Delegation-based Security Model for Web Services

Wei She, Bhavani Thuraisingham, and I-Ling Yen
{wxs061000, bhavani.thuraisingham, ilyen}@utdallas.edu
University of Texas at Dallas, Richardson, TX, USA

Abstract. Web service is the emerging standard that supports the seamless interoperation between different applications. While the interoperability, flexibility and automated composition are continuously enhanced, security is still the major hurdle. In recent years, lots of studies have been conducted in web service security and various security standards have been proposed. But most of these studies and standards focus on the access control policies for individual web services and do not consider the access issues in composed services. Consider a simplest service chain wherein a user x accesses service s_1, and s_1, in turn, accesses service s_2. The current web service security framework assumes s_1 accesses s_2 based on its own privilege; thus sensitive information may be incorrectly revealed to x. A better solution is that x delegates its privilege to service s_1 for this access. However, problems such as how much privilege to delegate, how to confirm cross-domain delegation, how to delegate additional privilege when needed, etc. arise. The problem becomes more complex when workflow involves many layers of services. In this paper, we propose a delegation-based security model to address all these issues. It extends the basic security models and supports flexible delegation and evaluation-based access control.

1. INTRODUCTION

Service oriented architecture (SOA) is a very popular paradigm for system integration and interoperation. Web service is the current standard for SOA. While it has many benefits, security is still the major concern. A lot of efforts by various standards groups such as W3C, OASIS, WS-I, etc. have been devoted to web service security standards. WS Security ([OAS06]) specifies an abstract web service security model including security tokens with digital signatures to protect and authenticate SOAP messages. SAML provides an XML-based standard for the exchange of authentication, entitlement, and attribute information. XACML is the core XML schema defined to represent access control policies. WS Trust ([OAS07]) defines extensions to WS Security, including the methods to issue, renew and validate security tokens, and the way to exchange and broker trust relationships. WS Federation further defines how trust relationships are managed and brokered in a heterogeneous federated environment. WS Trust also supports extended features such as simple delegation and forwarding of security tokens between different parties and exchange of policies. WS Policy specifies a framework for expressing web service constraints and requirements as policies using policy assertions. WS Security Policy extends WS Security by specifying the policy assertions to describe security policies. WS Privacy embeds into WS Policy with a language to describe the privacy preferences and organizational privacy statements. WS Secure Conversation is defined to allow establishing and amending the security context and computing and passing the session keys.

Besides the standards specifications, a lot of studies have also been conducted to enhance web service security. The focuses of these studies can be classified into: (1) message-level security and/or transport layer security mechanisms, such as XML encryption, XML digital signature, and sender/receiver authentication ([WOO98], [BHA07]); (2) access control mechanism for individual web services ([BHA05], [BER06]).

Existing works focus on individual web services and do not consider the services composition. Consider a simple case wherein a user x invokes service s_1, and in turn, s_1 invokes service s_2. Assume that s_2 has sensitive information that can be accessed by s_1 but not x. In current web service framework, it is assumed that s_1 accesses s_2 using its own privilege. In this case, the sensitive information on s_2 may be incorrectly revealed to x. It is not reasonable either to have s_1 filtering out s_2's response before sending it back to x since s_1 does not have the knowledge of s_2's policies. A better solution is that x delegates its privilege to s_1 for the access.

Delegation is the activity that one party hands over its authority and responsibility to another in order to accomplish certain tasks. There are many issues to be considered in the delegation model in web service environment. First, it may not be easy to determine in advance how much privilege should be delegated. Delegating unnecessary privileges may result in privilege misuse. Also, it is difficult for x to know in advance what privileges are required since the services may be dynamically invoked. Thus, besides determining which privileges to delegate, it is necessary to support delegation negotiation among the involved entities in a transaction. Also, the infrastructure should support delegation as well as confirmation of the delegated privileges.

Delegation has been widely studied since 1990s in the context of distributed systems, multi-agent systems, access control, trust management, grid services, workflow management systems, etc. One delegation context is for proxy systems. When there are proxies between the requester and the resources, the requester has to authorize the proxies to access the resources on its behalf. Several certificate-based or token-based solutions have been proposed ([GAS90], [VAR91], [KOR00], [WEL04], [YIA96]). In these approaches, the user is entitled to access the resources and delegation is to the proxies. They do not

1530-2059/07 $25.00 © 2007 IEEE
DOI 10.1109/HASE.2007.76

consider the situation when the user may have to "borrow" the privileges for indirect service accesses. Also, many delegation works only consider user based delegation, i.e., the proxy acts as the user and has the full privileges of the user. Thus only the authenticity of delegation is proven. Constrained delegation (one entity delete its partial privileges to another entity) has been studied in role-based access control. But, these models ([BAR10], [ZHA03], [ZHN03], [GAN04], [YE04]) adopt a different approach which separates the roles rather than using certificates. In [LI03], [LI99], a logic-based language, Delegation Logic (DL) is proposed to formally express the delegation of authority, negation of authority, and the conflicts between authorities. Although DL is capable of expressing the constrained delegation of authorities, it does not cover some delegation aspects required in web services such as the multiplicity of delegation, different agreement types of delegation, etc. Moreover, the issues of confirmation and negotiation of delegation are not considered.

This paper proposes a delegation-based approach to secure web services especially for static and/or dynamic service composition. Our delegation-based security model provides the framework and defines the key processes to secure the web services through issuing, sending, confirming, negotiating, and revoking the delegation tokens. By adding a dedicated service, security token base, we provide a framework to determine the required security tokens. Delegation token and delegation policy are designed to confine the delegations. Key processes for the delegation confirmation are defined; and performing confirmation based on services' mutual trust further reduces its communication overhead. In our approach, services can delegate its own security tokens when the invocation fails due to insufficient privilege such that more flexible delegation is supported. Reliability of security token service and the revocation schema are also supported.

The organization of the rest of the paper is as follow. A short introduction to XACML and XACML profiles are given in Section 2. An overview of delegation-based security model is given in Section 3. Section 4 presents the underlying delegation protocol. Key components and processes are discussed in this section; also, the major issues listed above are addressed. The hierarchical evaluation-based access control mechanism is provided in Section 5 followed by an illustrative example in Section 6. This paper is concluded in Section 7.

2. XACML BACKGROUNDS

XACML (eXtensible Access Control Markup Language) is a declarative access control policy language consisting of a data flow model and a set of policy constructs In XACML, a rule is the most basic unit that can be evaluated; one or more rules together with a rule-combining algorithm form a policy, and in turn, several policies with a policy-combining algorithm construct a policy set. A rule defines a target, an effect, and a condition. A target defines a set of subjects, resources, actions and environments to which the rule is intended to apply; and a condition will further refine this applicability. An effect is defined in a rule to specify whether it is a positive or negative authorization. A subject is usually a human being, or some application which performs the access, whereas a resource is a service or datum which is accessible to a set of subjects. An action essentially represents the mode that a subject can access upon a resource. Such a set of access modes may vary a lot in different applications. And finally, an environment refers to a system related condition under which the policy may apply. Policy writers can specify attributes for any involved subjects, resources, actions or environments such that attribute-based access control policies can be represented using XACML. XACML also allows for defining multiple subjects and resources in a target element which makes it more flexible. Providing attributes in resource elements makes XACML capable of describing content-based access control. XACML provides obligation element to support describing obligation policies, and defines XACML context to provide a canonical form for the requests and responses with various formats. With XACML context, a policy management system can be easily implemented in a decentralized manner. Other features that XACML supports such as attribute operations and policy indexing are also worth noting.

The features discussed above are supported by XACML basic constructs (core specification). The core specification is also extensible. For example, RBAC, hierarchical RBAC, etc. can be represented using XACML without extension. Consider role-based access control policies. A role can be represented using an attribute of a subject. In RBAC profile, a role policy set is defined as a policy set that associates the users (subjects) with a given role attribute value and a permission policy set associated with that role. And such a permission policy set contains policy elements and rule elements that describe the resources and actions that are permitted to access by this subject. Similarly, other advanced concepts such as role assignments and role hierarchies can also be interpreted in such a manner.

3. DELEGATION-BASED SECURITY MODEL

Figure 1 depicts the architecture of our delegation-based security model and its key elements.

We define an administrative domain as a territory governed by a single administrative authority. It is usually a set of hosts, resources, and networks inside a company, a department or a branch. Four key entities compose an administrative domain: a set of users (*u*), a set of services or agent systems(*s*), a security token service (*ts*) and a security token base (*tb*). For the sake of simplicity, we regard a user as a special type of service because they have similar behaviors in a system when they both act as an invoker (subject). Here, we call a service an invoker when

it is invoking an access, otherwise an invokee. A security token service is an administrative authority (service) which issues, authenticates, and verifies the security tokens. A security token base is also an administrative authority which gives the users a unified management interface. The services that a security token base provides to the users may vary in different applications, but in common, there are still fundamental functionalities that should always be supported. For example, in order for the invokers to select an appropriate portion of privileges (security tokens) for delegation, a security token base has to implement a mechanism to forecast the required privileges to ensure the invocation. Although our approach allows the security token service/base to be distributed at different sites, we still consider them as a whole. Thus, whenever we mention "a" security token service/base, it does not imply that a centralized solution is applied.

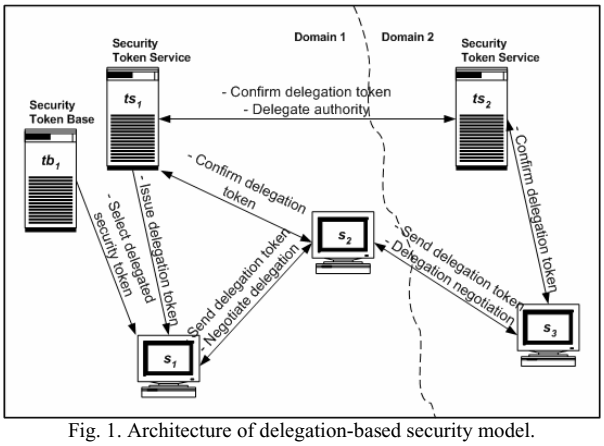

Fig. 1. Architecture of delegation-based security model.

A security token is a set of claims ([OAS06]). OASIS Web Security standard allows a security token to be digitally signed (X.509, SAML, Kerberos, and REL) or unsigned (username). In this paper, we assume that a security token is signed. To interpret a delegation and thus ensure a secured service invocation, we define a dedicated security token, called delegation token, to describe the delegation constraints. To distinguish delegation tokens from other tokens, we define an authorization token as one which describes the invoker's attributes or properties for an access. In our design, an access is allowed if and only if both the delegation and authorization tokens are proved to be legitimate. Another type of message exchanged in our system is called the delegation request. It is used when the invoker has to re-delegate additional privileges to complete the access. In the delegation negotiation process, it is required that the invoker proves the eligibility or necessity of the delegation request.

Consider the example shown in Figure 1. Service s_1 accesses s_2, and s_2, in turn, accesses s_3. Several key processes are needed in the delegation-based security model to support such a composed service scenario. Before s_1 invokes s_2, s_1 must choose a set of authorization tokens that are to be delegated and submit a request to its local security token service ts_1. This process is called the **delegation token retrieval**. It can be started by either the original server s_1, or the intermediate services s_2. The latter case may occur during the delegation negotiation process. A **delegation confirmation process** is started by the invoked service (s_2 or s_3) upon receiving an access request. For example, s_3 should submit all the delegation tokens along the service chain to its local security token service ts_2 for confirmation. ts_2 may interact with ts_1 to confirm the delegation when it lacks the necessary information. To confirm the delegation, both the authenticity and eligibility have to be verified. Here, authenticity check determines whether the delegation token is indeed issued by the authority it claims; whereas the eligibility check determines whether the delegation token conforms to the security constraints in both domains. Another key process is **delegation negotiation**, which is started by the invokee. Consider s_2 accesses s_3. When the delegated authorities are insufficient for ts_2 to grant the access, s_3 will send a delegation request back to s_2 with the related authorization policies attached to prove the necessity of the request. s_2 may choose to delegate its own privileges to continue with current invocation, or forward the request back to s_1.

To ensure that a delegation token is securely issued and used in a service call, authentication alone is insufficient. Thus, we need to define constraints so that delegations are used in a restricted manner. These constraints include delegation capabilities and delegation policies. In the rest of this paper, we simply use "capability" and "policy" for these two terms if not specifically indicated. A capability is a static constraint which can only be updated by a system administrator. A policy describes the system requirements regarding delegation in a more generalized way. It includes many constraints that cannot be represented by static constraints. For example, the system may require that no services can make a delegation beyond the regular office hour. A capability is not capable to express this constraint. Although a capability can be written as a policy, our approach does not encourage this. But, sometimes a policy may confine a value range for a capability. For instance, a policy may define that the capability "depth" of any user with the role "engineer" has the range of value from 0 to 3. Consider the example illustrated in Figure 1. In this example, either delegation capabilities or policies may be defined in the following cases: (1) when ts_1 issues a delegation token to s_1 (delegation from original user/service); (2) when ts_1 issues a delegation token to s_2 if s_2 requests (delegation from intermediate service); (3) when ts_1 confirms the delegation tokens for s_2, or ts_2 confirms for s_3; and (4) when a system administrator tries to assign a value to a particular delegation capability. In our approach, delegation policies are represented in XACML. Using the similar approach introduced in Section 2, we are able to describe delegation policies using the

basic XACML constructs with few extensions. We will discuss the details in Section 4.

4. DELEGATION PROTOCOL FOR WEB SERVICES

4.1 *Selection of Delegated Authorization Tokens*

In Figure 2, the invocation between s_1 and s_2 is made on behalf of the user. Therefore, s_1 does not present its own authorization tokens to s_2; instead, the token delegated from the user is used. The user knows what authorization token is required to access s_1, but has little knowledge about s_2 if the invocation is cross-domain. Although the security token services in different domains may periodically exchange such information with each other; it is still difficult to have the up-to-date information. Without having this knowledge in advance, the failure rate of transactions may increase dramatically as the length of invocation chain increases. Thus, the user has to invoke service s_1 at least twice in order to have some knowledge about the required tokens for accessing s_2. This is not acceptable in the case when the invocations are heavily nested or the service invocations are unpredictable. One possible solution to this problem is to delegate all of the user's security tokens ([YIA96]). This is also not acceptable from the security perspective, because the involving services are assumed to be mutually suspicious and thus delegating unnecessary security tokens brings the potential of misusing these tokens.

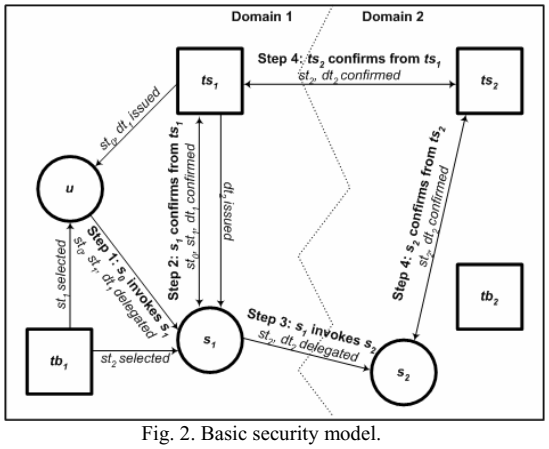

Fig. 2. Basic security model.

We propose to set up a dedicated security token base for an administrative domain to provide uniformed security token management to all the internal entities (Figure 3). Such a framework is flexible and algorithm independent and can accommodate any solutions to this problem. Relevantly, users and services request from the security token base to determine the tokens that are to be delegated. In our design, a security token is still issued directly to a user/service; instead, a copy of the token will be sent by the security token service to the security token base. Thus, neither the user nor the service can write tokens to security token base.

4.2 *Delegation Token*

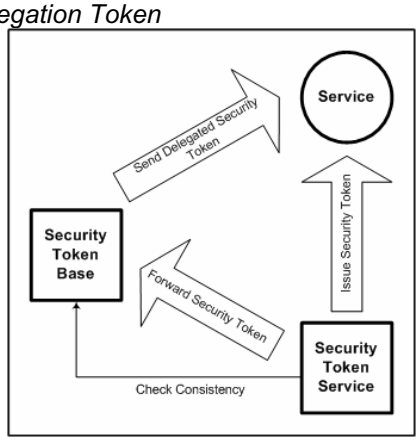

Fig. 3. Security token base.

In our model a user/service can request for an authorization token from any authority. For example, u (Figure 2) can have the authorization tokens from both ts_1 (local) and ts_2 (external). But a user/service can only request a delegation token from its local security token service. A delegation token defines the following attributes: the identities of delegater, delegatee, and issuer, a set of delegation capabilities (see Section 4.4), delegated authorization tokens, and a link to the previous delegation token along the delegation chain.

When there is a need to delegate several tokens, the user or the service sends a request to its security token service together with a list of labels of tokens which are to be delegated. The security token service decides what constraints (delegation capabilities) are put on the issued delegation token based on the delegation policies and endorses it. These delegation capabilities may include: delegability, count, multiplicity, depth, agreement type, and duration. For simplicity, we assume that these delegation capabilities are defined for one delegation token instead of for each delegated authorization token. Also, usually a delegation capability defined in a delegation token is the intersection of those capabilities defined in all included authorization tokens. Lots of work has been conducted in the area of role-based delegation; similar concepts can be brought here ([YIN04], [YE04]). We will discuss the details of delegation capabilities in Section 4.4.

In the scenario depicted in Figure 2, the security token service ts_1 will issue two delegation tokens. User u will submit a request with a list of delegated tokens to its local security token service ts_1. ts_1 will then issue a delegation token $dt(u,s_1)$ such as:

$$< (dom_1, u), (dom_1, s_1), (dom_1, ts_1), delegable, 10, 2, 5, uni, active, start\_time, end\_time, st_u^1, st_u^2, ..., st_u^n, nil >$$

Here, st_u^n denotes the i^{th} authorization token delegated by u and the link to previous delegation token is set to *nil* since it is the first one. When s_1 decides to further delegate these tokens, it has to include the previous delegation token so that ts_1 can determine whether the requested tokens are allowed. s_1 can decide to only delegate a portion or all of the

authorization tokens delegated from u; sometimes, s_1 may also delegate its own authorization tokens to s_2. The delegation token $dt(s_1,s_2)$ has the following format:

$$< (dom_1, s_1), (dom_1, s_2), (dom_1, ts_1), delegable, 10, 2, 5, uni, active, start\_time, end\_time, st^1_{s_1}, st^2_{s_1}, ..., st^n_{s_1}, dt_{u,s_1} >$$

4.3 *Claim Confirmation*

Confirmation of Delegation Token – The delegation confirmation process determines whether the delegation is eligible. Such a process involves four issues: (1) Whether the delegation token is indeed issued by the claimed issuer, (2) Whether the delegation conforms to the delegation capabilities defined in the delegation token, (3) Whether the delegation conforms to the delegation policies defined in all the domains along the delegation chain, and (4) Whether the delegation capabilities defined in the delegation token conform to the above delegation policies. To ensure that an invoked service is correctly delegated the authorization tokens, all the delegations along the delegation chain have to be confirmed. There are three possible ways to perform this confirmation. Consider an n-length delegation chain which starts from service s_1 and ends at service s_{n+1}. The most straightforward solution to confirm the delegation tokens received by s_n is to simply extract all the delegation tokens and confirm them from all the security token services involved. This solution has a computation complexity $O(n^2)$. A lightweight solution is that s_n only confirm the last delegation token $dt(s_{n-1},s_n)$ (possibly also $dt(s_n,s_n)$ for delegation negotiation). This solution assumes that the invoking service s_n has already confirmed all the delegation tokens before it. Although this solution greatly reduces the computation complexity; it is based on the assumption that s_{n+1} trusts s_n. An adaptive solution can be given based on the trust value between s_{n+1} and s_n. For example s_{n+1} performs a complete confirmation if and only if the trust value s_{n+1} has for s_n is greater than a threshold t. Figure 4 gives the algorithm of this trust-based confirmation.

```
Algorithm: ConfirmDelegation
Input: a delegation token dt, invoked service srv, trust
threshold t
Output: decision d   {YES, NO}

1.  WHILE  dt ≠ nil DO
2.     delegatee = getDelegatee (dt)
3.     delegator = getDelegator (dt)
4.     trust = getTrust (delegator, srv)
5.     IF delegatee = srv OR trust < t
6.        IF authenDelegation (dt, delegatee, delegator) = false
7.           return NO
8.     DC[] = extractDelegationConstraints (dt)
9.     DS = getDelegationStatus (dt, srv)
10.    FOR i = 1 to length (DC) DO
11.       IF confirmDelegationConstraints (DS, DC[i]) = false
12.          return NO
13.    dt = extractPreviousDelegationToken (dt)
14. return YES.
```

Fig. 4. Confirmation of delegation.

Confirmation of Authorization Token – The confirmation of authorization token is the process (Figure 5) that proves all the claims defined in the delegated tokens are truthful and the invocation is eligible based on all these presented claims. The process of authorization token confirmation does not return a binary value. Besides a positive or negative result, it also returns a marginal value which indicates an undetermined access control decision. Usually, such a returned value will trigger the invoked service to send back a notification to the invoker; and the invoker can decide whether to delegate additional privileges or terminate the transaction.

Cross-domain Confirmation – A cross-domain confirmation includes the confirmations of both delegation and authorization tokens. The confirmation of delegation token is done by its issuer which is the security token service inside the invoker's (ts_1) domain; while the confirmation of authorization tokens is done by the security token service inside the invokee's (ts_2) domain. From the standpoint of s_2, ts_2 is a local security token service and thus a trusted party; while ts_1 is an external security token service, thus s_2 cannot entirely rely on the confirmation result given by ts_1. In our design, a domain trust value must be taken into account. When its trust value is lower, the probability of giving a negative result will increase.

```
Algorithm: ConfirmAuthToken
Input: a delegation token dt, invoked service srv, access
control policy p
Output: decision d   {YES, NO, NOTIFICATION}, missing claim
set MCS[]

1.  WHILE  dt ≠ nil DO
2.     ST[] = appendDelegatedTokens (dt)
3.     dt = extractPreviousDelegationToken (dt)
4.  ST[] = mergeAuthTokenList (ST[])
5.  FOR i = 1 to length (ST) DO
6.     issuer = getIssuer (ST[i])
7.     IF authenAuthToken (ST[i], issuer) = false
8.     return NO
9.  RC[] = extractRequiredClaimsFromPolicy (p)
10. DC[] = extractClaimsFromAuthToken (ST)
11. IF accessAuthToken (DC, RC, MCS) = false
12.    d = NOTIFICATION
13. return d, MCS.
```

Fig. 5. Confirmation of security token.

4.4 *Delegation Capabilities and Policies*

Consider an example illustrated in Figure 2. A user u calls service s_1, and in turn, s_1 invokes s_2. u and s_1 are located in the same domain d_1, and s_2 is within a different domain d_2. Let ts_1 and ts_2 be the security token services for domain d_1 and d_2, respectively. Consider the following scenario. u and s_1 only have a part of the required authorization tokens to perform the requested task on s_2, and for this access to be legitimate, both tokens have to present. In this case, both u and s_1 have to delegate its authorization tokens. Thus, the set of tokens that s_2 receives includes two parts: u's delegation

token $dt(u,s_1)$ and s_1's delegation token $dt(s_1,s_1)$. This means that s_1 uses its own privileges to invoke s_2, therefore, it is considered as from s_1 to s_1 (denoted by $dt(s_1,s_1)$) instead of from s_1 to s_2 (denoted by $dt(s_1,s_2)$).Before s_2 evaluates each delegated authorization token, s_2 has to ensure that the current delegations ($dt(u,s_1)$ and $dt(s_1,s_1)$) are eligible. As we have discussed in previous sections, the eligibility of a delegation token includes two parts, authentication and verification. The problem of authentication has already been widely studied in the literature. Verification is the process of confirming whether a delegation meets the constraints (delegation capabilities and policies) that are defined for each involved entity.

Delegation capabilities are the static constraints that can be assigned. We consider several common capabilities: delegability, depth, duration, count, multiplicity, and agreement type. Delegability defines whether a delegation can be issued on this entity (user, service or authorization token). Depth is used to constrain whether and how many levels a delegated authorization token can be further delegated to other entities. With repeated delegations from one entity to another, a delegation chain is thus formed. Depth defines the maximal length of this delegation chain. When the depth is set to 1, the delegated authorization tokens can never be further delegated; and if the depth is set to 0, this entity is not delegable. Duration is the maximal period that the delegation can remain active. Count refers to the maximal number of delegations that can be made. Multiplicity is another similar capability which defines the maximal number of entities that can be simultaneously delegated. Sometimes it is desirable to delegate to a set of entities at once. This capability is used to constrain this type of delegation. Agreement type defines whether the delegator and delegatee have to come to an agreement before this delegation can be issued. Agreement type can be either bilateral or unilateral. Bilateral agreement is an agreement type wherein the delegation is eligible only if both of the delegator and delegatee agree; whereas unilateral agreement does not require the delegatee to present its "approval" ([BAR00]). Although our framework defines the above general delegation capabilities, it does not restrict the system designer to define additional capabilities.

An entity may act as different roles in different delegations. We define a delegation role as the role that an entity plays as in a delegation. In a delegation within a service chain, we define three types of delegation roles, including the original delegator, the intermediate delegator, and the delegatee. Same constraints for different delegation roles are considered as different constraints. Consider the delegation capabilities defined in the previous paragraph. An original delegator can have all the capabilities; while a delegatee can not have count or multiplicity.

A delegation policy defines further restrictions beyond what are provided by delegation capabilities. In our framework, we allow a security token service to issue a delegation token for multiple delegators, delegatees and authorization tokens. This delegation token is considered as an aggregated delegation token which defines a set of delegations. Conceptually, we restrict that a delegation is from a specific delegator to a specific delegatee and only for delegating one authorization tokens. In this sense, a delegation (represented by a delegation token) involves five entities including a delegator (either original or intermediate), a delegatee, a delegated authorization token, a parameterized domain/system, and an issued delegation token. XACML is not capable of representing such a delegation policy with five entities, because there are only four elements which can be used to describe the participants in an access. They are <Subjects>, <Resources>, <Actions>, and <Environments>. Fortunately, in our framework it is possible to compact some of the entities in a delegation so that XACML can be used to describe a delegation policy. To enable this, we combine the delegation token together with the delegated authorization token defined in it. Because the capabilities of a delegation token essentially defines the actual constraints of this delegated authorization token in a particular service invocation. Therefore, it is reasonable that we regard the capabilities of a delegation token as the new capabilities for the delegated authorization token. Another minor difficulty is how we can distinguish between an original and an intermediate delegator. In our approach, the capabilities for an original or intermediate delegator are simply regarded as different attributes of the same entity.

We use XACML in our framework to specify delegation policies. Here, we show how the key entities in our model can be mapped into the data flow diagram in XACML. A delegator (either original or intermediate) and its attributes are always written in element <Subject>, and included by element <Subjects> to represent a set of delegators. In a similar way, a delegatee can be represented in <Resource> and <Resources>, a delegated authorization token can be represented in <Action> and <Actions>, and a parameterized domain can be described in <Environment> and <Environments>. The capabilities defined in a delegation token can be incorporated into element <Action> as different attributes defined for that authorization token. In the following, we provide an example to illustrate how delegation policies can be represented in XACML.

The policy says: for any services acting as an intermediate delegator, the issued delegation token must be specified two additional capabilities, count and depth, wherein the count must be not more than 10 and depth not more than 5, and delegating an authorization token has to be evaluated as low risk. In this example, this issued delegation token should also be defined additional delegation capabilities within which count and depth are constrained by the policy. This policy is shown in Figure 6.

4.5 *Delegation Negotiation*

When the delegated authorization tokens are insufficient to access the invoked service, a delegation negotiation process will be initiated.

```
.....
<Rule RuleId="urn:utdallas:delegation:policy:ruleX" Effect="Permit">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">
      True
     </AttributeValue>
     <SubjectAttributeDesignator
     AttributeId="urn:utdallas:delegation:attribute:intermediate-delegability"
     DataType="http://www.w3.org/2001/XMLSchema#boolean"/>
    </SubjectMatch>
   </Subject>
  </Subjects>
  <Actions>
   <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
      Low
     </AttributeValue>
     <ActionAttributeDesignator
     AttributeId="urn:utdallas:delegation:attribute:risk-level"
     DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </ActionMatch>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">
      10
     </AttributeValue>
     <ActionAttributeDesignator
     AttributeId="urn:utdallas:delegation:attribute:new-count"
     DataType="http://www.w3.org/2001/XMLSchema#integer"/>
    </ActionMatch>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">
      5
     </AttributeValue>
     <ActionAttributeDesignator
     AttributeId="urn:utdallas:delegation:attribute:new-depth"
     DataType="http://www.w3.org/2001/XMLSchema#integer"/>
    </ActionMatch>
   </Action>
  </Actions>
 </Target>
</Rule>
……
```

Fig. 6. A delegation policy written in XACML.

There are two possible solutions for negotiation process: (1) direct negotiation with the initiator, and (2) recursive negotiation from service to service. The first solution is simpler but not so ideal especially in the cases where a long delegation chain is involved. The solution that initiator delegates more security tokens has few differences with restarting the current transaction which may lower the system performance. Another major problem is that the internal process of an invoked service should be transparent to the entity who invokes it; thus, it is more reasonable that the initiator has no knowledge about the services in the delegation chain after the first one he invokes. The recursive negotiation solution has even worse communication overhead comparing to the direct negotiation. To reduce the communication overhead, it is required that the intermediate services delegate their own tokens. Based on the trust values, a service may decide to (1) forward the notification to the invoker, (2) request redelegation from its security token service, or (3) terminate the transaction. Moreover, the invoking service has to confirm that the request is indeed issued by the invoked service and the requested tokens are necessary to continue with the access. To achieve this goal, the delegation request has to be digitally signed and attached a list of related security policies. Security token services have to maintain the relationships between policies and services to enable the confirmation of dependencies. Figure 7 shows a delegation negotiation algorithm in a single web service.

Algorithm: DelegationNegotiation
Input: notification message msg, missing claim set MCS[], requester r
Output: decision d {FAIL, SUCCESS, FORWARD}, redelegated security token list STL[]

```
1.  IF authentNotificationMessage (msg, MCS[]) = false
2.    return FAIL
3.  IF STL[] = hasRequestedSecurityToken (MCS[])≠ nil
4.    IF trust (r) = true
5.      return SUCCESS, STL[]
6.    ELSE
7.      return FORWARD, msg, MCS[]
8.  ELSE
9.    IF trust (r) = true AND STL[] = requestSecurityToken (MCS [])≠ nil
10.     return SUCCESS, STL[]
11.   ELSE
12.     return FORWARD, msg, MCS[].
```

Fig. 7. Delegation negotiation for single service.

4.6 *Revocation*

Our approach supports two types of revocations: (1) revocation using duration restriction; (2) user revocation ([ZHA03]). Security token service assigns an expiration time for each delegation token such that this delegation will be disabled after the expiration time. Furthermore, security token service assigns each delegated security token an expiration time; thus different security tokens may have

different active durations if different values are assigned. The user revocation is a cascaded revocation ([ZHA03]). For example, if a delegation token *dt* is disabled; all the delegation tokens in the delegation chain after *dt* will be revoked. There are two approaches for implementing a cascaded revocation: instant revocation and dilatory revocation ([ZHA03]). An instant revocation performs the cascaded revocation immediately after the first revocation takes place. On the contrary, a dilatory revocation revokes a delegation token only when a delegation confirmation is requested. Because of our trust-based confirmation process, the dilatory revocation can not be applied here. The user revocation has to be instant.

5. EVALUATION-BASED ACCESS CONTROL FRAMEWORK

We first introduce our hierarchical structure of administrative domains. As defined in Section 3, an administrative domain or a domain is a group of hosts, resources, and networks governed by a single administrative authority which is the security token service in our model. In our framework, the internet can be divided into thousands of domains and every entity has to reside within at least one domain. We assume that domains have empty intersections because their overlapping may bring managerial and technical difficulties. Several domains can be united into a single one, and in turn, such a united domain can further form a union with others. For distinguishing, we give these unions of domains a dedicated term, "abstract domain"; and those domains that cannot be divided any further are called "functional domain". For example, both university of Texas at Dallas and university of Texas at Austin may form a functional domain respectively; and UT system including these two universities forms an abstract domain. A security token service for a functional domain is called a "local" security token service (*ts*); and a security token service for an abstract domain is called a "global" security token service (*gs*). The functionalities of local and global security token services cannot be integrated in one web service. Furthermore, a global security token service is a public security service and thus always outside any functional domain. A functional domain usually has more than one local security token services; similarly, an abstract domain can also contain more than one global security token services. Abstract domains may sometimes provide public services to the functional domains within; these services together with global security token service network can be regarded as a special functional domain. Figure 8 depicts the hierarchical structure of administrative domains.

We use a tree to represent the hierarchical structure of administrative domains. In this tree, a leaf is a set of application services which are governed by a single local security token service network. An intermediary node is either a local or a global security token service. An edge represents a "governed by" relationship. We can say a collection of application services *x* is governed by a local security token service network *y*, and in turn, *y* is governed by a global security token service *z*. The set of application services can also be governed by a global security token service. We give a direction to the edges in such a service tree. A directed edge from a child to its parent is called a "delegate" (this is different from what we discuss in previous sections, for such delegation, readers can refer to [LI03], [LI99]). A directed edge from a parent to its children is called an "issue". An application service delegates all of its authorities to its local security token service. The reason that we fully delegate a service's authorities to the local security token service is that we expect the security policies to be enforced solely in the local security token service. Local security token service can delegate either part or all of its authorities to its parent, which is a global security token service. Security policies are defined by each application service, and integrated at local security token service which provides conflict resolution mechanism to guarantee the consistency. Security policies at local security token services will in turn be integrated at its parent level with conflict resolution. Thus, in the example in Figure 2, s_1 does not have to request security tokens directly from ts_2, since ts_2 may delegate part of its authorities to ts_1 through their common parents.

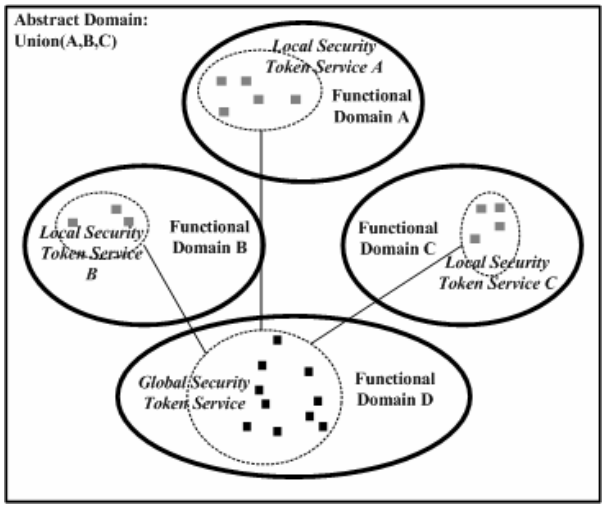

Fig. 8. Hierarchical structure of administrative domains.

We present an example to illustrate the bottom-up delegation of authority and top-down issue of security token. An application service defines the required security tokens to approve the access and authorizes its local security token service to manage them. In our example, ts_1 is authorized t_1 and t_2, ts_2 is authorized t_3, ts_3 is authorized t_5 and t_6, and gs_2 is authorized t_7 since it also acts as a local security token service. A security token service can decide to retain or delegate part or all of its authority to its parent. For example, ts_1 retains t_2 and delegates t_1 to gs_1. Such a decision is determined by ts_1's security policy and ts_1's evaluation of gs_1. Similarly, ts_2 delegates t_3, and gs_1 delegates t_1 and t_3, ts_3

retains t_6 and delegates t_5, and finally, gs_2 retains t_5 and delegates t_7.

The issue of a security token is initiated either by an application's request or via a top-down approach. Similarly with delegation, a security token service can decide to issue or retain a security token based on its policy and evaluation. In our example, gs_3 issues all of its security tokens to its children. Because of different evaluations, gs_1 issues t_7 to ts_2 but not ts_1. After such a propagation schema, the security token bases for all these administrative domains will have the following tokens: $\{t_1, t_2, t_3\}$ in s_1's, $\{t_1, t_3, t_7\}$ in s_2's, $\{t_1, t_3, t_5, t_6, t_7\}$ in s_3 and s_4's, and $\{t_1, t_3, t_5, t_7\}$ in s_5's. The evaluations which the delegation and issue of security token are based on are between different security token services; they essentially imply the mutual evaluations between different domains, and thus can be called domain-level evaluations. An evaluation can be a trust value, a credit or a contribution estimate.

Sometimes, an invoking service is requested to present a security token which is retained by some service in the path to the invoked service. For example, s_1 is requested by s_3 to present token t_6 which is retained by ts_3. In this case, s_1 has to negotiate with ts_3. s_1 requests its local security token service ts_1 to negotiate on its behalf. It is more reasonable that ts_3 makes the decision based its evaluation of s_1 instead of ts_1 since s_1 is the ultimate requester. Thus, all the security token services have to maintain a table of service-level evaluations. A problem is what a security token service's evaluation of an application service stands for. For example, what does ts_3's evaluation of s_1 (denote by $E(ts_3,s_1)$) mean? It is not applicable since they are not directly communicating with each other. In this case, $E(ts_3,s_1)$ is essential s_3's evaluation of s_1 (denoted by $E(s_3,s_1)$). Thus, all the application services have to manage their mutual evaluations. Security token services will integrate these evaluations and make decisions based on them.

6. ILLUSTRATIVE EXAMPLE

We now present an example that illustrates an application of our model in a scenario in which services are dynamically invoked. Table I provides a list of application services. To enable our application to automatically invoke the best service, their performances are estimated. In our example, a student named Jenny from University of Texas at Dallas wants to download some reports to help write her paper. There are four sites (SR_1 through SR_4) which share the reports she requests. But she cannot request from these sites directly but invoke service DR_1 or DR_2 to download the reports from these sites. The security token service in her university manages a matrix which stores the mutual trust values among these services and users (Table II). The row and the column are the entities involved in this transaction. The element in row i and column j (denoted by $M[i,j]$) refers to entity i's trust value of entity j. The trust value is represented by integer. The entity j with a larger value of $M[i,j]$ is more trustworthy to i. Table III lists the required security token for accessing each services. Table IV lists the security tokens that the user and services have on hand. Consider the following three cases.

Case 1: Jenny initiates the transaction at 9:00AM on Monday. Service DR_1 is chosen because of its better service quality. Jenny requests a delegation token encapsulating her security token "student role" from the local security token service and successfully invokes service DR_1 since the delegation is confirmed. DR_1 locates the best available service SR_1 and request a delegation token from the local security token service. The delegation is successfully confirmed by SR_1 but the security token that DR_1 presents is not sufficient to invoke the service; thus SR_1 sends back a notification requesting a security token "librarian role". On receiving the notification, DR_1 has to make the decision based on its trust values. DR_1 determines that Jenny has a higher trust value (8), thus delegates its own security token "librarian role" to invoke the service SR_1. Therefore, the transaction is successful.

TABLE I
SERVICE LIST

Domain label	Service label	Location	Latency	Service quality	Service time/date
utdtxus	DR_1	utd,tx,us	10ms	Good	8am-5pm, Mon-Fri
utdtxus	DR_2	utd,tx,us	10ms	Average	24 hrs, 7 days
utatxus	SR_1	uta,tx,us	20ms	Good	8am-5pm Mon-Fri
neumaus	SR_2	neu,ma,us	40ms	Average	24 hrs, Mon-Fri
wsuwaus	SR_3	wsu,wa,us	60ms	Average	24 hrs, 7 days
icluk	SR_4	icl,uk	200ms	Poor	24 hrs, 7 days

TABLE II
MUTUAL TRUST VALUES

	Jenny	DR_1	DR_2	SR_1	SR_2	SR_3	SR_4
Jenny	N/A	9	4	?	?	?	?
DR_1	8	N/A	?	8	5	5	5
DR_2	4	?	N/A	8	8	6	4
SR_1	?	8	2	N/A	?	?	?
SR_2	?	3	4	?	N/A	?	?
SR_3	?	8	4	?	?	N/A	?
SR_4	?	6	4	?	?	?	N/A

TABLE III
REQUIRED ROLES OF SERVICES

Service	DR_1	DR_2	SR_1	SR_2	SR_3	SR_4
Role	Student	Student	Librarian	Librarian	Student	Student

TABLE IV
AVAILABLE ROLES OF USER AND SERVICES

Service/user	DR_1	DR_2	Jenny
Role	Librarian	Librarian	Student

Case 2: Jenny initiates the transaction at 11:00PM on Monday. Service DR_2 is chosen. Jenny invokes service DR_2 and delegates her security token "student role" to DR_2. Jenny has the required role and the delegation is proved. DR_2 locates the best available service "SR_2" and starts the invocation. Since Jenny does not have the required security token "librarian role", SR_2 will send back a notification, and DR_2 has to make the decision whether to delegate its own

security token. Because DR_2 does not trust Jenny (trust value is 4), DR_2 does not delegate its own token and the notification will be forwarded to Jenny; since Jenny does not have the required security token, the transaction fails.

Case 3: Jenny initiates the transaction on Saturday. Jenny invokes service DR_2, since DR_1 is not available on weekends. Security token "student role" will be delegated to DR_2 to invoke service SR_3 since it is the best service available at that time. Because the delegated security token "student role" is sufficient to access service SR_3 and the delegation is confirmed, service SR_3 accepts the invocation and the transaction is thus successful.

7. CONCLUSION

In recent years, the flexibility, interoperability and automated composition of web services are greatly enhanced as more sophisticated models, standards and specifications are emerging. The web service security now becomes the major hurdle. Security models for individual web services now have difficulties in securing the composite services, and/or the dynamically invoked services. In such a highly dynamic environment, current WS access control framework cannot ensure that information is correctly revealed to the user.

Our delegation-based security model is to address this issue. We design a protocol to support delegation between users and services from the determination of delegated security token, through the format of delegation token and confirmation of claim, until the delegation negotiation process. We also discuss other related problems such as reliability of security token services and revocation schema. We define the hierarchical domain structure and thus provide a hierarchical access control framework. Evaluations such as trust, credit and contribution are also brought into this framework to support making the access control decisions. This paper defines the delegation problem in detail, and gives an abstract model to secure web services via delegation. Major issues within such an infrastructure are identified and briefly discussed. Future work such as descriptive language development and proofs can be based on these discussions.

8. REFERENCES

[BAR00] E. Barka, and R. S. Sandhu. "Framework for role-based delegation models". 16th Annual Computer Security Applications Conference. pp. 168. 2000.

[BAR10] E. S. Barka, and R. S. Sandhu. "A role-based delegation model and some extensions". 23rd National Information Systems Security Conference. 2000.

[BER06] E. Bertino, A. C. Squicciarini, I. Paloscia, and L. Martino. "Ws-AC: a fine grained access control system for web services". World Wide Web. Vol. 9. No. 2. pp. 143-171. Jun., 2006.

[BHA05] R. Bhatti, E. Bertino, and A. Ghafoor. "A trust-based context-aware access control model for web-services". Distributed and Parallel Databases. Vol. 18. No. 1. pp. 83-105. 2005.

[BHA07] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Corin. "Secure sessions for web services". ACM Transactions on Information and System Security. Vol. 10. Issue 12. 2007.

[GAN04] Gang Yin, Huai-min Wang, Dian-xi Shi, Yan Jia, and Meng Teng. "A rule-based framework for role-based constrained delegation". Proc. of the 3rd international conference on Information security. pp. 186-191. 2004.

[GAS90] M. Gasser, and E. McDermott. "An architecture for practical delegation in a distributed system". The Proc. of IEEE Symposium on Security and Privacy, p. 20, 1990.

[KOR00] Y. Kortesniemi, and T. Hasu. "A revocation, validation and authentication protocol for SPKI based delegation systems". Network and Distributed Systems Security Symposium. 2000.

[LI03] N. Li, B. N. Grosof, and J. Feigenbaum. "Delegation logic: a logic-based approach to distributed authorization". ACM Transactions on Information and System Security. Vol. 6. No. 1. pp. 128-171. 2003.

[LI99] N. Li, J. Feigenbaum, and B. N. Grosof. "A logic-based knowledge representation for authorization with delegation". IEEE Computer Security Foundations Workshop. pp. 162. 1999.

[OAS06] OASIS. "Web Services Security: SOAP Message Security 1.1". http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. Feb., 2006.

[OAS07] OASIS. "WS-Trust 1.3". http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf. Mar., 2007.

[VAR91] V. Varadharajan, P. Allen, and S. Black. "An analysis of the proxy problem in distributed systems". The Proc. of IEEE Symposium on Research in Security and Privacy. pp. 255-275. 1991.

[WEL04] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. P earlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. "X. 509 proxy certificates for dynamic delegation". 3rd Annual PK I R&D Workshop. 2004.

[WOO98] T. Y.C. Woo, and S. S. Lam. "Designing a distributed authorization service". Proc. of IEEE INFOCOM'98 San Francisco, CA. pp. 419-429. Ma., 1998.

[YE04] Chunxiao Ye, Yuqing Fu, and Zhongfu Wu. "An attribute based delegation model". Proc. of the 3rd international conference on Information security. pp. 220-221. 2004.

[YIA96] N. Yialelis, and M. Sloman. "A security framework supporting domain-based access control in distributed systems". Proc. of Symposium on Network and Distributed System Security. p. 26. 1996.

[YIN04] Gang Yin, Huai-min Wang, Dian-xi Shi, Yan Jia, and Meng Teng. "A rule-based framework for role-based constrained delegation". Proc. of the 3rd international conference on information security. pp. 186-191. 2004.

[ZHA03] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. "A rule-based framework for role-based delegation and revocation". ACM Transactions on Information and System Security. Vol. 6. Issue 3. pp. 404-441. 2003.

[ZHN03] X. Zhang, S. Oh, and R. S. Sandhu. "PBDM: a flexible delegation model in RBAC". Proc. of the 8th ACM symposium on Access control models and technologies. pp. 149-157. 2003.