3PAC: Enforcing Access Policies for Web Services

Jeroen van Bemmel[1] (jbemmel@lucent.com)
Maarten Wegdam[1,2] (wegdam@lucent.com)
Ko Lagerberg[1] (lagerberg@lucent.com)

[1]*Lucent Technologies, Bell Labs Advanced Technologies EMEA Twente*
Capitool 5, 7521 PL Enschede, The Netherlands

[2]*University of Twente, department of Computer Science*
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

Web Services fail to deliver on the promise of ubiquitous deployment and seamless interoperability due to the lack of a uniform, standards-based approach to all aspects of security. In particular, the enforcement of access policies in a Service Oriented Architecture is not addressed adequately. We present a novel approach to the distribution and enforcement of credentials-based access policies for Web Services (3PAC) which scales well and can be implemented in existing deployments.

Keywords: Web Services security, policy enforcement, distributed access control

1. Introduction

Web Services technology is commonly used to implement the Service Oriented Architecture (SOA) paradigm which promises simple, fast, and secure integration of systems and applications. SOAP, WSDL and UDDI are all widely supported XML-based open standards that help to realize this, but these technologies alone are not sufficient. In particular, security is for the most part considered "out of scope" in these standards, even though it is essential for deployment on public networks such as the Internet.

In this paper we address the issue of requestor authentication and access control for Web Services, in particular for the case where a provider provides discovery and access control for a set of services that are distributed across multiple administrative domains. An example of such a setting could be a telecom operator that collaborates with several content providers to offer bundled services to subscribers.

We attempt to answer the following research questions:

1. Which kind of access policies are required and/or can be implemented?
2. How can these policies be represented, distributed and enforced?
3. How can this be implemented in a secure, interoperable, efficient and scalable way?

The remainder of this paper is organized as follows. Section 2 provides background information on some aspects of Web Services security and access control. Section 3 describes our approach and a motivation for our choices. Section 4 documents our experiences with the implementation and validation of our ideas, and section 5 concludes this paper and identifies future work.

2. Background and concepts

Access control is defined as "the process of limiting access to the resources of a system only to authorized users, programs, processes, or other systems (in a network)" [1]. In the context of this paper we use this term to denote all the checks and procedures performed to determine if a received SOAP message should be admitted for further processing. In particular, this includes checking for compliance with the *access policy* which may require the use of particular security related mechanisms, for example digital signatures.

A common distinction made in literature on access control is based on the way the sender or 'subject' of a request is identified or classified. Schemes can be categorized as:

- Host-based access control
- Identity-based access control [3]

- Role-based access control [4], [5]
- Capability-based access control [6]
- Credentials-based access control [7]
- Hybrid forms

A detailed discussion on the merits and perceived advantages / disadvantages of the above schemes is out of scope of this paper. The scheme we use is best characterized as *credentials-based access control*, which (in our system) is a combination of all of the above: through authentication the identity of a requester is established and associated with one or more roles, typically in the form of membership of some group. The applicable policy is determined based on the combination <identity, roles, target service> and credentials are generated in the form of a signed token. This token is a receiver-restricted capability to access the target service, i.e. it should only be used by the requestor. More details follow in section 3.

2.1 Security threats

Web Services deployments are inherently vulnerable for abuse by unauthorized parties, especially in public networks such as the Internet. General security threats, e.g. denial-of-service attacks, are common for the technology and out of scope for this paper.

The threats that are pertinent to the 3PAC access control mechanism follow from the fact that potentially sensitive information is passed between the parties. The main threats are eavesdropping, message replay, message tampering, and man-in-the-middle attacks. The chosen implementation addresses these issues, as described in section 4.2.

2.2 Service discovery before access

A key characteristic of a Service Oriented Architecture is that the constituting services are *loosely coupled*. In practice, this means that dependencies between services are resolved during runtime using a suitable discovery mechanism. For Web Services the most common discovery mechanism in use is Universal Description, Discovery and Integration (UDDI). This OASIS standard is widely support by many vendors. Other alternatives exist (e.g. Liberty's ID-WSF Discovery service [8]) but we limit our discussion to UDDI (in fact our architecture does not depend on the particular discovery protocol used).

We make a distinction between two phases: the *service discovery* phase and the *service access* phase. A service consumer in a SOA first discovers a service and then accesses that service zero or more times. Both phases involve the exchange of SOAP messages, but the distinction is important since – typically – many more messages are exchanged during the access phase. This observation is key to the efficiency savings we realize: by performing the bulk of access policy evaluations during discovery, processing for each service access is limited.

2.3 Nature / types of policy rules

An important consideration is the flexibility of access control policies, in terms of "what can be expressed". We believe this design issue is best addressed in light of practical considerations: "What is really required?"

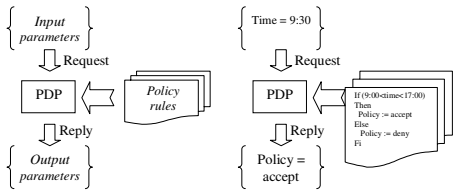

Figure 1 Policy rules with concrete example

Figure 1 illustrates the policy decision process: a request containing a set of *input parameters* is submitted to the policy decision point (PDP), which evaluates its set of policy rules and determines the outcome. A rule could for example be that "requests are only admitted between 9:00 and 17:00" (see figure).

We can classify policy rules in terms of the nature of parameters they use. We distinguish the following categories of parameters:

- properties of the *requestor*
- properties of the *target service*
- properties of an *individual request*
- *statistics* (functions over past values of properties, e.g. average load)
- *environment* settings or properties (including current time / date)

The set of all these parameters constitutes the context relevant for security. Using these parameters we can construct policy rules that are evaluated during service discovery and/or service access. For this paper we are not concerned with the syntax of policy rules, as many examples already exist (e.g. an XML-based syntax such as X-RBAC[2]). As stated before, for efficiency reasons we prefer to evaluate rules during discovery such that service access checks are relatively simple and fast. This can only be done for rules that do not depend on inputs only available during service access (for example: restrictions on particular request parameter values or rules based on up-to-date values of statistics). Such rules must be evaluated for every request and are therefore expensive in terms of required processing and the resulting increase of delay.

Rules based on statistics require state information to be maintained. Enforcement of such rules may reduce scalability depending on where in the architecture this information is stored, how accurate and fresh it needs to be, and whether it applies globally or only locally (e.g. per service instance) . Note that the mechanism used to collect and synchronise this state is in principle independent of the policy, although one may want to specify policies on the maximum stale time of information (i.e. policy-based caching).

3. The 3PAC architecture

We propose a novel architecture called 3PAC (3rd party access control) based on delegation of the computation intensive parts of access control (authentication and initial authorization) to a trusted third party. This third party offers service discovery functionality and performs authentication and identity- or role-based access control. It then generates signed credentials for use during service access. Our approach bears some similarities to Kerberos [12], some differences are discussed in section 3.5.

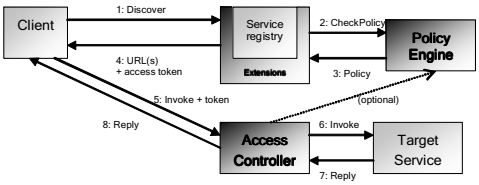

Figure 2 Access control using a token obtained during discovery

Figure 2 illustrates the 3PAC logical architecture and gives a high level overview of the message flow. The light components represent a standard Web Service deployment as it is used today. The dark shaded components are the proposed enhancements. The system operates in two phases:

Phase 1 - Service Discovery

During service discovery the client obtains the location of the service instance to be invoked. In a Web Service context, the WS client typically obtains a URL (or set of URLs) from a UDDI registry through a lookup by service name or identification[1]. The proposed extensions perform the following functions (in order):

- Authentication of the client performing the discovery
- Lookup and execution of a *discovery policy*
- Generation of a non-forgeable access token containing the *access policy*

The resulting service discovery phase corresponds to steps 1 to 4 in Figure 2.

The access token that the client receives in step 4 is used during each service access. The functions related to policies are delegated to a separate *policy engine* component (i.e., the Policy Decision Point), which may or may not be co-located with the registry (e.g. for performance reasons co-located may be better, but in multi-domain scenarios the policy engine might belong to a different administrative domain).

Phase 2 - Service Access

After service discovery the WS client can do zero or more service requests, which we call the *service access* phase. For each request the

[1] More elaborate search criteria are possible.

Access Controller(AC) component verifies the access token sent along (see Figure 2 step 5) and applies the access policy contained therein. As a result, service access may be denied or restricted. The service access phase corresponds to steps 5 to 8 in Figure 2. Depending on the access policy embedded in the token, the AC can consult a policy engine.

Clients are forced to follow this two-phased approach in order to obtain an access token; requests without token return an error response or are simply discarded. This implies that in our architecture clients *must* do discovery for each 3PAC-protected service they use. Given that dynamic discovery is already a common element of a Service Oriented Architecture (to achieve loose coupling and flexibility), we do not consider this a drawback.

When the discovery service and the target service belong to the same administrative domain they can work together to define and enforce a common access policy. In a multi-domain setting (i.e. the target service is operated by another third party and belongs to a different administrative domain than the discovery service) things become more complicated, since the domains may have different (or even conflicting) access policies. We leave discussion on this issue out of scope for this paper.

3.1 Revocation of granted access tokens

Revocation is notoriously hard to implement in a scalable way. Once a token has been signed and sent to the requestor, the client can use it until it expires. In systems that require revocation functionality (see e.g. [9]) several solutions are proposed, but ultimately the only way to solve it is to make a 'revocation check' against a list of revocations, for every request.

In our 3PAC architecture revocation can be supported on an 'as-needed' basis. That is, the requirement to check for revocation can be included as part of the access token. The discovery server would keep a list of revocable tokens issued, and include a URL to this revocation-check service in the token. The access controller then performs a query – but only for requests that are marked to need it. In addition, the discovery server might indicate a period for which the check is valid, such that the AC can cache the result. Of course the limitation of this approach is that once a token has been issued without revocation-check constraint it cannot be revoked. We argue that this revocation-as-needed is sufficient for many systems in practice.

3.2 Trust in 3PAC

A *trust model* is a 3-tuple (I,A,R) where *I* denotes the set of initial conditions, *A* represents the assumptions (unverifiable conditions) and *R* is the set of rules that describe "who trusts whom for what if why". In 3PAC the initial condition is that each access controller in the system has a set of public keys corresponding to discovery providers from whom it will accept tokens containing an access control policy. The assumptions made are common to most systems (e.g. signatures cannot be forged, private keys are not compromised). The set of rules depends on the deployment setting chosen, and may even be different per target service (i.e. it can be specified as part of the policy). In a typical scenario the AC trusts the discovery component to assert the correct subject and to provide the right access policy, if the signature validates. The discovery component in turn trusts the UsernameToken to contain the proper username if the password is found to be correct.

3.3 Supported policies and information

The access token accomplishes two things: it communicates the appropriate access policy, and (optionally) additional information in the form of assertions. A typical example is the identity(name) of the third party accessing the service, but in general any (static) properties can be communicated.

3.4 Discussion

We discuss different aspects of our 3PAC architecture that differentiate our solution:

Authentication and coarse-grained access control during discovery

Authentication normally takes place after discovery of a web interface, and before service access [10]. Our architecture combines

discovery with authentication, thereby allowing coarse-level access control based on requester identity and on which service is being discovered. In addition, we enable policies in which we would return a URL to a different service instance based on the requester identity, e.g. for QoS differentiation.

Fine-grained access control during service access

Our 3PAC architecture allows different fine-grained access policies which are enforced during service access. This includes policies that restrict for a specific requestor: which (if any) additional authentication is required, which methods can be called, which values can be used for parameters, what encryption is required and for what period access to the service is granted.

Tokens with embedded access policies

If access to a certain service is granted, the requestor receives a token with an embedded access policy to be applied for each access. By embedding the access policy in the token, we allow the Access Controllers - which enforce these access policies - to be implemented in a completely decentralized manner in which they do not need to interact with other components in the architecture. This results in a scalable architecture. Some policies may require global state, e.g., a policy to restrict the number of invocation that a requestor can make. These policies do require some form of interaction between the different instances of the Access Controllers, thus reducing scalability somewhat. However, this aspect can be controlled by choosing the appropriate policy per service.

The policy can be embedded in the token either by reference or by value. In the former case an additional indirection is added, the token could for example refer to one out of a set of policies preconfigured at the access controller, or it could contain a URL for the actual policy. Benefits are that the size of the token (and hence each service request) is reduced, and revocation could then be done more easily. On the other hand, preconfigured policies would introduce issues with referential integrity and would not be requestor specific, and URL references would require additional remote interaction (which could partly be alleviated through caching). In fact, dereferencing such a URL would require execution of some of the same policy logic as done during discovery, annihilating the benefit. A hybrid approach would however be possible (i.e. refer to a general, preconfigured policy at the access controller and add requestor specific constraints).

Multi-domain deployment and delegation

A web service can be deployed in a different domain than the service registry. In this case the web service may delegate authentication and (part of) the access control enforcement to the discovery domain. The AC can be deployed in either the discovery or the target service domain, and can be realized either as a separate component or co-located with each service.

Separation of concerns between application logic and access policy enforcement

The access policy enforcement is fully separated from the application logic, both on the requestor and on the web service side. The requestor side has to pass some form of authentication credentials at discovery, and the token during service usage. Depending somewhat on the deployment environment, this can be done by a system administrator without code change. The AC takes care of the enforcement of the policies on the web service side, and can also be implemented without requiring code changes to the web service. See also the next section for more information on implementation and deployment options.

Communication of requestor properties

Some service logic may depend on specific properties of the subject, for example the type of terminal a user is using to access the third party service. Rather than performing a lookup in some database for each request, such information may be attached to the request and is automatically protected from modification.

3.5 Comparison with Kerberos

Kerberos [12] is an authentication service used in an open network computing environment, to proof the identity of clients across an insecure network. Extensions for authorization and accounting purposes are described in [13]. By itself Kerberos cannot be used in a Web Services context, but several initiatives are being undertaken to enable the use of Kerberos security mechanisms. For example, WS-

Security defines a profile [14] to embed a Kerberos ticket in a SOAP header.

In terms of [13] the 3PAC access token can be seen as a restricted proxy (a service access capability). Kerberos requires the use of an additional authenticator per request to prevent replay attacks (through eavesdropping). In our solution we use existing, already widely deployed 'native' web technology (HTTPS) for this purpose. Besides this technological difference the conceptual differences are that this can be done selectively (per requestor / target service based on some policy) and that it is session based (as opposed to message based), which enables performance gains for series of requests. Furthermore, Kerberos does not address integration with a discovery service. New features we introduce are for example the possibility to add a restriction that tells the end-server to consult a given authorization server for each request, and the ability to include non-authorization related requestor properties.

4. Prototyping

We have built a prototype to validate the 3PAC approach and to do some initial performance measurements. Where possible, standardized SOAP extensions were used to implement the 3PAC functionality. In our implementation we assume there is a trust relation between the discovery component (UDDI registry/policy engine) and the AC, and message integrity is achieved using digital signatures.

4.1 Deployment environment

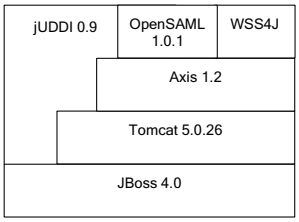

Figure 3 Prototype deployment platform

The basic deployment environment consists of a J2EE application server, web server and SOAP engine. We use common open source components for this Java™ environment, most notably JBoss, Apache Tomcat, and Apache Axis. On top of this base an open source UDDI implementation called jUDDI is used to implement standard UDDI v2.0 functionality. OpenSAML 1.0.1 is used to wrap the access token into a SAML [11] assertion. Finally, a pre-release of Apache WSS4J is used to provide WSS functionality. The platform schematically looks like Figure 3.

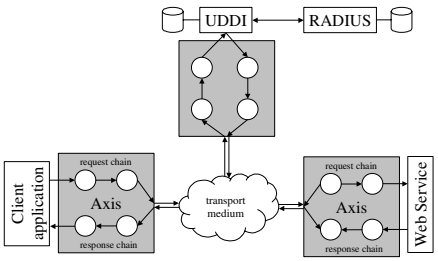

Figure 4 Prototype components

The complete prototype consists of four components, as illustrated in Figure 4:

- a 3rd party client application to discover and invoke a Web Service
- a UDDI registry that acts as a PDP and constructs access tokens
- a RADIUS server to authenticate clients
- a target Web Service with an associated PEP (the Access Controller which in our prototype is co-located with the Web Service)

The 3PAC functions are implemented using Axis request/response handlers that process the SOAP message headers. The additional functionality is completely transparent for the client, the UDDI registry and the Web Service, which allows us to use standard components and services without modifications (as long as they are based on Axis).

4.2 Protocol

The protocol extensions for service discovery and service access are based on the Web Services Security standard (WSS) [15]. During service discovery a WSS UsernameToken [16] is added to a header in the UDDI request to convey the client authentication credentials. The access token that is generated by the policy engine is wrapped in a SAML assertion and added to the UDDI response as a signed WSS SAML Token header [17]. The SAML assertion is cached by the client and included

as a WSS SAML Token in subsequent invocations of the target web service URL. Upon reception of a SOAP request, the access controller interprets the SAML Token, verifies the digital signature, checks that the public key corresponds to a trusted entity, and then enforces the embedded policy.

The use of WSS and SAML Tokens resolves most of the security issues as mentioned earlier in section 2.1. For security considerations and countermeasures to the security threats, see [17].

4.3 Access Token

The XML-encoded access token contains all conditions that apply for invoking the target web service by the requesting client. The token is implemented as an extension of the SAML "Condition" element in a SAML assertion, for example:

```
<Assertion AssertionID="..." ...>
  <Conditions NotOnOrAfter="2005-01-13T09:03:18Z">
    <PolicyCondition>
      <policy xmlns="http://www.lucent.com/3PAC">
        <appliesTo serviceId="MyService">
          <URL>http://somehost.com/MyService</URL>
        </appliesTo>
        <partyId>3pspl</partyId>
        <allowedMethods>methodA,methodB
        </allowedMethods>
      </policy>
    </PolicyCondition>
  </Conditions>
...
</Assertion>
```

4.4 Performance

The creation and enforcement of access control policies obviously impacts the performance of the Web Service infrastructure. We conducted a number of measurements to estimate the performance degradation caused by the mechanism.

The measured scenario is a service discovery followed by a Web Service method invocation. To simplify the measurements, the client application was authenticated locally by the UDDI registry instead of using an (external) RADIUS server.

The following measurements were performed:

1. Access control
2. 3rd party authentication
3. Creation of the access token wrapped in a SAML assertion
4. Creation of a digital signature
5. Verification of the digital signature
6. Enforcement of the access policy

Step 1 is a baseline measurement for a service discovery and method invocation without access control, as it is commonly done today. Steps 2, 3 and 4 are part of the discovery phase. Steps 5 and 6 are part of the service access phase.

To determine what parts of the mechanism have the biggest influence on the performance, the test run was executed multiple times. Each time an additional function was switched on, increasing the complexity and the overall processing time. All components run on a single desktop PC. The measurement results are listed in **Table 1**.

Step	duration (ms)	total (ms)
1	n/a	220
2	77	297
3	48	345
4	305	650
5	59	709
6	6	715

Table 1 Performance measurement results

The listed times are client-perceived duration of the scenario averaged over 100 separate measurements on a Dell Precision 420 dual Pentium® III 1GHz machine with 512MB of memory, running Windows™ 2000 SP4 with no significant background load.

As can be seen from the table, the average processing time of the measured scenario increases from 220 ms without access control to 715 ms with all parts of the mechanism enabled, i.e., an extra delay of about 500ms for a combination of a discovery and one method invocation.

Note that the mentioned times do not include any network delays, and no attempts were made to optimize any part of the code. Furthermore the prototype currently only supports the enforcement of relatively simple policies. The processing time may increase when more complex policies are supported.

Although the exact figures are not very important (they will be different in a real deployment), the results show that most of the 3PAC overhead can be attributed to service discovery (430 ms). The creation of the digital signature for the token is by far the most expensive operation. In a typical usage scenario, where the token is created once and

used multiple times, we consider this acceptable.

5. Conclusion

In this paper we have presented a novel approach to access control for Web Services, which is one of the major issues that prohibits deployment today. By resolving the static, request-independent parts of the access policy during service discovery and returning proof of conformance to the client using a signed access token, policy enforcement for each service request can be done locally and hence efficiently. As a result the per-request processing overhead is relatively small. The architecture does not prohibit the use of more flexible request-time policies such as limiting the maximum number of requests or revocation of the access token, the cost of additional processing and/or communication required for such policies is only incurred for requests that require this.

For future work we consider standardization and extension of the model to include the user of the third party service, such that properties of this user can be used for identity federation and privacy enforcement.

Acknowledgements

This work is part of the Freeband AWARENESS project (http://awareness.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025. We would like to thank Bharat Kumar from Bell Labs research for his contributions and suggestions.

6. References

[1] R. Agrawal et al, "Vinci: A service-oriented architecture for rapid development of web applications", in WWW10 Hongkong, May 01

[2] R. Bhatti, E. Bertino and A. Ghafoor, "A Trust-based Context-Aware Access Control Model for Web-Services", Proc. ICWS04 pp. 184-192

[3] L. Gong, "A secure identity-based capability system", Proceedings of the IEEE Symposium on Security and Privacy, pages 56--63, 1989

[4] J.S. Park and R.S. Sandhu, "Role-based access control on the web", ACM Transactions on Information System Security (volume 4 #1), 2001 pp 37-71

[5] R. S. Sandhu, et al. "Role-Based Access Control Models", IEEE Computer 29(2): 38-47, IEEE Press, 1996

[6] H. M. Levy. "Capability-Based Computer System",. Digital Press, 1984

[7] J. Biskup and S. Wortmann, "Towards a credential-based implementation of compound access control policies", Proceedings of SACMAT '04 pp 31-40

[8] Liberty ID-WSF Discovery protocol v1.1, http://www.projectliberty.org/specs/liberty-idwsf-disco-svc-v1.1.pdf

[9] L. Zhang, G. Ahn and B. Chu, "A rule-based framework for role-based delegation and revocation", ACM Transactions on Inf. Syst. Security, vol 6 no 3 2003 pp. 404-441

[10] M. N. Huhns and M. P. Singh, "Service Oriented Computing: Key Concepts and Principles", *IEEE Internet Computing*, vol. 9, no. 1, 2005, pp.75-81.

[11] Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1, OASIS Standard, 2003.

[12] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, 32(9):33-38. September 1994

[13] B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems, Proc. of the 13th International Conference on DC Systems, pp. 283-291, May 1993

[14] Web Services security Kerberos token profile, http://www.oasis-open.org/committees/download.php/1049/WSS-Kerberos-03.pdf (visited 20/12/2004)

[15] Web Services Security: SOAP Message Security 1.0, OASIS Standard, 2004.

[16] Web Services Security: UsernameToken Profile 1.0, OASIS Standard, 2004.

[17] Web Services Security: SAML Token Profile 1.0, OASIS Standard, 2004.