Abstract. In recent years, the issues in web service security have been widely investigated and various security standards have been proposed. But most of these standards and frameworks focus on the access control policies for individual web services and do not consider the access issues in composed services. Consider a simple service chain where service s₁ accesses s₂, and s₂, in turn, accesses service s₃. The information returned from s₁ to s₂ may be used to compute some results that are further returned to s₁. The current web service security framework does not provide any mechanisms to control such an information flow, and hence, sensitive information may be leaked to s₁ without the consent of s₂.

In this paper, we propose an enhanced security model to facilitate the control of information flow through service chains. It extends the basic security models by introducing the concepts of delegation and pass-on. Based on these concepts, new certificates, certificate chain, delegation and pass-on policies, and how they are used to control the information flow are discussed.

1. INTRODUCTION

Service oriented architecture (SOA) is a popular paradigm for system integration and interoperation. Web service is the current standard for SOA. While SOA has many benefits, security is still a major concern. A lot of efforts have been devoted to provide a better security mechanism in a web service environment. One of the major advances in recent years is the set of WS security specifications proposed by various standards groups and organizations including OASIS, WS-I, IBM, etc [OAS06] [IBM06]. WS Security 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 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.

Besides establishing standards, there have been other research works for extending the access control models in web service environment. Web service environment is open, distributed, heterogeneous, integrated, and mutually suspicious in nature, and involves new processes such as the publishing, discovery, and binding of web services which also need to be secured. R. Bhatti et al. proposed to incorporate contextual information, such as time, location, or environmental state, into WS access control models [BHA05]. In this model, they also proposed the dynamic trust level of a user, which is initialized by a trust establishing procedure, and adjusted by context in the access. In [BER06], E. Bertino et al. proposed to carry on negotiation in an access by specifying the message types and contents exchanged based on the agreement about security requirements and services reached by the requestor and the service provider, Skogsrud et al. proposed in [SKO03] to apply the solutions of trust negotiation into web service environments to secure the interactions among unknown users. Rao et al. proposed in [RAO04] a security model to secure the process of publishing, searching, and binding a service.

Most of the existing works focus on individual web services and do not consider service composition. Advanced WS security techniques allow a service to effectively control the accesses to its resources, but do not provide mechanisms to let the service control further flow. Consider an example where service s₁ invokes s₂, and in turn, s₂ invokes service s₃. Assume that s₁ has sensitive information that can be accessed by s₂ but not s₁. Following current web service framework, s₃ will evaluate s₂’s access solely based on s₂’s credential without considering s₁’s attendance in this transaction. Thus, the data on s₁ may be incorrectly revealed to s₁. Because s₂ has no knowledge about s₁’s policies, it is difficult to enforce s₁’s security policies at s₂’s side. This information flow problem in web service composition has not yet been considered in the literature. A better solution is that s₁ evaluates s₂’s request before sending out the data, based on s₂’s credential plus the information of the attending parties who may indirectly receive the disseminated data in a raw or processed form.

Another problem in existing WS security frameworks is the lack of direct supports to delegation, especially to delegation chain. For example, a hospital employs two web services ws_analyzer and ws_storage to help manage its medical data. ws_analyzer is an analyzer who performs analysis upon third party data, while ws_storage simply provides disk spaces to store medical information. Assume that the hospital invokes ws_analyzer to perform analysis upon the data that it stores at ws_storage. In this case, ws_analyzer has to have the access to ws_storage; while it is inappropriate for ws_storage to issue certificates to ws_analyzer, since ws_analyzer is not entitled to access these data beyond the current hospital transactions. To enable controlled formation of service chains, it is necessary to support delegation mechanisms. There are lots of works about delegation reported in the literature [GAS90] [KOR00] [WEL04] [YIA96] [YIN04] [ZHA03], and some of them also discussed conditional and constrained delegation. However, none of them provide a complete solution to carry out delegation in a secured way.

This paper proposes to use delegation and pass-on to secure interactions in a composed web service. We first examine the
global service environment from the perspective of the conventional “individualized” WS architecture, and develop a generalized system model consisting domains, services, and resources. The global system is assumed to be formed by several disjoint domains in which services and resources are defined. Each domain has its own certificate authority (CA) and verifier (VE). We assume the non-existence of third party mediators, and the mediation (authentication and verification) is done by interactions between CAs and VEs of the attending domains. Then, we formally define an enhanced WS security model which incorporates additional types of certificates to express the concepts of delegation and pass-on. These certificates are carefully defined and explained in details. In a service chain, delegation and pass-on certificates can also form a certificate chain whose eligibility is the basis of the access decision in our system. In this model, pass-on is the default action for the attending services in a service chain; while, delegation can only be enabled based on request (delegation request). We also provide the definition of our policy system, and explain how delegation and pass-on policies can be represented using existing policy languages. We give a detailed system flow which describes how an access request is generated by the invoker and evaluated at the invokee’s side, and also provide a brief discussion about the verification of a certificate chain.

The organization of the rest of the paper is as follow. The definitions of our generalized system model and the service chain model are presented in Section 2. The key constructs of the enhanced WS security model is detailed in Section 3, including the definitions of certificates, certificate chain, request types, and the policy system. Section 4 gives a formal definition of the enhanced system model, and provides a general system flow. This paper is concluded in Section 5.

2. MODEL OF THE WS SYSTEM

2.1 A System Model

In a WS system, we consider services, resources, domains, and certificates as four key entities. Users and other applications are also considered as services. By service invocation, resources of one service are used by another. We focus on information resources, which can be any types of raw or processed data in arbitrary forms. Domain is a collection of services and resources administrated by an authority. In our system, domains are disjoint, and each domain only contains a single authority managing a set of services and resources.

Our system is certificate-based in the sense that all the access control decisions have to be made based on the certificate that the requester presents to the invokee. A certificate is a set of assertions (claims) that an authority (in a domain) makes about a particular service. Our system model is formally defined in the following.

DEFINITION 2.1. System Model. A system \( \mathcal{S} \) is a three-tuple \( (D, S, R) \).

1. \( D = \{d_1, d_2, \ldots\} \) is a set of disjoint domains. Each domain \( d_i \) has a certificate authority \( d_i.ca \) which makes assertions and issues certificates, a verifier \( d_i.vr \) which verifies accesses to the resources in \( d_i \), and a set of knowledge \( d_i.K = \{d_i.k_1, d_i.k_1, k_2, \ldots\} \). A knowledge, \( d_i.k_i \) is either a rule or a piece of supporting information managed in domain \( d_i \) that can be used for producing an access control decision.

2. \( S = \{s_1, s_2, \ldots\} \) is a set of services, in which each element \( s_i \) is a service provided in a domain. The domain of service \( s_i \) is denoted as \( \text{dom}(s_i) \), where \( \text{dom}(s_i) \in D \).

3. \( R = \{r_1, r_2, \ldots\} \) is a set of resources. A resource \( r_i \) is an identifiable object stored in domain \( d_i \) and has atomic access. The domain of resource \( r_i \) is denoted as \( \text{dom}(r_i) \), where \( \text{dom}(r_i) \in D \).

4. Each service \( s_i \) owns a set of certificates \( s_i.C = \{s_i.c_1, s_i.c_2, \ldots\} \). The issuer of this certificate is \( d_m.ca \) where \( d_m \) is a certain domain.

2.2 A Service Chain Model

When several individual services are composed together, it is called a composite service and those that compose it are the component services. When discussing the security issues, we only consider one request served by the composite service. Multiple requests can be handled the same way, one at a time. The component services, including the initial requester and the service invocations form a service graph, in which each component service is represented by a vertex and service invocation by a directed edge. In this paper, we further simplify a service graph into a service chain. This simplification is only for making the discussion clearer. The solution we discuss is applicable to general service graphs.

DEFINITION 2.2. Service Chain. In a system, a length-\( n \) service chain is a sequence of services \( (s_1, s_2, \ldots, s_n) \) (\( V1 \leq i \leq n, s_i \in S \)), which satisfies the following properties:

1. \( s_1 \) is the initial requester.
2. \( \forall i, j, 1 \leq i, j \leq n, i \neq j \Rightarrow s_i \neq s_j \).
3. Service invocation only occurs between \( s_i \) and \( s_{i+1} \) (\( 1 \leq i \leq n-1 \)), and \( s_1 \) is called an invoker and \( s_{n+1} \) is called an invokee.

3. ENHANCED SECURITY MODEL FOR WEB SERVICES

In conventional service access models, each service invocation in a service chain is treated as an independent event. Consider a service chain \( (s_1, s_2, s_3, s_4) \). Assume that \( s_1 \) has the privileges to access \( s_3 \) and \( s_4 \), but \( s_2 \) does not have the privilege to access \( s_3 \) and \( s_3 \) does not have the privilege to access \( s_4 \). In existing service invocation protocols, there is no way to let \( s_2 \) access \( s_3 \) or let \( s_3 \) access \( s_4 \). On the other hand, if \( s_2 \) does not have the privileges to access \( s_3 \) or \( s_4 \) but \( s_2 \) does, then \( s_2 \) can access critical resources of \( s_3 \) and \( s_4 \) and pass them on to \( s_1 \) either in the processed form or in their original form without concerning whether this may cause undesired information leakage. In other words, the two directions of information flows in conventional service invocation protocols are not symmetric; the service invocation direction has strict constraints while the returning direction is fully unprotected. If for the second case, \( s_1 \) has the privilege to access \( s_2 \) implies \( s_2 \) can trust \( s_1 \) and pass the execution results derived from the critical data (or even the raw data) of \( s_3 \) and \( s_4 \) to \( s_1 \), then for the first case, \( s_1 \) should be allowed to delegate its privilege to \( s_2 \) and \( s_3 \) to allow their invocations to \( s_3 \) and \( s_4 \), respectively. If such delegation requires special certificates, then passing back the data derived from the critical
information of $s_3$ and $s_4$ to $s_1$, who had no privilege to access the critical data of $s_3$ and $s_4$, should only be allowed when a special certificate is issued to consent the pass on action.

In this paper, we design a new access control protocol for service invocations that empowers the services and their domains to control the accesses more effectively. A service chain embeds the information flow. In the forward flow, the information of the requesters cumulates and flow toward the service providers. For the service chain example $(s_1, s_2, s_3, s_4)$, the request $s_2$ sends to $s_3$ may contain information computed from $s_1$’s request. On the return path, the information returned from the service providers cumulates and flow toward the requesters. For $(s_1, s_2, s_3, s_4)$, the response returned from $s_2$ to $s_1$ may contain data that can be used to derive the information returned from $s_4$ to $s_3$ and from $s_3$ to $s_2$. In our protocol, we design two types of certificates, delegation and pass-on certificates, to support delegation and control pass-on actions. The delegation certificate allows a service $x$ to delegate some of its privileges to another service $y$ so that $y$ can act on behalf of $x$ to access other services. The pass-on certificate is introduced to provide a way for a service $x$ to authorize a service $y$ to further propagate its request information and also to allow a service $u$ to authorize another service $v$ to passed on its returned information to other services. Relative to existing protocols, the delegation mechanism makes it possible to form some service chains that are not possible in conventional protocols. The pass-on mechanism takes away the default power of passing on request/returned information to unknown services and makes the pass-on of information a specifically given instead of a default privilege.

In the first example given above, $s_1$ can issue a delegation certificate to $s_2$ to allow $s_2$ to access $s_3$, and $s_2$ needs to delegate the privileges obtained from $s_1$ to $s_3$ to allow $s_3$ to access $s_4$. These two correlated delegation certificates form a certificate chain. In the second example, $s_2$ needs to present a pass-on certificate to $s_3$ and $s_4$, to certify a “pass-on” relationship from $s_2$ to $s_1$, so that $s_3$ and $s_4$ can decide whether to allow the access with the information flow from $s_2$ to $s_1$. Pass-on certificate can also form a certificate chain. Certificate chains should be handled carefully. Assume that $s_3$ is a malicious service and it tries to retrieve unauthorized information from $s_4$. $s_3$ may make use of $s_1$’s privilege for accessing $s_4$ and let $s_1$ indirectly delegate the privilege to it even though $s_1$ may not trust $s_3$. To prevent this, we can either allow $s_1$ to confine the delegation chain size or allow $s_3$ to specify a “black list” in the delegation certificate. Or, $s_4$ can prohibit the accesses through delegation. Similarly, the certificate chain size or a black list can be specified in a pass-on certificate as well. To achieve this, we use the concept of conditions to parameterize services, resources, domains and certificates. For example, a service can define a condition: $\text{open hour} \geq 9\text{AM}$ and $\text{open hour} \leq 5\text{PM}$, which states that any access before 9AM or after 5PM will be denied. Also, we can define a certificate as un-delegable or un-passable to disable delegation or pass-on.

Consider service chain $(s_1, s_2, s_3, s_4)$. Assume that $s_2$ has the privileges to access $s_3$ and $s_4$, but $s_1$ does not have the privileges to access $s_3$ and $s_4$, and $s_4$ does not have the privilege to access $s_4$. We show how these certificates can be used to complete the service invocations. In our system, access with a pass-on certificate is considered as default, while the delegation certificate can only be issued by request (delegation request). In this example, on receiving an access request from $s_1$, $s_2$ determines it needs to invoke $s_3$ to finish the tasks. $s_2$ issues a pass-on certificate by default before invoking $s_3$ using its own privileges. After evaluating related policies, $s_3$ may generate three possible decisions. On generating a grant decision, the service invocations will continue to $s_4$; while giving a deny decision simply means that the service invocation is not allowed and the service call will fail. Beside these two possibilities, $s_3$ will also and probably produce an undetermined decision which will be returned to $s_2$ followed by a delegation request. Let assume that $s_3$ produces a grant decision. In this case $s_3$ will continue to invoke $s_4$, and also by default, issue a pass-on certificate for the flow from $s_3$ to $s_2$. For $s_3$’s access, $s_4$ will generate an undetermined decision due to insufficient privileges. Also, a delegation request will be issued and forwarded back to $s_2$. Using this delegation request, $s_2$ can request its certificate authority to issue a delegation request, or it can also simply stop the service invocation based on its security policies. Assume that $s_2$ successfully has the issued delegation certificate. Thus, such a service call succeeds with the requested data items flowing from $s_4$ back to $s_1$. In this case, the pass-on certificate $s_2 (\text{dom}(s_1), \text{ca})$ issued to $s_3$, and the delegation certificate $s_2 (\text{dom}(s_2), \text{ca})$ issued to $s_3$ form a certificate chain which will be first verified at $s_4$.

In the following, we first provide formal definitions to the above concepts, including authorization, delegation, and pass-on certificates as well as certificate chains. Several request types are defined subsequently based on these constructs. Policy system is introduced next with an illustrative example. A formal delegation-based system model is proposed at last followed by some further explanation about this system.

### 3.1 Certificates

This section defines three types of certificates involved in our system: authorization, delegation, and pass-on certificates. Conditions are used to parameterize all these certificates and, claims are included in these certificates to specify properties.

In the following, we adopt a bottom-up methodology by first providing the definition of an expression, then the claim and the condition which incorporate it. The authorization, delegation, and pass-on certificates are defined next, and used to formalize the certificate chain.

**Definition 3.1. Expression, Claim, and Condition.** An expression $E$ has the form $\text{attr} \circ \text{val}$, where $\text{attr}$ is an attribute describing a certain characteristic (e.g. name, age, ip address, etc.), $\text{val}$ is a value, and $\circ$ is any comparison operator.

1. A claim is an expression made by a certificate authority or a security officer to assert a property of an entity. A claim has the form $\text{claimer:recipient;E}$ in which, $\text{claimer}$ is the entity who makes such a claim, $\text{recipient}$ is the entity who is asserted to hold such a property, and $E$ is an expression.
2. A condition is an expression made by a certificate authority or a
security officer to state a prerequisite requirement for an access to be granted. A condition has the form claimer:decision;E, where claimer is the entity who asserts such a condition, decision is a Boolean value which can be true (granted) or false (denied), and E is an expression.

Claims and conditions are different. A claim essentially states a truth as long as it is not expired. For example, the claim \( \text{ca\_common: sslibrary:service\_provider\_name = "UtlDbLib"} \) will be true if the certificate that includes it stays unexpired. However, a condition gives a pre-requisite, which does not always hold. On access, such a condition is to be evaluated by the certificate authority, and if it holds, the decision included in the condition will be given. The condition \( \text{ca\_common: true:delegation\_chain\_length \leq 10} \) means that the length of the delegation chain for the access has to be less than or equal to 10 for the access to be granted.

**Definition 3.2. Authorization Certificate.** A service \( s_i \) owns a set of authorization certificates. Each authorization certificate \( s_i, ac_i \) is defined as \( s_i, ac_i, CLM, s_i, ac_i, CON \), where \( s_i, ac_i, CLM = \{ s_i, ac_i, clm_1, s_i, ac_i, clm_2, \ldots \} \) is a set of claims, and \( s_i, ac_i, CON = \{ s_i, ac_i, con_1, s_i, ac_i, con_2, \ldots \} \) is a set of conditions.

Figure 1 depicts an authorization certificate which certifies the service’s name, clearance level, user type, and delegation type which are to be evaluated based on access control policies. It also specifies two conditions which lead to a positive decision if satisfied.

![Fig. 1. Authorization certificate.](image1.png)

Like authorization certificates, delegation and pass-on certificates can also be defined by a set of claims and a set of conditions. But in a delegation or pass-on certificate, claims can only be defined to describe the properties of the certificate. For example, the claim \( \text{ca\_common: sslibrary:issu date = "03/02/2008"} \) specifies the issue date of the certificate. Delegation certificate is defined to represent a delegation relation between two services. Also, each delegation has to carry at least one authorization certificate. Thus, we need to include additional fields: delegator, delegatee, and a set of authorization certificates. Moreover, to support the concept of a certificate chain which is needed for delegation and pass-on actions, we need to include a pointer that can be used to reference another delegation or pass-on certificate. We formally define delegation certificate as follows.

**Definition 3.3. Delegation Certificate.** A service \( s_i \) owns a set of delegation certificates. Each delegation certificate \( s_i, dc_i \) is defined as \( s_i, dc_i, rcurv, s_i, dc_i, CLM, s_i, dc_i, CON \), \( s_i, dc_i, AC, s_i, dc_i, pred \).  
1. \( s_i, dc_i, rcurv \) is the delegatee who is delegated the authorization certificates specified in \( s_i, dc_i, AC \). \( s_i, dc_i, rcurv \in S \). In a service chain \( \{ s_i, s_2, \ldots, s_n \} \), \( s_i, dc_i, rcurv = s_{i+1} \).
2. \( s_i, dc_i, CLM = \{ s_i, dc_i, clm_1, s_i, dc_i, clm_2, \ldots \} \) is a set of claims made by \( \text{dom}(s_i) \).

The pass-on certificate is quite similar to the delegation certificate, except that it does not carry any authorization certificates and it has an opposite flow direction. Consider the service chain: \( \{ s_1, s_2, s_3 \} \). When \( s_2 \) access \( s_3 \) using its own privileges, it has to issue a pass-on certificate representing its pass-on relationship to \( s_1 \). In this case, \( s_1 \) is defined as the receiver of the certificate. Its formal definition comes after.

**Definition 3.4. Pass-on Certificate.** A service \( s_i \) owns a set of pass-on certificates \( s_i, pcc_i \) which is defined as \( s_i, pcc_i, rcurv, s_i, pcc_i, CLM, s_i, pcc_i, CON, s_i, pcc_i, pred \).  
1. \( s_i, pcc_i, rcurv \) is the service who receives the data item passed. \( s_i, pcc_i, rcurv \in S \). In service chain \( \{ s_1, s_2, \ldots, s_n \}, s_i, pcc_i, rcurv = s_{i+1} \).
2. \( s_i, pcc_i, CLM = \{ s_i, pcc_i, clm_1, s_i, pcc_i, clm_2, \ldots \} \) is a set of claims made by \( \text{dom}(s_i) \).
3. \( s_i, pcc_i, CON = \{ s_i, pcc_i, con_1, s_i, pcc_i, con_2, \ldots \} \) is a set of conditions made by \( \text{dom}(s_i) \).
4. \( s_i, pcc_i, pred \) is an identifier which can be used to reference a
delegation or pass-on certificate which is the predecessor of $s_i$, PC$_i$ in the service chain.

Figure 3 depicts a pass-on certificate which is quite similar to the delegation certificate shown in Figure 2, except that there are no authorization certificates included.

### 3.2 Certificate Chain

Given a service chain $(s_1, s_2, ..., s_n)$. Consider the last service invocation between $s_{n-1}$ and $s_n$. We claim that to verify whether the access is valid, $s_n$ has to verify the set of authorization certificates presented to $s_n$, and all the involved delegation and pass-on certificates along the service chain, because the same delegation or pass-on certificate may result in different access decisions at different sites. For example, $s_i$ issues a pass-on certificate and presents it to $s_{i+1}$. $s_{i+1}$ grants this access because $s_{i-1}$ is included in a “white list” at site $s_{i+1}$. While, $s_n$ may deny this request if it believes that info about a disseminated data item will be ultimately delivered to a service $s_{n-1}$ that is not trusted by $s_n$. The authorization certificates that $s_{n-1}$ presents to $s_i$ $(2 \leq i \leq n - 1)$ do not need to be evaluated again at $s_n$ since they only concern the service invocations at site $s_i$. As can be seen, all the access requests involved in a service chain have to include a certificate chain containing a sequence of delegation or pass-on certificates. In the following, we define the concept of a certificate chain.

**Definition 3.4. Certificate Chain.** A certificate chain $\mathcal{C}$ is defined as a sequence of certificates $(s_1, \ldots, s_n)$ in which $s_i$ is either a delegation or pass-on certificate. $\mathcal{C}$ also satisfies:

1. $c_i.@pred = nil$.
2. $\forall 1 < i \leq n$. $c_i.@pred = c_{i-1}$.
3. $\forall 1 < i \leq n$, if $c_{i-1}$ and $c_i$ are both delegation certificates, $c_i.\mathcal{A}C \subseteq dc_{i-1}.\mathcal{A}C$.
4. $\forall 1 < i \leq n$, if $c_{i-1}$ and $c_i$ are both delegation certificates, given any two conditions $a, b$, $a \in c_i.\mathcal{C}ON$. $b \in c_{i-1}.\mathcal{C}ON$. if $a, b$ have the same attributes, then $a \Rightarrow b$.

The 4th condition in Definition 3.4 states the following restriction in a certificate chain: the conditions defined in $i$th delegation certificate must be more restrictive than those in $(i-1)$th delegation certificate.

### 3.3 Access and Delegation Requests

This section defines the access request and delegation request. In our system, an access request contains a set of authorization certificates which are used for current access, and a certificate chain up to the last delegation or pass-on certificate. If the last certificate in the certificate chain is a pass-on certificate, then the owner of the access request is simply the owner of the pass-on certificate. For example, assuming that in a service chain $(s_1, s_2, s_3, s_4)$, $s_3$ accesses $s_4$ via a pass-on certificate $s_2, PC_2$, then the owner of this access request is simply $s_3$. On the other hand, if the last certificate in the chain is a delegation certificate, then the owner of the access request should be the initial delegate of those authorization certificates used. Assuming that in service chain $(s_1, s_2, s_3, s_4)$, both $s_3$ and $s_4$ are accessed via delegation ($s_1$ delegates $s_2$ to access $s_3$, and $s_2$ delegates $s_3$ to access $s_4$). In this case, the owner of the last access request is the initial requester, $s_1$. The owner of the access request as well as the requester should both be taken into consideration when the access is under evaluation.

There are two possibilities that an access may fail: the certificates presented to the invoke are invalid, or the required certificates for the access are absent. For the latter case, an undetermined decision will be made, and a delegation request will be generated.

**Definition 3.5. Access Request.** An access request $AR$ is defined as a five-tuple $(O, R, I, \mathcal{A}, \mathcal{C})$.

1. $O$ is the owner of $AR$.
2. $I$ is the current requester of $AR$.
3. $\mathcal{A} = \{a_1, a_2, \ldots\}$ is a set of authorization certificates used for the access.
4. $\mathcal{C} = (c_1, c_2, \ldots)$ is a certificate chain up to the last delegation or pass-on certificate.

**Definition 3.6. Delegation Request.** A delegation request $DR$ is defined as a four-tuple $(O, R, I, \mathcal{A}, \mathcal{D})$.

1. $O$ is the owner of the access request.
2. $R$ is the requester of this access request.
3. $\mathcal{A} = \{a_1, a_2, \ldots\}$ is a set of missing attributes which cause the access request to be evaluated as “undetermined”.
4. $\mathcal{D} = \{p_1, p_2, \ldots\}$ is a set in which each element $p_i$ is a tuple (pol, $A$), where $A \subseteq \mathcal{A}TR$. pol is a policy rule, and $A$ is a set of missing attributes according to rule pol.

### 3.4 Delegation and Pass-on Policies

In our system, the certificate chain presented to a service is verified prior to the authorization certificates. To validate a certificate chain, all the included delegation and pass-on certificates have to be evaluated. Thus, policies have to be defined to guide the certificate evaluation and decision process. Because both delegation and pass-on follow a different concept from that of conventional access control systems, existing policy specification languages (e.g. XACML, etc.) are insufficient for describing these two behaviors. In this section, we introduce two dataflow models for delegation and pass-on, and based on them, the delegation and pass-on policies are formally defined and interpreted using extended XACML.

#### 3.4.1 XACML

XACML is a declarative policy specification language based on XML tags. It defines a well known dataflow model with four key entities: subject, resource, action, and environment. Subject (described by tags (Subject) and (Subjects)) is the actor who performs the access and generates the request. Resource (described by (Resource) and (Resources)) is a protected data item. Action (described by (Action) and (Actions)) is a specific operation on a certain resource, and environment (described by (Environment) and (Environments)) simply refers to those system conditions independent of a particular subject, resource, or action. In XACML, a policy is defined as a set of rules (described by (Rule)) in which the tag (Target) is used to incorporate all the conditions that have to be evaluated for the decision as specified in attribute “Effect”. XACML also defines four types of “match” statements ((SubjectMatch), (ResourceMatch), (ActionMatch), and (EnvironmentMatch)) to specify...
expressions as defined in our Definition 3.1.

3.4.2 Delegation Policies

Delegator, delegatee, and authorization and delegation certificates are four key entities in a delegation action. There are two points where delegation policies should be evaluated: before a delegation certificate is issued and before the access. The latter case follows the conventional access control diagram, and does not need policy language extension. But the former case follows a new diagram. For the sake of simplicity, we assume that only one authorization certificate is delegated and only one data item flows between two services; multiple authorization certificates and multiple data items can be represented simply by several policy rules.

DEFINITION 3.7. Delegation Policy. A delegation policy is defined as a set of clauses, each clause is defined as a six-tuple (Condarg, Condgtg, Condrec, Connder, Conndec):
1. Condarg, Condgtg, Connder, and Conndec stand for the conditions of delegator, delegatee, and delegation and authorization certificates.
2. Condrec stands for the conditions that system environment has to meet for the policy to be applicable.
3. Dec is the access decision (Boolean).

This definition incorporates delegation and authorization certificates into a delegation policy. In reality, it is sufficient to keep one of them because in a service chain, the authorization certificate is used either through a delegation or by itself, but cannot be both. When an authorization certificate is delegated, the issued delegation certificate inherits all the conditions defined for the authorization certificate and thus replaces the latter when the access is evaluated. This implies that upon access, either the conditions of the authorization certificate or those of the delegation certificate will be considered instead of both. Therefore, the delegation policy can be further reduced to the form (Condarg, Condgtg, Connder, Conndec). And, through simple mapping, such a delegation policy can be easily represented using XACML.

In the following, we first show how the key entities in a delegation can be mapped into the dataflow diagram in XACML. A delegator and its conditions are always written in element (Subject) and included by element (Subjects) to represent a set of delegators. In a similar way, a delegatee is represented in (Resource) and then included in (Resources), and an authorization/delegation certificate is written in (Action) included in (Actions) with an additional attribute to indicate whether it is an authorization certificate or a delegation certificate. There is no need to map system environment.

Moreover, we need to define several additional attributes in order to specify a claim or a condition based on Definition 3.1. These attributes will be included in four “match” tags, since we need to represent the claimer, recipient and/or decision. Also, we need an additional attribute to distinguish between a claim and a condition, since in our policy system they will be verified in different ways. More specifically, a claim usually represents a truth that is asserted by an authority, thus to verify a claim, the cross domain authentication is required. While, a condition usually specifies what status current service chain or service invocation has to be in for the access to be allowed; such a status should be determined by the invokee’s certificate authority, thus cross domain authentication is not necessary. Therefore, we define the following attributes: ClaimerId, RecipientId, SubEffect, and SubType to represent the claimer, recipient, decision, and expression type (claim or condition). These attributes are to be specified in the four “match” statements as we introduce above (SubjectMatch, ResourceMatch, ActionMatch, EnvironmentMatch).

The original attribute “Effect” specified in (Rule) now only stands for the decision for the evaluation of claims. Because all the conditions specified in a policy rule have their own decisions (sub-decisions), we also need to define how these sub-decisions can be combined to form an overall decision. For the sake of simplicity, we now simply allow “AND” and “OR” operations for these sub-decisions, and provide attribute “overall” to hold such information. If an “AND” is specified, a logical-AND operation will be performed for all the sub-decisions (1 for grant, and 0 for deny); and vice versa.

In Figure 4, we provide an example to illustrate how a delegation policy can be written in XACML. The policy says: for any services acting as a delegator, the issued delegation certificate cannot be further delegated to others.

---

Fig. 4. A delegation policy written in XACML.

3.4.3 Pass-on Policies

The delegation policy focuses on how to issue a valid delegation certificate between two services, which is from the perspective of the delegator and the delegatee. However, pass-on policy is from the perspective of a service chain, simply because it concerns whether the information flow between two services is valid. Such an information flow may be formed by several continuous service invocations by delegation or pass-on. The key entities involved in a pass-on action are the requested resources, and receiver and invoker services. The actual access instead happens between the invoker and invokee services where pass-on policies are enforced. Receiver service is an arbitrary service in the service chain, and is proved to ultimately have the requested resource via a pass-on relationship to the invoker service. Such a pass-on relationship can be formed by a chain of pass-on or
delegation certificates or their combination. Obviously, such an information flow also exists in a delegation action.

**Definition 3.8. Pass-on Policy.** A pass-on policy is defined as a set of clauses, each clause is defined as a five-tuple \((\text{Con}_{\text{own}}, \text{Con}_{\text{erv}}, \text{Con}_{\text{expenses}}, \text{Dec})\) in which:
1. \(\text{Con}_{\text{own}}, \text{Con}_{\text{erv}}, \) and \(\text{Con}_{\text{expenses}}\) stand for the conditions that the owner of the data item, the receiver of the data item, and the data item (resources) itself have to be met respectively.
2. \(\text{Dec}\) stands for the conditions that system environment has to meet for the policy to be applicable.
3. \(\text{Dec}\) is the access decision (Boolean).

To evaluate whether the data flow along a service chain complies to a pass-on policy, the key is to determine which service is the owner of the transferred data item. Consider the service chain: \((s_1, s_2, s_3, s_4)\). Assume that \(s_3\) retrieves a data item \(r_1\) from \(s_4\), and creates a new one \(r_3\), which is later delivered to \(s_2\) and \(s_1\). It’s difficult to determine the owner of \(r_3\). If \(r_3\) is a copy of \(r_4\), then its owner is simply \(s_4\). If it is not possible to get any partial information of \(r_4\) from \(r_3\), then \(r_3\) \(\in\) \(S_3\). It becomes even more difficult since the access decision is made before the actual data flow, which means that \(r_3\) may not exist when the access is evaluated. A simple solution to it is to classify the resource into raw data and processed data. The owner of raw data will never change, and that of a processed data item is always its producer, even if it may still reveal the information about the original raw data.

XACML can be used to represent a pass-on policy using a similar mapping mechanism as we show for the delegation policy. To make them distinguishable, we define the type attribute in element (Rule) to indicate whether it is a delegation policy or a pass-on policy. When type = "pass_on", the following mapping mechanism is applied: (Subject) and (Subjects) represent the owner of the data item, (Action) and (Actions) represent the receiver of the data item, and (Resource) and (Resources) refer to the data item passed. Figure 5 shows a policy rule written in XACML which says that the secret data cannot be passed to untrusted parties.

**SYSTEM FLOW**

4.1 **An Enhanced System Model**

In this section, an enhanced system model is formally defined to achieve better security control in WS systems. This definition is an enhancement of the system model given in Definition 2.1. It is also based on the definitions given in the above sections. In the enhanced system, the certificate set \(C\) is divided into three disjoint subsets for authorization, delegation, and pass-on certificates. All the entities in our system are described by claims and conditions. Also, each domain defines a set of delegation policies and a set of pass-on policies. We present our formal definition in the following.

**Definition 4.1. Enhanced System.** A system \(\mathfrak{S}\) is a four-tuple \((D, S, R, C)\).

1. \(D = \{d_1, d_2, \ldots\}\) is a set of disjoint domains. In our enhanced system, each domain \(d_i\) contains the following components: a set of claims \(d_i, \text{CLM} = \{d_i, \text{cm}_{1}, d_i, \text{cm}_{2}, \ldots\}\), a set of conditions \(d_i, \text{CON} = \{d_i, \text{con}_{1}, d_i, \text{con}_{2}, \ldots\}\), a certificate authority \(d_i, \text{ca}\) which makes assertions and issues certificates, a verifier \(d_i, \text{ve}\) which verifies accesses to the resources in \(d_i\); a set of delegation policies, \(d_i, \text{DPOL} = \{d_i, \text{dpo}_{1}, d_i, \text{dpo}_{2}, \ldots\}\), and a set of pass-on policies, \(d_i, \text{PPOL} = \{d_i, \text{ppo}_{1}, d_i, \text{ppo}_{2}, \ldots\}\).
2. \(S = \{s_1, s_2, \ldots\}\) is a set of services, in which each element \(s_i\) is a service provided in a domain. The domain of service \(s_i\) is denoted as \(\text{dom}(s_i)\), where \(\text{dom}(s_i) \in D\). Each service \(s_i\) is defined by a set of claims \(s_i, \text{CLM} = \{s_i, \text{cm}_{1}, s_i, \text{cm}_{2}, \ldots\}\), and a set of conditions \(s_i, \text{CON} = \{s_i, \text{con}_{1}, s_i, \text{con}_{2}, \ldots\}\).
3. \(R = \{r_1, r_2, \ldots\}\) is a set of resources. A resource \(r_i\) is an identifiable object stored in domain \(d_i\) and has atomic access. The domain of resource \(r_i\) is denoted as \(\text{dom}(r_i)\), where \(\text{dom}(r_i) \in D\). Each resource \(r_i\) is defined by a set of claims \(r_i, \text{CLM} = \{r_i, \text{cm}_{1}, r_i, \text{cm}_{2}, \ldots\}\) and a set of conditions \(r_i, \text{CON} = \{r_i, \text{con}_{1}, r_i, \text{con}_{2}, \ldots\}\).
4. Each service \(s_i\) owns a set of authorization certificates, \(s_i, \text{AC} = \{s_i, \text{ac}_{1}, s_i, \text{ac}_{2}, \ldots\}\), a set of delegation certificates, \(s_i, \text{DC} = \{s_i, \text{dc}_{1}, s_i, \text{dc}_{2}, \ldots\}\), and a set of pass-on certificates, \(s_i, \text{PC} = \{s_i, \text{pc}_{1}, s_i, \text{pc}_{2}, \ldots\}\).

4.2 **General System Flow**

Consider service \(s_i\) in a service chain \((s_1, s_2, \ldots, s_n)\). Since \(s_i\) is not the initial requester, \(s_i\) has to access \(s_{i+1}\) via either delegation or pass-on. Since the authorization infrastructure is not our focus and there are a lot of existing research works to deal with the issues, we assume that a trust assessment framework and an authorization infrastructure are in place to make authorization decisions and issue certificates. Our system also assumes pass-on as a default action when extending a service chain. As we show in Section 3, service \(s_i\) always generates the access request first via requesting a pass-on certificate from its local authority \(\text{dom}(s_i).\text{ca}\). \(\text{dom}(s_i).\text{ca}\) can only issue a delegation certificate based on a delegation request generated by \(s_{i+1}\) ‘s authority \(\text{dom}(s_{i+1}).\text{ca}\). Such a procedure is the general system flow that is applied in our system. The following shows the general system flow in a service chain \((s_1, s_2, \ldots, s_n)\).
1. Set \(i = 1\), initial invoker \(s_1\) creates an access request \((s_1, s_i, s_{i+1}, \{s_1, \text{ac}_{1}, s_1, \text{ac}_{2}, \ldots\}, \text{nil})\) and invokes \(s_2\).
2. Increment \(i\) by 1.
3. \(s_i\) requests its certificate authority \(\text{dom}(s_i).\text{ca}\) to issue a
pass-on certificate whose receiver is \( s_{i-1} \).

4. \( \text{dom}(s_i).\text{ca} \) issues a pass-on certificate \( s_i, pc_i = (s_{i-1}, \{s_{i-1}.pc_i, clm_{i-1}, s_{i-1}.pc_i, clm_{i-2}, \ldots, s_j, pc_j, pred \}) \), where \( s_j, pc_j, pred \) is the delegation or pass-on certificate that was used by \( s_{i-1} \) to access \( s_i \).

5. \( s_i \) invokes \( s_{i+1} \) by generating an access request \( (s_i, s_i.s_{i+1}, \{s_i, ac_i, s_i, ac_{i-2}, \ldots, s_j, pc_j \}) \).

6. \( s_i \) sends the authorization certificates \( \{s_i, ac_i, s_i, ac_{i-2}, \ldots, s_j, pc_j \} \) to \( \text{dom}(s_{i+1}).\text{ca} \) for verification.

7. \( s_{i+1} \) sends the certificate chain \( (s_i, pc_i) \) to \( \text{dom}(s_{i+1}).\text{ve} \) for verification, and evaluates the access based on its pass-on policies \( \text{dom}(s_{i+1}).\text{PPOL} \).

8. If the decision is “grant”, go to 2; if the decision is “deny” due to missing attributes, request missing attributes from \( \text{dom}(s_j).\text{ca} \) where \( 1 \leq j \leq i \) and go to 7, otherwise, continue.

9. \( \text{dom}(s_{i+1}).\text{ve} \) issues a delegation request \( (s_i, s_i, s_{i+1}, \{att_{i-1}, att_{i-2}, \ldots, (pol_i, A_i), (pol_{i-2}, A_{i-2}), \ldots \}) \) and sends to \( s_{i+1} \).

10. \( s_{i+1} \) forwards the delegation request to \( s_{i+1} \) via \( s_i \).

11. \( s_{i-1} \) sends the delegation request to its certificate authority \( \text{dom}(s_{i-1}).\text{ca} \) and requests a delegation certificate.

12. \( \text{dom}(s_{i-1}).\text{ca} \) evaluates the delegation request based on its delegation policies \( \text{dom}(s_{i-1}).\text{DPOL} \).

13. If the decision is “negative”, go to 18, otherwise \( \text{dom}(s_{i-1}).\text{ca} \) issues a delegation certificate \( s_{i-1}.dc_i = (s_i, \{s_{i-2}.dc_i, clm_{i-1}, s_{i-2}.dc_i, clm_{i-2}, \ldots, s_j, dc_j, \ldots \}) \) to \( s_{i-1} \).

14. \( s_{i-1} \) sends the delegation certificate to \( s_i \), and \( s_i \) presents the delegation certificate to \( s_{i+1} \), the access request is changed to \( (s_i, s_i, s_{i+1}, \{att_{i-1}, att_{i-2}, \ldots, (pol_i, A_i), (pol_{i-2}, A_{i-2}), \ldots \}) \) and sends to \( s_{i+1} \).

15. \( s_{i+1} \) sends authorization certificates \( \{s_{i-2}.dc_i, ac_i, s_{i-2}.dc_i, ac_{i-1}, s_{i-2}.dc_i, ac_{i-2}, \ldots, s_j, pc_j, pred \} \) to \( \text{dom}(s_{i+1}).\text{ca} \) for verification.

16. \( s_{i+1} \) sends the certificate chain \( (s_{i-1}.dc_i) \) to \( \text{dom}(s_{i+1}).\text{ve} \) for verification, and evaluates the access based on its delegation policies \( \text{dom}(s_{i+1}).\text{DPOL} \).

17. If the decision is “grant”, go to 2, otherwise, continue.

18. End execution.

4.3 Certificate Chain Verification

Consider the service chain \( (s_1, s_2, \ldots, s_n) \). In our system, a delegation certificate has to be verified repeatedly at different sites. For example, if delegation certificate \( s_3, dc_1 \) is issued and first verified at site \( s_3 \), then it will be verified at all the services \( s_i \) where \( 5 \leq i \leq n \). This is because that a delegation certificate can be revoked at any time in our system, and the evaluation result may be different at different sites. Delegation certificate needs to be verified because the invoker has to ensure that the privileges the requester presents are indeed from the service as claimed in the certificate. But the verification of a pass-on certificate is to ensure that the pass-on relations built up based on the certificate chain is truthful and complete. Essentially, a pass-on certificate describes a direct pass-on relation between two services. The verification procedure is to verify that the invoker is the same as the receiver claimed in the certificate. They also need to be repeatedly verified at different sites. For example, in a service chain \( (s_1, s_2, s_3, s_4, s_5) \). Assume that \( s_2 \) presents \( s_3 \) its pass-on certificate \( s_2, pc_1 \) and have it verified, and \( s_4 \) only trusts \( s_3 \) to a certain degree. In this case, although the access decision is grant, yet \( s_4 \) does not trust \( s_3 \)’s ability of performing such a verification. Thus, the verification has to be done again. This argument also implies that if \( s_4 \) trusts \( s_3 \)’s ability, such a verification can be omitted. As can be seen, a service chain containing \( n \) services has to verify at most \( (n-1)(n-2)/2 \) and at least \( (n-2) \) delegation/pass-on certificates.

5. CONCLUSION

Web service security is a challenging area in SOA environment. Conventional SOA security models focus on individual services and exhibit some problems. In this paper, we present an enhanced security model to provide better control of the information flow among services. We design a protocol to support the delegation and pass-on actions between services, and the decision process is based on the verification of a certificate chain. We also provide a detailed discussion of our policy system including the concepts of delegation and pass-on policies and how they can be written in XACML. Other issues such as how a certificate chain can be effectively verified have also been discussed. The work in this paper provides a foundation for further research and development of a comprehensive security paradigm for well-controlled secure web service interactions.

6. REFERENCES


