FERMILAB-PUB-08-100-CD

An Interaction-based Access Control Model (IBAC) for Collaborative Services

Mine Altunay
Fermi National Laboratory
maltunay@fnal.gov

Gregory T. Byrd, Doug E. Brown, Ralph A. Dean
North Carolina State University
{gbyrd, debrown, ralph_dean}@ncsu.edu

ABSTRACT

A collaboration is a collection of services that work together to achieve a common goal. Although collaborations help when tackling difficult problems, they lead to security issues. First, a collaboration is often performed by services that are drawn from different security domains. Second, a service interacts with multiple peer services during the collaboration. These interactions are not isolated from one another -- e.g., data may flow through a sequence of different services. As a result, a service is exposed to multiple peer services in varying degrees, leading to different security threats. We identify the types of interactions that can be present in collaborations, and discuss the security threats due to each type. We propose a model for representing the collaboration context so that a service can be made aware of the existing interactions. We provide an access control model for a service participating in a collaboration. We couple our access control model with a policy model, so that the access requirements from collaborations can be expressed and evaluated.

KEYWORDS: access control, collaboration context, web services, workflow planning

1. INTRODUCTION

In service-oriented architectures (SOA), a collaboration involves multiple services working together to achieve a common goal. Services are expected to cooperate and interact with each other. Through these interactions, each service exchanges information and accomplishes its own part in the collaboration. Our work focuses on collaborations that include services drawn from different security do-mains that may or may not trust one another.

A collaboration can be realized via many technologies, from mash-ups [10] to scientific workflows [5][6]. All of these technologies use the same SOA principle: defining web services as autonomous end points that partake in a complex application.

Although collaborations are beneficial for tackling difficult problems, they lead to important security issues. One of them is managing access to a participant service. By joining a collaboration, a service agrees to interact with several peer services. Due to these interactions, the service would become subject to varying security threats (described below and in Section 3.1).

The fundamental component of collaboration is an *interaction*: a data transfer between two services, a sender and a recipient, that is triggered by an action taken by the sender service and is ended by an action taken by the recipient service (Figure 1). The action refers to execution of a specific operation of the service (as defined by Web Services Description Languages (WSDL) [18]).

A collaboration includes several interactions among multiple services. These interactions are not isolated from one another; instead, they follow one another to disseminate data and ultimately achieve the collaboration's goal. A specific interaction is affected by other interactions. For example, in Figure 1, the order information sent by the buyer service in Interaction 1

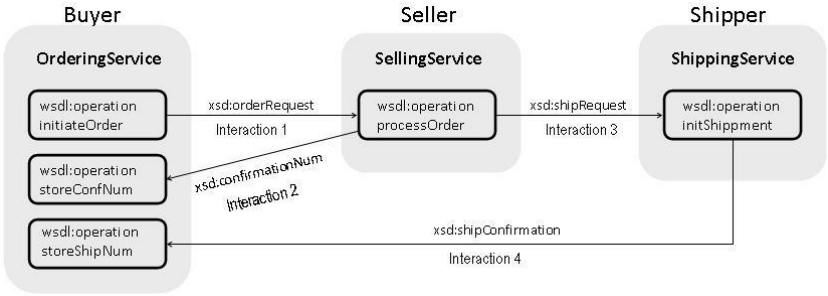


Figure 1. Web Service Collaboration.

affects the outcome of the seller service, which later uses this information to invoke the shipper service (Interaction 3). As a result, the execution of the initShipment operation is also affected by the buyer service.

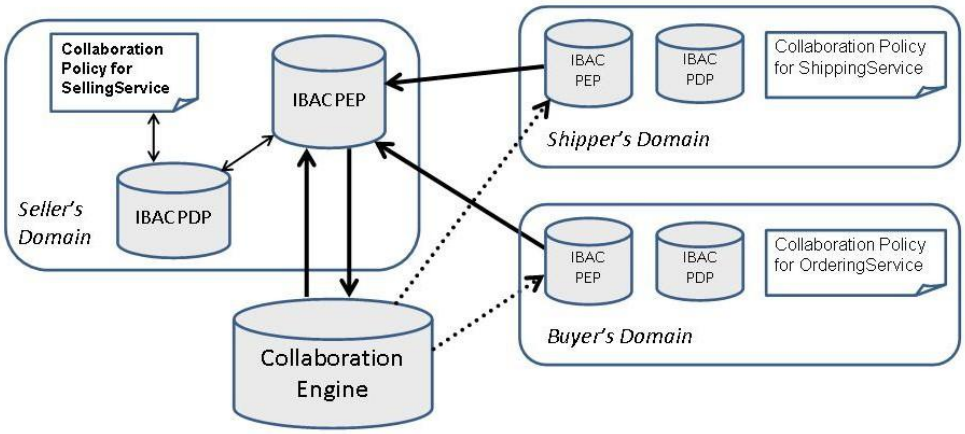


Figure 2. IBAC Framework.

An interaction introduces security threats to both sender and recipient services. From the recipient's view, the sent data may include viruses or Trojan horses, or performing the requested action can expose the recipient's domain to the sender. From the sender's view, the transferred data may be confidential, or the sender's operation may expose the sender's business logic to the recipient. Moreover, a recipient may refuse to interact with a sender which had previously interacted with an un-trusted service. Likewise, a sender may refuse to interact with a recipient that will later interact with an un-trusted service. (In Section 3.2, we will explain how a sender can learn about future interactions of its recipient.)

Consider the seller service in Figure 1, and Interactions 1 and 3, where the seller service acts first as a recipient and then as a sender. For its security, the seller service must evaluate both interactions. The evaluation of each interaction must consider the differences in the roles played by the seller service, the data transferred, and the interacting peer services. The seller service may have different security requirements for allowing invocation of the processOrder operation than for sending some of its output to shipping service.

Our work aims to protect a service in collaboration by managing access control decisions for the service. To achieve this, we propose an *interaction-based access control model (IBAC)*. We model a collaboration in terms of interactions, and define a *collaboration context*. By evaluating the collaboration context, the IBAC model can determine the access decisions for a service. Our work enables a service to evaluate a proposed collaboration context *before* deciding to join the collaboration. If the access decision returned by the IBAC engine is deny, the service refuses the collaboration, because the collaboration contains insecure interactions.

Figure 2 illustrates an overview of our work from the Seller Service's perspective. The collaboration engine (responsible for planning the collaboration) sends invitations (the proposed collaboration context) to three services. Each service has an IBAC engine (PEP+PDP) and a collaboration policy to evaluate a proposed collaboration in terms of security. During the evaluation, each service requests and receives credentials of its peer services and determines its policy decision. Our main contributions are the IBAC model (implemented in the IBAC engine) and the collaboration policy model.

Prior work [2] has introduced context-awareness into the access control models. The context was defined as any information characterizing an object and surrounding environment, such as the time, location or the load over the object. We define a *collaboration context* as an ordered collection of interactions. We first study the interaction types that can occur in collaborations and discuss the security threats due to each type (Section 3.1). Then, we use the interaction types in order to develop a model for the collaboration context (Section 3.2).

IBAC model (Section 3.3) is designed to evaluate a collaboration context against the access requirements. It has a comprehensive approach: it evaluates the entire context to reach an access decision, instead of evaluating each interaction individually. This is to understand the security consequences of combined interactions. Therefore, a deny decision from IBAC engine means denial of the entire proposed collaboration.

Based on our access control model, we develop a new policy model (Section 4), that is designed to express access requirements for joining collaborations. A collaboration policy is prepared even before a service is advertised for collaborations. Therefore, the access requirements are expressed in a generic fashion, such that arbitrary collaboration proposals can be evaluated against a policy. To achieve this, we express the access requirements based on interaction types. At evaluation time, the policy evaluation engine dynamically selects the access requirements that match the proposed collaboration

context and only evaluates those requirements to reach a policy decision. In addition, the selected access requirements are not uniformly applied to all of the peer services; different subjects (i.e. peer services) can be applied to different access requirements based on their interaction types. In order to ease the adoption of our policy model, we implemented it as an enhancement of a widely-adopted access control language, XACML [11]. We implemented components to evaluate and enforce our access control and policy models (Section 5).

2. BACKGROUND & RELATED WORK

2.1. Background and Assumptions

We define a collaboration as a directed acyclic graph, a *collaboration graph*.[1] Each node of the graph indicates a service with one or more operations. An edge indicates the data transferred between two operations. The direction of the edge is same as the direction of the dataflow. When we refer to an interaction, we mean a specific data transfer (i.e. an edge) between two service operations.

A service is the provision of any kind of facility to the public, such as computing power, storage, or remote code invocation. A service is not limited to its domain boundaries; it is exposed over a network, and utilizes Web-Service standards such as WSDL [18] and SOAP [15]. Each service's security policy is private, and is not divulged to other services, or to the collaboration. Each service has credentials (such as X.509 service certificates) that can be evaluated by its peers for authorization and authentication purposes.

We do not limit ourselves to any specific collaboration technology; the collaboration can be executed in any manner. We assume that there is a collaboration engine that is responsible for managing the collaboration, such as defining the choreography (i.e. order of interactions) and selecting suitable services for the interactions. Selected services are bound in the resulting choreography document (e.g., a WS-CDL [19] document -- see Section 5.1.1). We do not assume that the selection process has taken any security aspects into consideration. Thus, services may be drawn from different security domains, and they may or may not trust one another.

The collaboration engine invites selected services to join the collaboration. The proposal message includes a collaboration context, based on which services make their decisions to join the collaboration. We assume that the services trust the collaboration engine in creating accurate collaboration contexts. (An untrusting service can decline the invitation.) For added security, the collaboration engine should sign its messages. It is also expected that services will check the collaboration context against the run time accesses made by their peers, but such checking is beyond the scope of this paper.

2.2. Related Work

Kang's work [7] and the WAS framework [8] both assume a multi-domain security model. A central engine acts as a trusted third party, consults with the services' domains and determines which services can interact with one another. The main drawback of these frameworks is that the central engine requires prior knowledge about the security policies of services. Our work assumes that each service's policy is private and is not divulged to the collaboration enginer, nor to the peer services.

In Koshutanski's work [9], only the service that initiates a collaboration is evaluated by its peer services: the first service is authorized by the second one, and when the second service interacts with a third service, the second one is evaluated by the third service, and so on. Koshutanski assumes that since the first service caused all of the interactions, it is sufficient to evaluate the first service by all other services. Unlike our approach, he omits the interactions among other services.

Shehab [14] introduces the secure access path, which represents the access history of a user. He assumes multiple security domains and cross-domain role-mappings between the domains: a role in one domain can have the privileges of another role in the mapped domain. However, he does not discuss the generation of role-mappings, which requires pre-established trust relationships among the domains and a central agreement over the role-mappings. Each domain is assumed to be aware of which mappings are forbidden or authorized. An access decision involves checking the access path of an access request to see if there are any unauthorized mappings (i.e. interactions).

This work is similar to ours, in that an access path can represent an interaction between two domains. Shehab's work assumes each domain knows which role mappings are allowed. In our model, each service has a separate private policy and autonomously evaluates the collaboration context from its own view – it is not dependent on any centrally agreed mappings. An interaction deemed secure by one service may be deemed insecure by its interacting peer.

[1] The model is not limited to acyclic graphs, but our current implementation does not support cycles.

Toninelli [17] and Shafiq [13] propose a model for collaborative environments; however, a dynamic collaboration consists of a requestor and an object that do not know each other. Toninelli's access control model dynamically changes the access requirements based on the collaboration context. She achieves this by reasoning over the context and policies. Shafiq uses trust negotiation and trust management to map unknown users into the GTRBAC model.

3. AN INTERACTION-BASED ACCESS CONTROL MODEL (IBAC)

A subject, or a requestor, is an entity that requests access. An object is a resource that is being requested. An action is an activity that is to be performed on the object. A collaborative peer is a service. In a collaboration, a service can act both as a subject and as an object.

3.1. Interaction Types

The IBAC model manages access to a service based on the types of interactions that are present in a proposed collaboration. An interaction between two services is classified along two dimensions: the proximity of the services (direct vs. indirect) and the direction of dataflow between them (upstream vs. downstream).

A *direct interaction* occurs when the first service transfers data to the second, without relaying it through other services. For example, seller and shipper in Figure 1 have a direct interaction. Any direct interaction between services is a *bilateral relationship*, even when the dataflow seems to be one-sided. To illustrate this, the seller service presents a shipping request to the shipper service and the shipper determines if it trusts the buyer for invoking initShipment operation. However, there are actually two relationships: (1) the seller determines that it trusts the shipper to send its request, and (2) the shipper determines that it trusts the buyer to process the request.

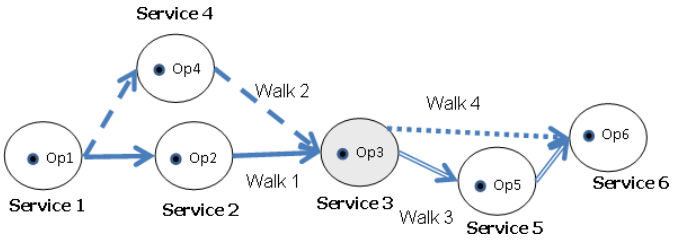
Both seller and shipper access requests involve risk. From the seller's perspective, the shipper could be a rival company with whom the seller is not willing to do business; from the shipper's perspective, the seller could be a malicious user who sends a Trojan horse. Existing access control models such as TrustMaker [3], RBAC [12] or ACL-based schemes, are geared towards assessing the trustworthiness of the requestor. The *reverse* trust evaluation – i.e., the trustworthiness of the requested object from the subject's viewpoint – is not explicitly modeled. Instead, it is assumed that the subject implicitly makes a trust evaluation before launching its request. This implicit modeling does not work in a multi-party collaboration, where services do not have a say in the selection of other services, nor may not have established trust among each other. The collaboration engine selects services to interact with each other; this does not guarantee that services do not possess any security threats to one another. IBAC model, on the other hand, allows the reverse trust evaluation: the seller can evaluate the shipper's trustworthiness by evaluating its downstream interaction, whereas, the shipper can evaluate the seller's trustworthiness by evaluating its upstream interaction.

An *indirect interaction* occurs when data is relayed through one or more intermediate services. The buyer and the shipper services in Figure 1 have an indirect interaction. There are several reasons why indirect interactions must be carefully evaluated. Confidential documents or the results of a sensitive service are typically passed among several peers throughout the collaboration; thus even an indirect neighbor might have access to confidential data, or a modified version or a portion of the confidential data. The original owner and the final recipient of the data are subject to security threats introduced by the intermediate parties that handled and processed the data. An intermediate domain with security breaches may unknowingly expose other domains to these threats. Furthermore, partnership agreements and competition among businesses may prevent them from doing business with certain organizations. Even when such interactions are safe from a security standpoint, the higher-level business logic may forbid them.

We refine direct and indirect interactions with respect to the direction of the dataflow: *upstream and downstream* interactions. A service has an *upstream interaction* with another service when it is the recipient of the data. When a service is the sender of the data, it has a *downstream interaction*. For example, the seller has a direct-upstream interaction with the buyer in Figure 1, and the buyer has a direct-downstream interaction with the seller.

Refining an interaction with respect to its dataflow is important. Although two services participate in the same interaction, their roles, i.e. sender and recipient, and their actions are different. Thus, the security threats introduced to the services are different. The sender, for example, is concerned about revealing its data to the recipient. The recipient is concerned about allowing the data flow into its domain. Informing a service only about the interaction type such as direct or indirect is not sufficient. The service must also be informed about its role in the interaction because, based on its role, a service's access requirements from an interaction may change.

3.2. The Motivation for IBAC Model



Collaboration Request for Service 3:

Interaction Types:	Subjects:	Objects:	Actions:
upstream:indirect, walk 1	Service1/Op1	Service 3/Op3	invoke
upstream:direct, walk 1	Service2/Op2	Service 3/Op3	invoke
upstream:indirect, walk 2	Service1/Op1	Service 3/Op3	invoke
upstream:direct, walk 2	Service4/Op4	Service 3/Op3	invoke
downstream:direct, walk 3	Service5/Op5	Service 3/Op3	consume
downstream:indirect, walk 3	Service6/Op6	Service 3/Op3	consume
downstream:direct, walk 4	Service6/Op6	Service 3/Op3	consume

Figure 3. Sample Collaboration and Collaboration Request.

At the heart of our work is IBAC's ability to express access requirements based on interaction types. A service owner defines the security requirements for her service even before advertising the service for collaborations. Therefore, the requirements are not specific to a given collaboration. The service owner considers the functionalities that her service offers and the collaboration scenarios that her service would likely to participate.

For example, if a service provides loan approval for purchases, it is likely that it will be used in scenarios where it interacts with banks, car dealers, lenders and buyers services. Obviously, the service owner cannot predict each and every collaboration scenario, nor should she have to. Instead, she defines security requirements for possible interactions that may involve her service.

To achieve this, she does a risk vs. threat analysis by answering several questions. For example, is the service sensitive enough to be protected against indirect peers? If so, what security requirements must be requested from such peers? Is the direction of indirect peers affects the requirements – downstream or upstream peers should be subjected to same requirements? Does the exact distance between two peers affect the security requirements – should the policy set different requirements for varying distance, instead of subjecting all indirect peers to the same rules? Should direct peers be subjected to access requirements that are designed for traditional one-to-one interactions or new requirements must be written? If so, what should be the requirements with respect to the direction?

For each interaction type, the owner analyzes the risk vs. threat factors and the resulting requirements constitute the service's collaboration policy. Our work does not include how risk vs. threat analysis must be done for a service. Our work aims to enable a service owner to express its requirements in a policy model and evaluate them. In the next section, we explain how our policy model expresses such requirements.

3.3. Collaboration Context

The collaboration context of a service is the collection of interactions that affects the security of the service. A collaboration context is specific for each service, even in the same collaboration. The context indicates the dataflow from and into the service throughout the collaboration. Formally, the collaboration context of service V is a collection of directed walks W_m such that W_m begins or ends with V. The directed walk W_m can have an upstream or downstream direction.

A collaboration engine generates a context for each service of the collaboration. Since the collaboration engine has a global view of the graph, a context includes both preceding and succeeding interactions affecting a service. This allows the service to make access decisions based not only on the past access history (as in Chinese Wall [4] and Shehab's work [14]), but also on future accesses -- i.e., interactions with downstream services.

3.4. Access Control Model

IBAC model has four entities defined: interaction, subject, object and action. The object represents the service that is protected by the model.

Our access control model has an interaction-based view; each access requirement is stated for an interaction type. Each subject (peer service) is distinguished by its interaction type and evaluated against the access requirements specified for that interaction type. For example, in Figure 3, both Service 1 and 2 request invocation of Service 3; Service 2 has an upstream-direct interaction, whereas Service 1 has an upstream-indirect interaction. As a result, 3 can apply different access requirements to 2 and 1.

Since existing access control models aim to evaluate a single interaction between a subject and an object, a single access request represents a single interaction. This is insufficient for IBAC model; therefore, we define a *collaboration request* in lieu of an access request. A collabora-tion request is generated from the collaboration context and represents mul-tiple interactions. It conveys information about all of the accesses (direct/in-direct, upstream/down-stream) that will be performed by the peer services once the collaboration is executed.

The collaboration re-quest, generated by the IBAC PEP,

includes four pieces of information: the interaction types, subjects, the actions, and the object (see Figure 3). The collaboration request maintains the association among a subject, its interaction type with the object, and the requested action. The interaction type can either include direct/indirect keywords, or it can specify the exact number of edges between two services. There are multiple subjects, actions, interaction types; however, there is only a single object: the requested service and its operations.

There are two actions defined in our model: *invoke* and *consume*. The invoke action is requested by an upstream service (subject) in order to invoke an operation over the requested service, while the consume action is requested by a downstream service (subject) that will access the data out of the protected service.

4. COLLABORATION POLICY MODEL

4.1. Policy Requirements

First, a policy must be able to express access requirements that are designed for specific interaction types. These requirements must be expressed in a generic way to evaluate arbitrary collaborations. Second, a collaboration policy must be easily integrated into an existing access control system. It must coexist with policies that are traditionally used to evaluate one-to-one access requests. A collaboration policy (1) must not disrupt any existing access control system, (2) must be easily augmented to the existing system, and (3) may make use of existing policies whenever desirable. The third requirement promotes policy reuse among the collaboration policies and underlying policies.

A collaboration policy is the smallest unit that manages access decisions for a service. For each service that participates in collaborations, there must be a separate collaboration policy. Within a collaboration policy, an access rule is the smallest building block that states the access requirements sought from a subject that exhibits a specific interaction type.

4.2. Access Rules

An access rule has three elements: Tar-get, Type and Conditions. The Target element consists of the designated inter-action type, subject, object and action entities. In addition to using direct/indirect keywords, the interaction type can indicate the number of edges between a subject and an object. In a rule that specifies an upstream interaction, the action must be set to *invoke*. In a rule that targets downstream interactions, the action is set to *consume*. The object is the service being protected by the policy. Since a service can have multiple operations, different rules of the service's policy can be stated for different operations. Then, the rule target includes the service operation in addition to the service name. (See Figure 4.)

The Type of a rule indicates whether it is evaluated by the local collaboration policy, or an existing underlying policy (external to the collaboration policy.) This allows for rule reuse. (Both types of rule are shown in the policy for Figure 4.)

The Conditions element contains the access requirements of a rule. An access requirement is represented as a predicate, whose evaluated result will be either true or false. A true evaluation is associated with "permit", and false evaluation is associated with "deny".

The result of each rule is combined with respect to a combination algorithm. The result of the combination algorithm constitutes the policy decision. A policy writer can specify a custom-made combination algorithm; using Boolean operators AND and OR.

4.3. Policy Evaluation

After the IBAC PEP generates the collaboration request, the request is matched against all of the collaboration policies stored at the IBAC PDP. The matched policy

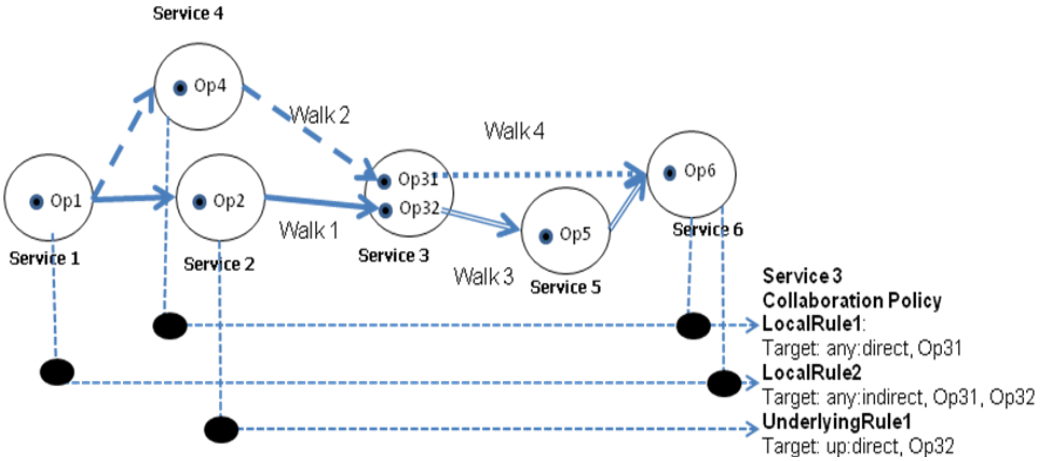


Figure 4. Policy Evaluation.

starts evaluating its rules against the collaboration request. Each rule is matched against a subject, the subject's interaction type with the object, and the requested action. If there is no match, the rule result is "inapplicable". If the rule target matches, the rule evaluates the subject.

In Figure 5, we show a proposed collaboration and Service 3's collaboration policy. The LocalRule1 is specified for the Op31 and targets any peers with an direct interaction type: Service 4 and Service 6 both match and they are separately evaluated against this rule. In order for a rule to return permit decision, all matching subjects must satisfy the rule; thus, both 4 and 5 must meet this rule's requirements. Each rule's result is combined with respect to the policy combining algorithm, which determines the policy decision. A policy decision can either be permit or deny.

5. IMPLEMENTATION

Due to the difficulties involved with developing a new policy language, we enhanced a widely-adop-ted language, XACML [11]. Sun Microsystems' implementation of the XACML framework [16] is used as the foundation for our prototype.

We also implemented the collaboration engine and the IBAC engine that has a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP). A detailed discussion of the implementation can be found elsewhere [1]; here we briefly discuss the collaboration engine, and changes to the XACML language and evaluation framework.

5.1. Collaboration Engine. We implemented the collaboration engine as a standalone web service. The prototype collaboration engine only manages the access control issues within the collaboration; it does not handle service selection and discovery, execution, fault recovery, etc. We anticipate that our code would be incorporated into a fully-fledged collaboration engine that handles the missing aspects.

The collaboration engine has a repository of collaborations, described in WS-CDL (Web Services Choreography Description Language) [19]. We chose WS-CDL because (1) it is an XML-based language, (2) it describes the collaboration as a collection of interactions among multiple parties, and (3) it maintains a global view of the collaboration. A WS-CDL document includes the service bindings, where each binding points to the WSDL of a selected service.
When a service is selected for a collaboration and notified by the collaboration engine, the service's PEP informs the engine about the interaction types required by its collaboration policy. Note that the PEP does not divulge its collaboration policy entirely, only the interaction types required. The collaboration engine creates the collaboration context, identifies and informs the peer services that have interactions with the evaluating service.. Each peer's PEP sends its credential to the authorizing service's PEP. If a peer does not have the requested credential type, the collaboration is denied. The evaluating service's PEP collects the peers' credentials with the collaboration context, and creates the collaboration request.

The collaboration engine repeats the above steps for each service proposed for the collaboration and allows each to evaluate its peers. The results of these evaluations are collected to determine whether the collaboration is allowed.

5.2. XACML Enhancements. Since a collaboration request represents multiple interactions, we implemented it as a collection of XACML requests, each containing an interaction, a subject, an action and a resource (i.e. an object). Each XACML request represents a single interaction from the collaboration context.

We also modified the XACML policy-matching logic. Since a collaboration request has multiple XACML requests embedded inside, we ensured that a selected collaboration policy simultaneously matches all of the embedded XACML requests. Otherwise, the collaboration request could be evaluated against a policy that is not designed for all of the interaction types within the collaboration request. Each rule checks for each XACML request whether it matches the rule and, if so, it evaluates the XACML request. Moreover, we modified the native XACML policy-matching and rule-matching algorithms to consider interaction entity.

XACML rules do not have any types; we enhanced this by introducing two rule types. For *Local* rule type, we did not make any modifications. For *Underlying* rule type, we modified the XACML rule evaluation logic by inserting a software hook. When the hook is executed, it converts the portion of the collaboration request that is being evaluated back into a plain-XACML request format, and sends this to the underlying enforcement agent. (The hook can create requests based on the policy language of the underlying system; current implementation assumes it is native XACML.) The policy decision returned from the underlying system is treated as the result of the Underlying type rule.

6. CONCLUSION

Managing access to a service in collaborative environments is more challenging than in traditional one-to-one settings. The number of peers and the multiplicity of the interactions between the peers complicate the access management.

In this paper, we studied the consequences of peer-to-peer interactions over the access management of a service. We modeled an interaction as dual access request between two peers. This allowed each peer to evaluate the security risks from their own perspective. Moreover, we found that due to the continuous flow of interactions, peers interact with one another directly or indirectly. This made us realize that accesses among the peers occur at different levels. Different interaction types leads to different accesses between the peers. Nevertheless, each interaction type, hence each access, introduces its unique security threats.

This situation motivated us to design an interaction-based access control model that considers the interaction types as an integral element of its decision logic. Different interaction types, hence different accesses, may be applied to different access requirements. In addition, we found that an interaction originally deemed secure may become a security threat when combined with another interaction. Therefore, we developed a model for representing the collaboration context and designed our access control model to evaluate this context comprehensively. Overall, our work aims to increase services' willingness to collaborate by enabling them to address the security issues in a policy-driven and collaboration-agnostic manner.

References

[1] Altunay, M., COLLABORATION POLICIES: ACCESS CONTROL IN SOA-BASED DYNAMIC COLLABORATIONS, PhD Thesis, North Carolina State University, 2007.

[2] Ardagna, C.A., M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "Supporting Location-based Conditions in Access Control Policies," *ACM Symposium on Information, Computer and Communications Security (ASIACCS '06)*, 2006.

[3] Blaze, M., J. Feigenbaum, J. Ionaddis, and A.D. Keromytis, "The Role of Trust Management in Distributed Systems Security," In SECURE INTERNET PROGRAMMING: THE SECURITY ISSUES FOR MOBILE AND DISTRIBUTED OBJECTS, Springer-Verlag, 1999, pp. 185-210.

[4] Brewer, D.F.C., and M.J. Nash, "The Chinese Wall Security Policy," *IEEE Symp. on Security and Privacy*, 1989.

[5] Deelman, E., G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D.S. Katz, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal*, 13(3), 2005, pp. 219-237.

[6] Hull, D., K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P.Li, and T. Oinn, "Taverna: a Tool for Building and Running Workflows of Services," *Nucleic Acids Research*, Vol. 34, 2006.

[7] Kang, M.H., J.S. Park, and Y. Peng, "Access-Control Mechanisms for Inter-Organizational Workflow," *ACM Symp. on Access Control Models and Technologies*, 2001, pp. 66-74.

[8] Kim, S.-H., J. Kim, S.-J. Hong, and S. Kim, "Workflow-based Authorization Service in Grid" *Intl. Workshop on Grid Computing (GRID'03)*, 2003, pp. 94-100.

[9] Koshutanski, H. and F. Massacci, "An Access Control Framework for Business Processes for Web Services," *ACM Workshop on XML Security*, 2003, pp. 15-24.

[10] Murray, G., "Asynchronous JavaScript Technology and XML (AJAX) with the Java Platform," http://java.sun.com/-developer/-technicalArticles/J2EE/AJAX/, Oct. 2006.

[11] Organization for the Advancement of Structured Information Standards (OASIS), "Extensible Access Control Markup Language (XACML)," http://docs.oasis-open.org/-xacml/2.0/-access_control-xacml-2.0-core-spec-os.pdf, Feb. 2005.

[12] Sandhu, R., "Role-Based Access Control Models," *IEEE Computer* , 29(2), 1996, pp. 34-47.

[13] Shafiq, B., E. Bertino, and A. Ghafoor, "Access Control Management in a Distributed Environment Supporting Dynamic Collaboration," *Workshop on Digital Identity Management*, Nov. 2005.

[14] Shehab, M., E. Bertino, and A. Ghafoor, "Secure Collaboration in Mediator-Free Environments," *ACM Conf. on Computer and Communication Security (CCS)*, Nov. 2005.

[15] "Simple Object Access Protocol (SOAP) 1.2," http://www.w3.org/TR/soap12, April 2007.

[16] Sun Microsystems, http://sunxacml.sourceforge.net

[17] Toninelli, A., R. Montanari, L. Kagal, and O. Lassila, "A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments," *Intl. Semantic Web Conference*, 2006.

[18] W3C, "Web Services Description Language (WSDL) 1.1", http://www.w3.org/TR/wsdl, March 2001.

[19] W3C, "Web Services Choreography Description Language (WS-CDL) Version 1.0," http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, Nov. 2005.