avoiding the need to sign on multiple times with a variety of passwords. An unfortunate side effect of the way SAML is used for SSO is an amplification of the damage that can be done by malicious or erroneous code running in a login session. Without SSO, a virus running in a program can abuse any privileges of the logged in user, but only on that machine. With SSO, the resources at risk extend to any service in any Domain that has granted access to the user.

4.8. Distributed Identity management

Note that we haven't raised the difficult issue of distributed identity management. Alice most likely has a *distinguished name* (DN), which has components listing her domain, organization within that domain, *etc*. Such names unnecessarily expose organizational information. Worse, they require updates to all the policy databases whenever a DN changes, either because of a job change or an internal reorganization. There is a well-known vulnerability in Active Directory Servers [5] that is closely related to delays in updating such designations. A new domain entering the system may have to reassign the DN's of its members to avoid conflicts with those in the system it is joining.

4.9. Summary

Using identity-based access control results in a system that is hard to manage, hard to evolve, and is susceptible to erroneous programs and viruses. Some problems, such as confused deputy attacks, might be preventable by extraordinary care in design and coding. Others, such as the difficulty of managing the policy databases in a very large system, are inherent in the approach.

5. Authorization-Based Access Control

Let's step back and see how IBAC works. With identity-based access control Alice makes a request of service A6 and includes proof of her identity. The service submits this information to the policy engine, which determines the authorizations and reports back to the service. The service bases its access decision on this authorization, not Alice's identity.

That being the case, we can make a small change

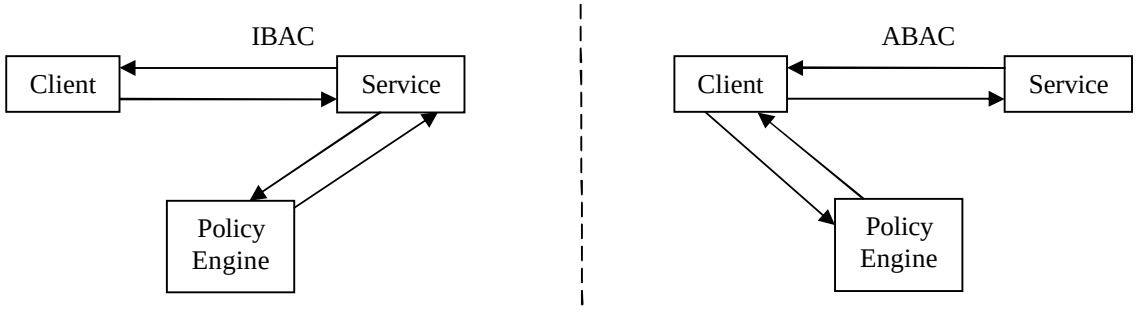

Figure 2. Comparison of identity-based (IBAC) and authorization-based (ABAC) access control.

to the procedure. We'll have Alice present her identity to the policy engine and get back the set of authorizations that represent what she is allowed to do. She then makes the request of service A6 and includes the rights she'd like to exercise. Service A6 need only verify that the authorization for this service hasn't been forged. We call this approach Authorization Based Access Control (ABAC).

All we've done is move two lines on the architecture diagram in Figure 2. Instead of Alice providing her identity to the service and the service receiving the authorization from the policy engine, Alice presents her identity to the policy engine, receives her authorizations, and presents the appropriate ones to the service when making a request. As we'll see below, this small change results in a far more manageable, evolvable, and secure system.

As with identity-based access control, we start by populating the policy database. Domain 2 makes service B5 available to users in Domain 1 by giving Domain 1 an *authorization* and a description of the policy to be enforced, such as two-factor authentication. The agreement between Domains 1 and 2 specifies that Domain 1 will only give this authorization to principals who satisfy the terms of the policy.

This approach sounds like Domain 2 is giving up too much control by having Domain 1 enforce the policy. As we saw earlier, this loss is illusory. Even with identity-based access control, Domain 1 is actually in control. If Domain 2's policy is for two-factor authentication, Domain 1 can always sign a SAML assertion to that effect whether it's true or not. Since there is no way for Domain 2 to detect such violations, nothing is gained by insisting that Domain 2 enforce the policy on Domain 1's principals.

Domains 1 makes service A3 available to Domain 2 in a like manner, while Domains 2 and 3 make a similar exchange. If Domain 3's policy for service D5 is that it can be passed on, then Domain 2 will make the authorization to use the service available to Domain 1 in the same way it did its own services.

We now have a situation where each domain has a database of all the services available to its principals with a set of authorizations and their corresponding policies. That means that there's no point in searching other domains' UDDI repositories. If the information exchanged includes UDDI registry information, each domain gets a level of control not provided by UDDI itself by simply restricting access to its repository to principals in its domain. If more control is desired, this model can be extended to organizations within each domain.

Once the services are registered in each domain's policy database, the policy engine sets up the authorization lists. While identity-based access control has a list of users associated with each service, authorization-based access control has a list of authorizations for each principal. The policies specifying which users get which rights can be expressed using a policy language such as KeyNote [6].

When Alice logs in, she is handed the base set of authorizations she needs to do her job, one of which may be the authorization to search Domain 1's UDDI repository. She presents this authorization along with her search request and finds the weather forecasting service A6. The tModel in that service entry specifies that she'll get better results if she has authorization to get topographic data. Alice then searches for topographic data services and finds service D5.

If Alice doesn't already have the authorizations to use A6 and D5, she next sends a request to her policy engine. If she meets the requirements, the policy engine returns the proper authorizations to her. Alice can now invoke the weather forecasting service A6, specifying the authorization to use D5 as a parameter.

6. Advantages of ABAC

Authorization-based access control doesn't have the problems inherent in identity-based access control. Since each domain only has information about its own principals, there are no problems of distributed identity management. When Alice changes jobs and Bob takes over her duties, Domain 1 simply changes the authorizations that Alice and Bob can get. If necessary, it can revoke those that are no longer appropriate. No other domain needs to be involved or even informed of the changes. Since no other domain is involved, there is no information leakage about organizational structure from one domain to another.

Easy delegation is another advantage of authorization-based access control. If Alice is sick when a weather prediction is necessary, she can send Bob the authorizations needed to do the work. Alice can also revoke that authorization when she returns to work. There's no need to inform anyone, not even Domain 1.

While delegating this way sounds like a security vulnerability, it isn't. Consider the threats. If Alice is trustworthy, she will only give her authorizations to someone she trusts not to abuse them. If Alice is not trustworthy, she can abuse her privileges. She can also take orders from Bob, making the requests he would make if he had the authorizations and sending the results of those requests to him. Alice can always be held responsible for any actions taken with authori-

zations granted to her. If an authorization is needed to communicate with another principal, then Domain 1 can control Alice's delegations by controlling the communications authorizations it grants to her.

Another advantage of ABAC is that system upgrades are far simpler. Users only authenticate to their domains, so changes to the authentication mechanism can be local. Also, the format of an authorization is needed only by the service that it references. In our example, the authorization to use D5 can be represented as a SAML 3.0 authorization even though Domain 1 is still using SAML 2.0. In fact, Domain 3 can use a completely different representation, one that doesn't use digital certificates at all. New organizations can join the system with little effect on their internal processes.

Note that each request need carry only the authorizations that Alice designates. If her program has an error or has been infected by a virus, the software can only abuse the authorities Alice provided, not all of her authorities as with identity-based access control. Thus, the virus problem, which would be exacerbated by SAML-style SSO, is mitigated.

Alice isn't the only beneficiary. Services don't need code to prevent confused deputy attacks. As before, we'll assume that service B2 invokes C5 to get satellite data and C3 to update audit information. There is no way for Alice to specify that C3 is to receive the output because she can only designate a service if she has an authorization to it. Service B5 will only use Alice's authorizations when producing output for her.

Authority-based access control leads to systems that are more scalable, because each domain is only responsible for its own principals, is more evolvable, because a service is the only one that needs to interpret the contents of the authorizations, is more private, because organizational information doesn't leak between domains, is more manageable, because of easy delegation, and is more secure, because fewer attacks are possible.

Audit trails and non-repudiation are important components of the system. Fortunately, they aren't lost when doing access control with authorizations. The signatures and encryption keys needed to protect the requests from tampering and prying provide sufficient information. If that proves inadequate, there is little problem requiring that authentication accompany the requests. Just don't use it to make access decisions.

7. Implementation options

It's important to know exactly what an authorization is before implementing an architecture based on ABAC. Fundamentally, an authorization is unforgeable proof that a particular request should be honored. In particular, the access decision does not depend on the identity of the requester.

Care is needed in designating the resource. Proper enforcement of the Principle of Least Authority requires that the designation be as specific as possible. Designating a file is better than designating a set of files. Even better is designating a particular operation on a file, such as read or write but not both.

It's also important that the designation not be subject to the *stale name problem*. Say that the authorization designates a file by name, and that file is deleted. If that name is reused some time later, then the old authorization may be applied incorrectly to the new resource.

The oldest form of computerized authorization is a capability [7]. So, if you're starting from scratch, you might want to use a capability secure language, such as ***E*** [8]. If your code base is in Java, you could use e-speak [9], which uses SPKI [10] certificates as capabilities. Web based applications might be able to use the web-calculus [11], which merges the REST model of computation [12] with capability security.

Sometimes, you're more constrained. You may be using legacy interfaces that require non-capability arguments, such as strings. You may also be forced to use specific standards. For example, the US Department of Defense is implementing its Global Information Grid (GIG) architecture [13] using SOA and has mandated a long list of standards, including SAML. Since changing mandated standards is extremely difficult, it's important to use them if at all possible. The current implementations of the GIG use only SAML identification and attribute fields. However, nothing in the specification says that you can't use the authorization fields of SAML [1]. You won't get the full benefits that come from other implementation options, but you won't have the problems associated with IBAC, either.

8. Conclusions

Large scale distributed systems are inherently different from stand-alone computers. There is little reason to think that designs for the latter are applicable to the former. Yet that's what identity-based access control does. It takes an access model from one realm and tries to make it work in the other. Authorization-based access control, which has advantages even on

stand-alone computers, is a better match to the requirements of distributed systems that span administrative domains.

Other distributed systems, particularly those that cross administrative boundaries can also benefit from switching from IBAC to ABAC. Administering a GRID [14] node involves creating and deleting accounts for users in many organizations. Adopting ABAC can avoid the problems associated with distributed identity management by allowing delegation of blocks of authorities.

One of the main complaints of Principal Investigators on PlanetLab [15] is the difficulty in delegating subsets of their authorities to their graduate students. Solutions to this problem that are being used for the 1,000 or so machines in PlanetLab today will not be practical as its size increases. ABAC, by decoupling the policy decisions into manageable chunks, avoids the scalability issues inherent in IBAC.

The SOA is quite different from the familiar systems that use identity-based access control. The SOA crosses administrative domains; it has far more users and separate components; it is far more dynamic in the rate and number of things that change; and no one party is in charge of managing updates. If the SOA is to achieve its goals, it is critical to reduce the coupling between domains to the greatest possible extent. Identification-based access control (IBAC) results in a tightly coupled system, one that requires distributed identity management, results in information leakage between domains, and makes delegation and upgrade difficult. Authorization-based access control (ABAC) avoids these problems while reducing the vulnerability of the system to viruses and confused deputy attacks.

Acknowledgements: Marc Wilson helped me work through the sample scenario with the Navy consultants and encouraged me to write the white paper that led to this paper. Discussions with Tyler Close, Bill Frantz, Norm Hardy, Mark Miller, Chip Morningstar, Marc Stiegler, Brian Warner, and Ka-Ping Yee helped clarify the ideas presented here. Ka-Ping Yee's detailed comments helped improve the presentation.

9. References

[1] 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

[2] OASIS, "UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, Dated 20041019", http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[3] Open Group, *CDSA Explained, An indispensable guide to Common Data Security Architecture*, The Open Group, (2001)

[4] Hardy, N., "The Confused Deputy", *Operating Systems Reviews*, **22**, #4, (1988). Also at http://www.cap-lore.com/CapTheory/ConfusedDeputy.html

[5] CERT, http://www.kb.cert.org/vuls/id/960267

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. "The Role of Trust Management in Distributed Systems Security." Chapter in *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, (Vitek and Jensen, eds.) Springer-Verlag, 1999. Also at http://www.crypto.com/papers/trustmgt.pdf

[7] J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations" *Comm. ACM*, 9(3):143-155, (1966)

[8] M. Miller, http://erights.org

[9] A. H. Karp, "E-speak E-xplained", CACM, vol. 46. #7, pp. 113-118, July (2003) , Also http://www.hpl.hp.com/personal/Alan_Karp/espeak/

[10] C. Ellison, http://www.ietf.org/rfc/rfc2692.txt (1999)

[11] T. Close, "web-calculus: Powerful Web Services through Composition", (2003) http://www.waterken.com/dev/Web/REST/

[12] R. Fielding; "Architectural Styles and the Design of Network-based Software Architectures"; Doctoral dissertation, University of California, Irvine; (2000).

[13] R. E. Levin, "The Global Information Grid and Challenges Facing Its Implementation", GAO-04-858, (2004). Also at http://www.gao.gov/new.items/d04858.pdf

[14] Grid, http://www.grid.org/home.htm

[15] PlanetLab, http://www.planet-lab.org/