xDAuth: A Scalable and Lightweight Framework for Cross Domain Access Control and Delegation

Masoom Alam†, Xinwen Zhang‡, Kamran H Khan†, and Gohar Ali†
†Security Engineering Research Group IMSciences Pakistan
{masoom.alam,kamran,gohar}@imsciences.edu.pk
‡Huawei America Research Center, Santa Clara, CA, USA
xinwen.zhang@huawei.com

ABSTRACT

Cross domain resource sharing and collaborations have become pervasive in today's service oriented organizations. Existing approaches for the realization of cross domain access control are either focused on the model level only without concrete implementation mechanisms, or not general enough to provide a flexible framework for enterprise web applications. In this paper, we present *xDAuth*, a framework for the realization of cross domain access control and delegation with RESTful web service architecture. While focusing on real issues under the context of cross domain access scenarios such as no predefined trust relationship between a service provider domain and service requestor domain, xDAuth leverages existing web technologies to realize desired security requirements while supporting flexible and scalable security policies and privacy protection with low performance overhead. We have implemented xDAuth in a medical module in OpenERP, an open source ERP system. Our evaluation demonstrates that xDAuth is a feasible framework towards general cross domain access control for service oriented architectures.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Security

Keywords

cross domain access control, permission delegation, authentication, authorization, RESTful, web services security

1. INTRODUCTION

The boom of online services has challenged traditional enterprises to open their legacy systems for cross domain access. Consequently, many business processes become more dependent on services provided by others out of their own domains. For example, in a hospital information domain, sharing of patient medical records among other healthcare and insurance service providers is required for many reasons. Similarly, it is a common practice for many organizations to consume services from financial institutions such as audit and financial statement analysis, by sharing their information in online service manner.

The transition in business processes introduce new security challenges. In particular, as autonomous and self-governing administration exists in individual domains, authentication and access control mechanisms need to consider access from external users in efficient and scalable way with least trust to external entities. In addition, delegation has been considered as a critical requirement in cross domain resource sharing and collaborations [18, 13, 29, 17, 23, 28]. Beyond administrative authorizations, a user can delegate partial or complete permissions to others from different domains in a discretionary manner, which can result undesired permission propagation and information leakage.

Cross domain access control and permission delegation have been widely studied in research literatures. Du and Joshi [14] have presented the solution to assign roles to users from different domains and consider issues with role hierarchies. In particular, they have discussed the possibility of exchange of role model among domains. Hasebe et al. [17] have introduced the notion of capability into the RBAC96 for achieving capability-based delegation (CRBAC) in cross domain scenarios. Atluri et al. [8] have presented delegation model and policies in workflow management system. There are also extensive research efforts in general delegation in single control domain context (cf. Section 7). However, these approaches focus on issues of access control and delegation model and policy specifications only. They do not propose concrete enforcement mechanisms, especially practical solutions in service oriented cross domain environment.

Many web-based open standard authentication and authorization protocols have been widely deployed by Internet services. OpenID [24] is a widely used protocol to delegate authentication functions to online identity providers (e.g., Google, Yahoo, MySpace). Single sign-on services such as Microsoft Live ID [6] and Google Accounts API [1] authenticate users for multiple web applications and services. OAuth [4] is a prominent open standard authentication and authorization protocol between web domains, which allows a user to share her private resources stored on one web site to another site without having to share her credentials, typically a

SACMAT'11, June 15–17, 2011, Innsbruck, Austria.

username and password. These solutions, on the other side, focus more on open standard functions and APIs, while provide less support for fine-grained and domain-specific access control and delegation policies. Furthermore, these protocols usually focus on Internet-based services, and it is usually difficult to directly use them in legacy enterprise information systems, where each domain still has autonomous authority on authentications and authorizations.

In this paper, we present *xDAuth*, a general framework for the realization of cross domain access and delegation control. xDAuth leverages a trusted delegation service to serve as a decision making point for cross domain access requests. Each resource sharing (or service provider) domain can publish security policies to the delegation service via open RESTful [15] web service interfaces. Upon an access request [1], the service provider redirects the user client (e.g., a web browser) to the delegation service for authentication and authorization. Instead of authenticating the user itself, the delegation service further redirects the user to an authentication service, e.g., the one in her own domain. The delegation service then obtains the user's attributes upon successful authentication and makes decision if the access request should be allowed, and redirects the user client back to the service provider domain if so. In this way, xDAuth provides strong privacy protection: the delegation service does not learn the authentication credential of a cross domain user, and the service provider can define policies to hide the information of shared resources from the delegation service. In addition, the separation of policy decision point (the delegation service) and policy definition (the service provider) enables very flexible and scalable deployment of the framework. Very importantly, with the unique position of the delegation services for authorization, xDAuth can seamlessly support many access control and delegation constraints in cross domain environment, such as separation of duty (SoD) and the Chinese Wall policy.

There are several design and implementation challenges for xDAuth. First, as the delegation service is a central trust point, efficient decision making is mandatory. Secondly, when proxying the authorization for a service provider domain and the authentication of a service requestor domain, the delegation service should maintain seamless session management between the two redirections. Last but not least, xDAuth should have built-in revocation mechanism, not only for authorization policy revocation from a resource sharing domain, but also for revoking a user with already authorized permissions in an active session.

We have implemented xDAuth in a medical module of OpenERP, an open source ERP system. Our implementation supports a set of flexible access control and delegation policies for a medical information domain, to share medical records to other health care services. Built on top of maturing RESTful web service architecture and protocols, xDAuth provides friendly experience for web users. Our evaluation demonstrates that xDAuth is a feasible and lightweight framework for general cross domain access control and permission delegation in service oriented architectures.

Outline: The rest of the paper is organized as follows. Section 2 presents some motivating use cases towards a flexible but lightweight cross domain access control framework and summarizes our design principles. Section 3 gives an overview of the xDAuth framework and Section 4 presents its design details. Implementation and evaluation of xDAuth in OpenERP are presented in Section 5, followed by discussions on variant extensions in Section 6. Finally, Section 7 presents related work in cross domain access control and delegation, and Section 8 concludes this paper.

[1]We consider cross domain access only in this paper.

2. MOTIVATING USE CASES AND SYSTEM DESIGN PRINCIPLES

We present two motivating use cases where cross domain access control and permission delegation are essential for their security problems. We then identify overall design requirements towards practical solutions.

2.1 Motivating Use Cases

Distributed health care information system In nationwide health information network (NHIN) architecture [22], multiple health care providers form a virtual coalition to share health care data and provide sophisticated and efficient online services. Traditionally, health care data of a hospital is provided to an authorized user (e.g., a physician) in the same domain, which is authenticated with a username and password. In a coalition, external physicians belonging to other hospitals may also need access the electronic health records (EHR) of a patient not registered at their hospitals. A trivial solution is to create a new account for every external physician and assign proper permissions. Obviously, this leads to a situation where the security administrator of the hospital domain has to handle many of external accounts in on-demand way. Alternatively, a primary doctor [17] or a patient [10] in the hospital can explicitly delegate corresponding permissions (access rights on the EHR) to an eternal physician. On one hand, it is risky to enable complete discretionary delegation between users in cross domain scenarios, as a typical user may not have the knowledge or awareness of sensitive permissions which can cause serious information leak or privacy compromising. On the other hand, complete security administrator controlled solution fails on authorization flexibility, e.g., when an external physician needs to access the EHR under emergency situation and while there is no pre-defined delegation policy for the physician by the security administrator.

Inter-organizational workflows Commercial organizations often take consultancy services from financial institutions, such as financial statement analysis and audit. Different employees from a financial institution need to inspect the information of a target domain. Therefore, explicit delegation relationship is usually needed to allow the access. The situation becomes complex when one user has access to multiple organizations financial data. Trivially, each user obtains multiple delegation credentials for accessing data of many client systems, which is obviously not scalable and cost effective on security administration, especially when the set of users to access variant target domains is not fixed. Furthermore, performing both authentication and authorization at the resource sharing domain make it difficult to enforce many dynamic security constraints such as the Chinese Wall policy. Existing approaches use cryptographic protocols for handling delegation. However, management of such mechanisms is a very tedious task such as permission revocation [11, 12]. An adaptable framework is needed where the user can be authenticated in her own domain, and then be au-

thorized for accessing external services based on her authentication attributes.

2.2 System Design Principles

The above two use cases represent many cross domain situations where a general framework is desired for flexible, scalable, and lightweight access control and permission delegation. Among many requirements, we highlight several here which represent the salient features and objectives of our design.

Requirement#1: Flexible constraints and scalable security: The desired solution should support flexible security policies for cross domain environments, especially when an access request is from a user without authentication credential at the resource provider domain. In particularly, the framework should be extensible to support dynamic and scalable application specific constraints such as separation of duty and the Chinese Wall policy among multiple domains.

Requirement#2: Trade-off between scalability/efficiency and trust: As aforementioned, performing both authentication and authorization at resource provider domain introduces the scalability and flexibility issues. However, delegating authentication to external entities is not an option in our design as it significantly increases the trust base. Instead, the framework should leverage existing authentication mechanism of a resource requestor's home domain. On the other side, we can assume some trust for a resource provider domain to a centralized entity for authorization decision deriving purpose, while authorization policies are still defined by the security administrator or individual users in the resource sharing domain. This enhances the scalability of cross domain authorization as there is no need for a resource sharing domain to explicitly trust and link to the authentication mechanisms of all possible resource requestors' domains.

Requirements#3: Preserving privacy for access requestor and resources: An access requestor's credentials are used for authentication and result attributes are used for deriving authorization decisions. Ideally, a resource sharing domain does not need to know the requestor's authentication credentials and attributes in her home domain, except an identity. Furthermore, the authorization decision making entity should not know what real resources and permissions that a requestor is accessing.

3. OVERVIEW OF XDAUTH

In real world, enterprises often delegate the security verification of incoming people (to its premises) to companies which have specialized skill set in doing such job. Based on stated policies of an organization, these security companies verify various credentials of incoming people, before they are allowed to enter the premises of the organization. Limited time permits are often issued, thus, not every person needs a security clearance.

Our approach mimics these human verification systems. As Figure 1 shows, a *service provider* (SP, e.g., an enterprise web application) delegates authentication and authorization tasks for cross domain access to a service called *delegation service* (DS). Instead of performing authentication by itself, the DS further delegates the authentication task to the existing mechanism of the *service requestor* (SR) domain. Therefore, the DS acts as a mediator between the SP domain – the enterprise, and the SR domain – e.g., the user's

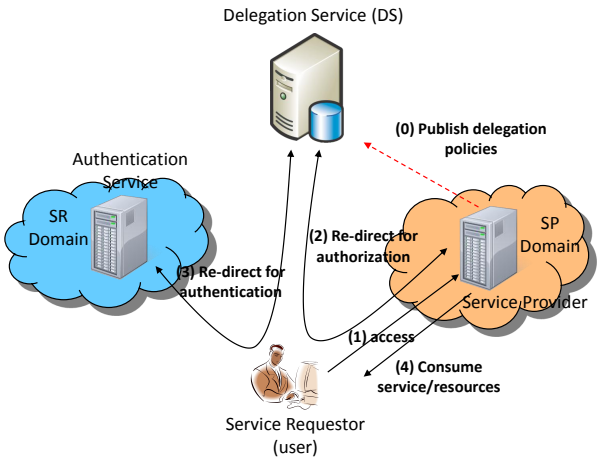

Figure 1: Overview of xDAuth.

home domain. In general, an SP domain can be an SR domain of another, and a single DS can work for multiple SP and SR domains.

xDAuth ensures the essential control of an SP domain with two facts: authorization decisions made by the DS are based on access control and delegation policies from the SP domain, and each authorization is based on authenticated information of the user from her home domain. More specifically, when the SP receives an access request from a user, the user's client (e.g., a browser) is redirected to the DS for authorization decision. After the user is redirected to the DS, a list of domains are presented. The SR selects her home domain (or any other domain that she can be authenticated) from the list and is redirected again to the authentication service interface of the SR domain for authentication. After successful authentication, the user is redirected back to the DS along with her identity and security attributes (e.g., roles and clearance). These attributes are then verified and evaluated by the DS against pre-defined policies by the SP domain. The DS then redirects the user back to the SP along with her identity, domain information, and authorization result, which takes corresponding actions based on the result.

Several benefits can be achieved with this *delegated* authentication and authorization mechanism in cross domain access scenarios. First, an SR domain is made aware of accessing a service by a user to another domain, which enables auditing seamlessly. Also, further security policies can be enforced in the SR domain, e.g., cross domain access to a particular SP is only allowed to certain users only. Therefore, the domain might refuse to authenticate a user after evaluating its own cross domain security policies. Secondly, an SR has no authentication credentials (e.g., username and password) on the DS except her home domain information (e.g., the URL of the authentication service). By handling two different sessions, rather than two different accounts, xDAuth increases the efficiency and flexibility of cross domain authorizations.

From privacy perspective, the stated paradigm has two advantages. Firstly, the privacy of the user is protected as there is no user credentials or attributes provided to the SP domain. All user attributes are verified at the DS end only and the DS does not have the user's authentication creden-

tials. Secondly, the privacy of an SP is also protected as the DS has no knowledge of the resource (at the SP end) and permissions that the SR is asking. Easily, the security policies defined by SP domain can use pseudonyms of resources and permissions without revealing real internal information to the DS when publishing policies.

As a centralized service, the DS has the knowledge of concurrent cross domain accesses from an SR, therefore can enforce many flexible security constraints such as dynamic separation of duty. Optionally, the DS can maintain historic information of the access requests from variant SR domains, which enables it to enforce other general constraints such as the Chinese Wall policy. These constraints are specified by individual SP domains as part of cross domain security policies.

The role of the DS in xDAuth is very similar to the WRYF service in Shibboleth [21]. However, there is significant difference on the design of xDAuth from Shibboleth, which achieves different security objectives. Specifically, in Shibboleth, the WRYF service just maintains a list of home organization access points of users. When a user is redirected from an SP to the WRYF, the user selects her home organization and the WRYF redirects her to the access point. After that, the interactions are purely performed between the SP and SR – SR does authentication and SP does authorization. That is, the WRYF service is simply a redirection proxy and does not obtain authentication results of users and evaluate authorization decisions. In xDAuth, the DS performs authorization evaluation based on user authentication results. With the central position of the DS, many flexible security constraints crossing multiple domains can be enforced such as dynamic separation of duty and the Chinese Wall policy, which are not viable in Shibboleth. At the same time, xDAuth still maintains strong privacy protections to both SPs and individual users.

Threat and Trust Assumptions The main objective of xDAuth is to prevent unauthorized access to protected resources in an SP domain from external users. Therefore, any access request to an SP which is not authorized by the SP or DS is a potential threat.

By delegating the authorization decision to the DS, we assume that the SP trusts the DS to make right decisions based on pre-defined and published policies. This also implies that the SP trusts that DS keeps the integrity of the policies. However, we do not assume that each SP trusts all possible SR domains explicitly. That is, we do not require the web of trust between domains of SPs and SRs; instead, xDAuth leverages the DS as a proxy of trust. After an SR user is authenticated at her parent domain, and successfully authorized at the DS end, a transitive trust relationship is established between the SR and SP through the DS. With this, we eliminate the complexity of trust management between SP and SR domains. For example, an SP does not need to store credentials (e.g. , public key certificate) of each SR domain in order to verify the message authenticity and integrity, while the trust burden is handled by the DS in the middle.

We also trust the SP user's client agent such as web browser. We do not consider attacks on the cryptography used in protecting the integrity and authenticity of messages between SP, DS, and SR.

4. DESIGN OF XDAUTH

This section first gives the bootstrap of xDAuth including policy specification, domain registration, and policy publishing. We then illustrate the authorization and authentication protocols for cross domain access control, cross domain constraint enforcement, and revocation mechanisms.

4.1 xDAuth Policy

In xDAuth, a user from an SR domain is allowed to access resources in an SP domain, if allowed by cross access or delegated policies of the SP. Without loss of generality, we explain how *delegation policy* can be defined and enforced in this section, while cross domain access control policies can be easily supported with similar mechanisms. For cross domain delegation, a user or an administrator in the SP makes a delegation request to an internal authorization service. The delegation request is verified against a set of *delegation control policies* in the SP. Therefore, an xDAuth policy is generated by combining the information contained in the delegation request and that in the delegation control policies.

Formally, a delegation control policy is defined as a set of *rules*, each of which stating the *delegation_status* of individual permissions and constraints. In general, a permission p is defined as a pair (o, A), where o is an object (or resource) and A is a non-empty set of actions. Therefore, a permission essentially identifies possible access actions on an object within a particular domain. A constraint c defines conditions such as the life time of a delegated permission, the *delegatee*'s attributes such as roles or domain names. Therefore, a delegation control policy mandates an explicit approval of the delegation of a permission. The *delegation_status* is a boolean value specifying whether a permission is delegatable or not for a *delegator* (user or role) in the SP.

A delegation request is defined as a triple (s_i, p, s_j) where $s_i \in S$ is a subject playing the role of a delegator, p is a permission, and $s_j \in S$ is a subject playing the role of a delegatee. Each delegation request is evaluated against delegation control policies in the SP domain. If there is a delegation control policy that allows the request, it is approved by the internal authorization service of the SP, and a cross domain delegation policy is generated with the (s_j, p, c), where c is the constraint corresponding to the delegation control policy. Formally:

$xDAuthPolicyGen$: $(DR \otimes P) \rightarrow \{xDAuthP \mid Error\}$, where $xDAuthPolicyGen$ is a mapping from a set of delegation requests DR and set of control policies P to a set of xDAuth policies $xDAuthP$ or an error. $\otimes$ operator matches a particular delegation request against the set of delegation control policies. We note that there can be multiple control policies that can satisfy one request, where multiple authorization polices can be generated.

As an example, consider a delegation request made by a doctor in an hospital for the blood tests of a patient. This requires access to the medical record of the patient. The following delegation query (DLQ) is being made:

$DLQ(Doc001, Lab001, readPatientRecord,$
$designation = ''pathologist'' \ and \ lifetime = 300mins)$,

where `Doc001` requests the delegation of read permission on the patient record to another domain `lab001` with the delegation constraint that designation of a user from `lab001` should be `pathologist`. The delegation control policy in this case, verifies that whether a xDAuth policy exists that

can satisfy the above delegation request. In case, an xDAuth policy exists, the above delegation request will not be approved.

4.2 Domain Registration

Consider an SP domain which provides services via `http://sp.com`, and the DS `http://ds.com`. In order to share resources, an administrator of the SP domain should first register on the DS via a web service interface `http://ds.com/register`. The registration requires information of the SP including service name, service access URL, and other metadata such as the services it offered (e.g. blog, finance solution, social networking, etc). Similar registration is required for an SR domain. In addition, a call back interface (e.g., `http://sr.com/authenticate`) is also provided by the sR domain to the DS, which is used by the DS for redirecting authentication requests.

As the results of an registration, the DS returns a domain key and secret pair. The domain key is a 30-byte public string that is unique to identify the domain, and the secret is a 10-byte shared secret between the DS and the domain. Table 1 lists necessary web interfaces for the DS and individual domains.

4.3 Publishing Policy

Each SP domain has an internal authorization service that approves delegation requests made from local users. When an administrator wants to delegate any permission of accessing resources to others in different domains, we assume the local authorization service provides necessary interfaces and tools to help the user make appropriate selections, such as corresponding objects and set of access actions which she wants to delegate, and applicable constraints such as valid time period. Upon this specification, the authorization service can approve the request based on pre-defined delegation control policies in the domain. In reality, the approval of a delegation request can be done automatically by system, or manually approved by administrators.

After a delegation request is approved, the authorization service creates a real cross domain delegation policy with the delegated permission information and constraints, and publishes it to the DS via a service interface `https://ds.com/policy/publish`. Each policy can be identified with an id by the authorization service such that it can be referred later, e.g., for update or revocation. For privacy purpose, the permission information in the delegation policy can be pseudonyms such that the real information of shared resources is hidden from the DS.

4.4 xDAuth Protocol

Consider a user from an SR domain wants to access a resource of a service provider in an SP domain by accessing `http://sp.com`. Figure 2 shows the work flow of xDAuth protocol to authorize this request. We assume that the user has not been granted for any access before this request at the SP. Therefore, she does not present any delegation permit with her access request. We also assume that the SP does know that the request is from external, e.g., via the client IP address of the HTTP request.

In order to authorize the request, the SP first generates a request token, and then redirects the user's browser to the DS for authentication and authorization. The HTTP redirection request includes the request token and permission

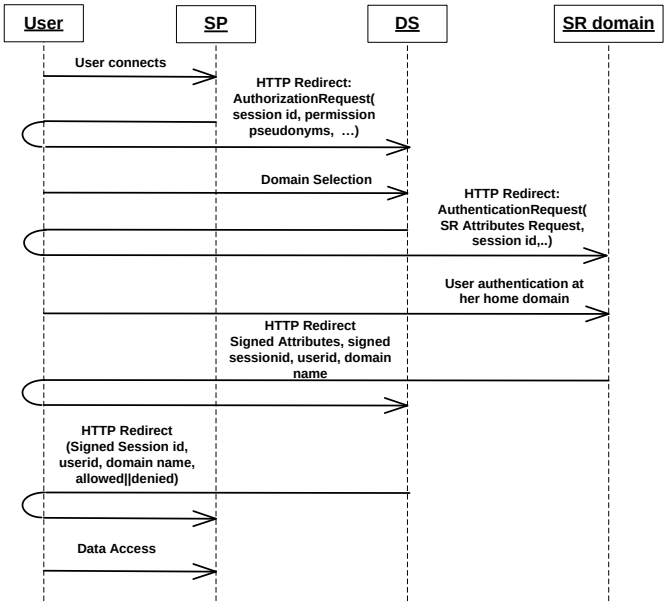

Figure 2: xDAuth protocol.

information required by the user's access, e.g., permission pseudonyms. A unique session id is also used in the HTTP request to prevent replay attacks.

After the user is redirected at the DS, she is asked to select her home domain from a list of registered SR domains. Upon the selection, the user is in turn redirected by the DS to the authentication service interface of SR with the domain name, which is provided to the DS during the SR domain registration process. This HTTP redirection includes the domain name and other information of the SP, as well as the session id. The user then authenticates herself, e.g., by login with username and password. Local security policies of the SR domain may be enforced within the authentication service, e.g., by checking if the user is allowed to access the SP's resources. We note that if the user is already authenticated in the SR domain, the authentication service can directly obtain the results, e.g., by reading the cookie at the user's browser. Overall, the authentication mechanism in the SR domain can be variant but provide single interface to the DS.

Upon successful authentication and authorization in the SR domain, the user is redirected back to the DS along with authentication results, such as roles, identities, or other attributes. The HTTP redirection is signed by the shared secret between the DS and SR, such that the DS can verify the integrity and authenticity of the authentication results. When the DS receives these, it first evaluates the user's access request based on the authentication results and prepublished delegation policies by the SP domain authorization service. Note that as the same session id is used when the user is redirected back from the SR, the DS can link the authentication results to the user who make the original access request to SP. The DS then redirects the user back to the SP, along with the authorization result (allowed or denied), the request token, the session id, and the necessary user identity information if the access is allowed (e.g., for logging and auditing purposes at the SP side).

Table 1: Web Service Interfaces of DS and Individual Domains

URL	Function Description
`http://ds.com/`	Main interface for authorization.
`http://ds.com/register`	Register individual domains, called by domain administrators.
`https://ds.com/policy`	Publish, update, and revoke cross domain access control policies, called by an SP domain authorization service.
`https://ds.com/update`	Update and revoke an authenticated user, called by an SR authentication service.
`https://sr.com/authenticate`	Authenticate a user in SR domain, called by DS.
`https://sp.com/revoke`	Revoke an authenticated user who has been allowed in a cross domain access session, called by DS.

Once receives and verifies the authenticity of the HTTP response, the SP generates an access token based on the request token, and allows the access of the user to required resources. The access token is used in all following transactions of the same session. Moreover, the access token along with different attributes is stored in the browser of the user as a delegation permit.

4.5 Cross Domain Constraint Enforcement

Enforcing cross domain security constraints is one of the major benefits of xDAuth. As the policy decision point (PDP) for cross domain access, the DS has the capability to enforce very flexible constraints. We use the Chinese Wall policy as an example to explain this capability. Consider a simple Chinese Wall policy which states that two SP domains are in conflict of interest such that resources in SP1 should not be accessed by a user who has an active session with SP2 at the same time. With the DS's record of active authorized sessions, it maintains two simple lists of SR domains that have active sessions in SP1 and SP2, respectively. When a new request from an SR domain for authorization of SP1, the DS simply checks if the same SR appears in the active list of SP2. If so then the request is denied due to the Chinese Wall policy. In order to support general Chinese wall policy which is a constraint for multiple sessions, the DS implements a function that records the history of accesses to conflicting domains. Moreover, in the current xDAuth, a single DS is used among a set of federated domains. We note that with Shibboleth and OAuth-like protocols, as there is no record of active sessions, cross domain constraints are difficult to enforce. Similarly, more general and fine-grained constraints such as dynamic separation of duty (DSoD) [27, 20] in user and role levels can be enforced with same mechanism.

4.6 Authorization Revocation

Revocations can happen in xDAuth in two levels: policy revocation, and access token revocation. The revocation of a delegation policy is relatively simple with the help of a dedicated web interface provided by the DS. This service URL is provided to each SP domain as the result of domain registration. When revoking a policy, the authorization service in an SP domain sends the domain name and policy id to the DS. After verifying the authenticity of the revocation request, i.e., with the signatured generated from the shared secret between the DS and SP, the DS removes the policy in its local database.

There can be several scenarios to revoke an active access token issued by the SP to a cross domain access session. First of all, if a user within the SP domain or the SP service wants to revoke an active accessing session, it can directly revoke the access token issued for that session. Immediately, the user cannot access the resource. A more complex situation appears when any of the accessing user's attributes in her home domain is changed, e.g., a role is deactivated or even revoked by the SR domain. In this case, the SR authentication service needs to send the updated user information to the DS, via a dedicated web interfaces `http://ds.com/update`, given by the DS during registration phase. This HTTP request should include previous session id that is obtained from the DS for original authentication request, therefore the DS can re-evaluate the user's permission based on updated information. If the user's access should be revoked, the DS issues an revocation request to the SP via a web interface `http://sp.com/revoke`. The same session id is used so that the SP can make corresponding actions on the active session. The lifetime of the session id and access token are global system parameters, e.g., from a few minutes to hours.

5. IMPLEMENTATION & EVALUATION

In this section, we provide the implementation details of xDAuth framework along with a case study for a medical domain. The underlying information system playing the role of the SP is an open source enterprise resource planning (ERP) system called OpenERP [5]. We then evaluate the performance of xDAuth with our implementation and suggest performance improvement strategies.

5.1 xDAuth for Healthcare

Medical [2] is an open source module in OpenERP that provides a complete electronic medical record (EMR) system with patient information and other medical information. It also contains a hospital information system (HIS) integrated with other OpenERP modules like inventory and financial management.

As OpenERP does not support cross domain access, in current Medical module, each user must have an local account with username/password in order to access medical objects. Therefore, in the current settings, it is not possible for an external laboratory to connect to an OpenERP server and process the medical record of a patient. To enable cross domain access, we have developed a module called `xauth` in OpenERP. The module is derived from the OAuth Python library [4], which consists of a server module called `provider` and a client library `test.py` to provide xDAuth functionalities.

Our implementation consists of three servers: the SP domain running a Medical information system [2], the DS server running a Python web service, and the SR authen-

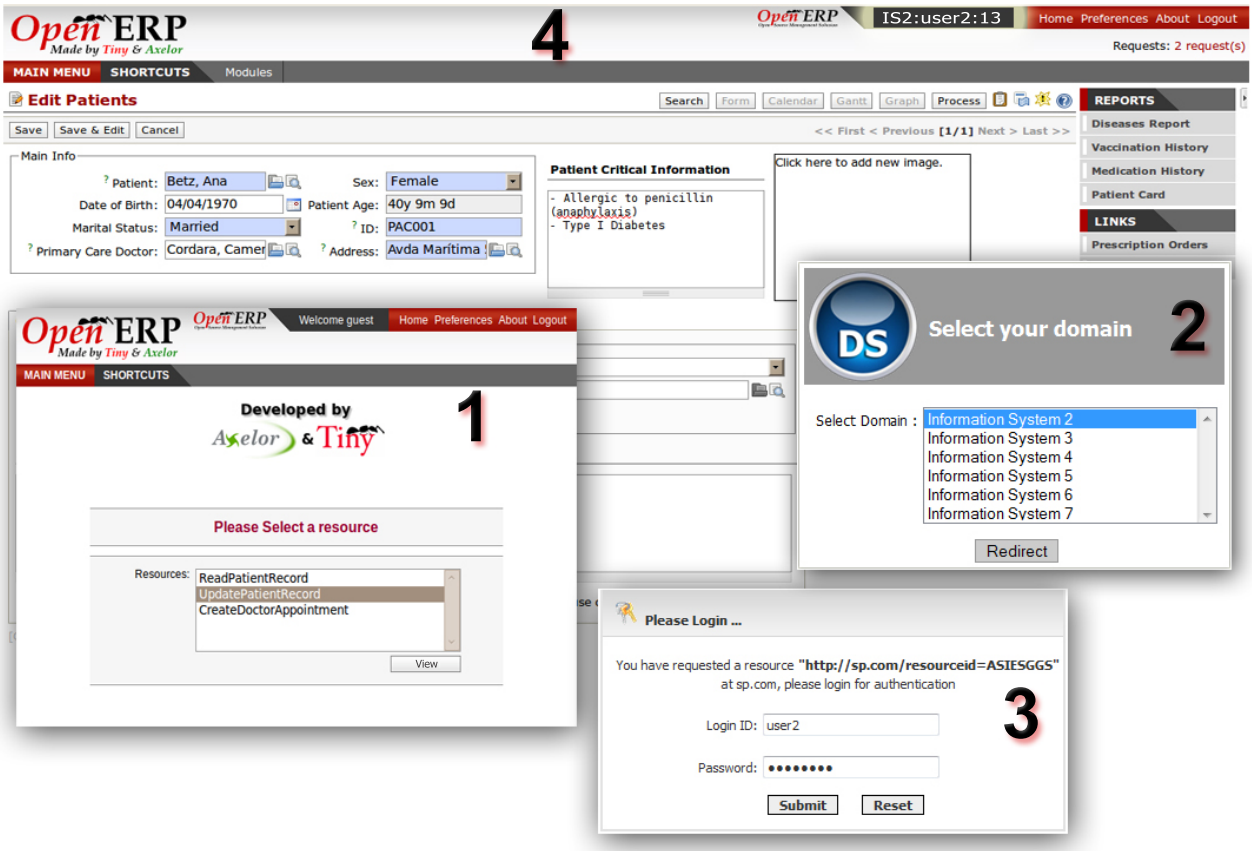

Figure 3: A snapshot of the xDAuth protocol: (1) view list of shared resources at the SP, (2) domain selection at the DS, (3), authentication at the SR home domain, and (4) resource access at the SP.

tication server running a PHP service which provides typical username/password login. The username/password database is extract from the Medical in SP domain to the SR authentication server. Figure 3 shows a snapshot of the xDAuth work flow for an external user to access Medical EMR of a patient. Once the user selects a particular resource and presses the view button in Medical main web page (cf. Fig 3 – Part 1), the client part of the `xauth` is called and redirects the user's browser to the DS service along with the unique link through which the necessary permission is identified at SP. It is important to note that the actual resource information such as `ReadPatientRecord` is not forwarded as part of the redirection for privacy, while only a pseudonym is used, which is the same name used when a delegation policy is created in the Medical domain for reading patient EMR. The user is prompted with a list of registered SR domains (cf. Fig 3 – Part 2). After the user selects her home domain, the browser is redirected to the SR authentication page (cf. Fig 3 – Part 3). After successful authentication, the user is redirected back to the DS, which evaluates the delegation policies defined by the Medical domain, and then redirected back to the original resource web page. If the access is allowed, the user can view the requested patient record (cf. Fig 3 – Part 4).

The reminder of this section gives more details of xDAuth policy in XACML and the implementation of `xauth` module and the DS service, followed by our performance evaluation.

5.2 xDAuth Policy in XACML

As a de-facto standard, XACML [3] is a natural choice for xDAuth policies specification. The reason is that the DS works like a central policy decision point for multiple service providers, therefore XACML can be used to express and evaluate policies from a variety of SPs using a single language.

In our implementation, the `<Subject>` element of an XACML policy identifies the domain of the delegatee subject. A more detailed information about a delegatee can also be included here such as roles or even identities. The `<Resource>` element refers to the resource URL in an SP domain, and the `<Rule>` element specifies the corresponding constraints which the DS must evaluate regarding a request. In our implementation we focus on enforcing cross domain constraints. Specifically, the XACML policy specifies simple mutual conflict of interests among individual SP domains, such that a user cannot have active sessions in both domains in a conflict pair concurrently. Another possible setting could be to include the SR's domain information within the `<Rule>` element of XACML. Therefore, policy encoding in the same way helps the DS in the organization of a set of xDAuth policies from various SPs.

5.3 `xauth` Module

The `xauth` module extends the `base` module in OpenERP with an extra class called `res_delegation`. The `res_delegation` class is responsible for handling delegation requests from the local users and generating xDAuth policies. It further adds an extra attribute `external` in the `res_users` class by using the inheritance mechanism in OpenERP. This attribute is used to distinguish a local user from an external user. In addition to that, within the `res_delegation` class, a Python function is defined called `delegationRequestHandler`, which handles the incoming delegation requests from the local users and creates or update an existing xDAuth policy at the DS.

5.4 Implementation of DS

The DS is a standalone service which implements the server part of the `xauth` module. This library contains all the functions which an SP or SR can call. Besides, two other important functions are implemented. Firstly, as the DS proxies two HTTP redirections, we use the function `xauth_request_token`, the handler of the first redirection at the DS side, to initiate the second redirection with the SR authentication server by calling the `test.py` at the SR service side. Secondly, `xauth` integrates a Python XACML engine [7] to evaluate XACML delegation policies.

Delegation Decision Making As multiple delegation policies might exist that can be applied on a single user, it is the responsibility of the DS to efficiently evaluate an access request. Several strategies can be employed to efficiently evaluate a set of delegation policies applied to an SR user. In OAuth protocol, for each query from a web consumer to a backend service regarding user data (for example, user contact list), a separate function is defined at the backend service. These functions are responsible for providing data of a user to the consumer. We extend this mechanism to include a function that can also answer some logical queries such as whether an SR user's role is or senior to a particular role x (i.e., $role \geq x$) in the SR domain. For example, if the DS has a delegation policy with delegatee's domain is the SR, but with a condition that access is not allowed to an SR user with role lower than x. In this case, the DS can efficiently evaluate the authorization of an SR user without asking all the attributes from the SR domain. Similarly, multiple logical queries can be defined. Negative delegation policies can also be enabled for efficient evaluation, by first evaluating these policies.

Delegation Revocation As the SP trusts the DS on authorization and authentication, logically, it is the responsibility of the DS to intimate the SP regarding any change in the attributes of the user in SR domain during an active session. For this, whenever a change occurs about the user's attributes in her domain, the DS should be notified by the corresponding SR domain, so that this change can be propagated to the SP domain if a delegated permission should be revoked.

In order to handle revocation efficiently, the SR domain updates to the DS regarding those attributes only, for which the DS has asked during the authentication step of the xDAuth protocol. The DS then re-evaluates the delegation policies to decide if the active session should be revoked. For example, suppose the SR domain updates the DS regarding a local revocation of attribute x for the user. According to a delegation policy which specifies a delegation constraint of $x \wedge y$, the delegation permission should be denied, and a revocation signal is sent to the SP's corresponding interface.

5.5 Performance Evaluation

In our testbed, the DS, SP, and SR run on individual Dell Optiplex desktops with Ubuntu 10.4, 2.8GZ Core2Quad, and 4 GB RAM, within 100MB LAN. The overhead of an xDAuth session includes the two HTTP redirections, and policy loading and evaluation times at the DS. Based on our 50 measurements, the average time for the first redirection from the Medical service to the DS takes about 800ms. The second redirection can occur in 560m or 3s, depending on the factor that whether the user is already authenticated in her home domain or not. We note that the times taken by redirections are almost constants in the xDAuth protocol, with slight variant according to network connection status. We do not count the time taken by a user to login her SR domain, which can vary according to different authentication mechanisms.

Table 2: Performance Evaluation of DS

No. of Policies	Load Time	Evaluation Time (w/o cache)	Evaluation Time(with cache)
10	5ms	48ms	38ms
100	15ms	64ms	46ms
1000	29ms	88ms	64ms

On the other hand, the time taken by the DS for xDAuth policy loading and evaluation is heavily dependent on the number of policies. We evaluate this performance overhead with 10, 100, and 1000 delegation policies, respectively. As Table 2 shows, The major overhead at the DS side is taken by policy evaluations. To improve the performance, we implement a cache mechanism in the XACML evaluation engine. As we can see in Table 2, with authorization decision cache support, the performance of the evaluation engine is improved 25% by average. For large number of policies from many different domains, another strategy can be employed to have policy index such that the policy loading time can be reduced by loading only policies for a particular SR domain.

6. DISCUSSION

In order to reduce the attack window, the permit token expiration time can be set to minimum. Therefore, in critical situations, a delegation permit can be made only good for a single session. This means that delegation permits have to be reissued each time the user logs into a cross domain service and have a very limited lifetime. However, delegation permits expiry time must be chosen adequately to balance usability so that users are not prompted for authentication too frequently.

According to our understanding, OpenERP defines permissions at the class level and not at the business object level. Once a user has been granted read/write permissions on a class, he/she can view all the object's data which are instances of this class. This limitation is inherent in our implementation. Therefore, multiple delegatees having permissions on the same object can view each other's data. Tracking the identity of a local user who has made a delegation can be a problem in cases where multiple users have delegated the same set of rights on the same objects. An SP

creates a temporary identity of an SR by augmenting the domain information taken from the DS and identity of the local user who has enabled this delegation with each other.

In the cross domain access scenarios, one of the major concerns is the privacy protection of a user. An SP usually needs the identity of a user for tracking changes in its system. On the other side, disclosing the identify of a user to an SP might not be desirable due to several reasons. For example, financial monitoring departments often perform inspections of financial institutions. In this case, disclosing the identity of a particular inspector is prohibited, although inspections rights are delegated. This feature is can be easily supported by xDAuth by not forwarding the user identity from the DS to the SP, once the cross domain access is authorized.

Multi-step delegation has been discussed extensively in many delegation models [9, 29, 13]. In cross domain context, multi-step delegation has two alternatives: a user can delegate her authorized permission of an active session to a local user (i.e., in the same SR domain), or to another user in a different domain. xDAuth can be extended to support both cases. For the first one, the local authorization service in SR domain can approve the delegation request, and file the request to the DS, which can evaluate the permission based on delegation policies from the SP domain. Once allowed, the DS can send a request to a dedicated SP interface for this further delegation. The SP can revoke the original active access token and issue a new access token to the new delegatee. For the second case, the original delegatee can file a delegation request to her local authorization service (in SR domain), and if allowed, the authorization service can send a *re-delegation policy* to the DS. Users from the third SR domain can use the permission with the same xDAuth protocol. However, we argue that multi-step delegation in xDAuth in general is a complex process including more than two nested HTTP redirections. Certificate based delegation mechanisms [11, 12] may be integrated with xDAuth for practical multi-step delegations, which is the topic of our future research direction.

7. RELATED WORK

Delegation in Role-based and Workflow Management Systems Among those approaches, RBDM0 [9] was the first approach that aims at modeling user-to-user delegation in the RBAC model. RDM2000 [29] proposes hierarchical roles and multi step delegation. Delegation Logic (DL) [19] and PBDM [30] limits the delegation scope to individual permissions and roles. Crampton et al. [13] presented a formal model for administrative scope of delegations in the context of workflow systems. The main difference between these existing theoretical approaches and that we are concerned is the implementation issues with the enforcement of cross domain delegations using existing web standards.

Cross Domain Access Control and Delegation Crampton et al. [14] presents an approach where a restrictive role hierarchy mechanism is used for external users. Whenever an SR user requests a set of permissions, an authentication service verifies which role can satisfy the requested permissions and uses an assistant matrix to determine the minimum of roles that can satisfy the requested permissions. The authentication service in their architecture ensures that an external user cannot activate inherited permissions in the SP domain. Their architecture can be supported by xDAuth using an authentication service at the SP side. CRBAC [17] provides the formal details of the integration of capability based access control in to RBAC96. The concept of capability is exclusively used to represent permissions that can be delegated in cross domain access scenarios in order to reduce administration cost. Therefore, any user having a special permission `create` can create a capability and assign it to an external user. xDAuth provides a classical mechanism for capability delegation described in CRBAC. In addition, an internal authorization service in SP governs the delegation of permissions to external domains. Shafiq et al. [26] proposed a framework for integrating access control policies of heterogeneous and autonomous domains to form global policy. This integration of access control policies might cause some conflict which are removed in a way that semantics of the global policy are preserved without changing the the autonomy of each individual domain. Our focus is this paper is on the enforcement of a delegation authority that can take decisions on the behalf of various domains. Community authorization service (CAS) is an authorization mechanism between virtual groups in Grid computing [23]. The main difference between xDAuth and CAS is that in CAS, a user first requests a capability from the CAS service before initializing the access at a resource provide side, while in xDAuth, we use web redirections to obtain more friendly user experience.

Web-based Authorization and Delegation OAuth is a de-fato protocol for authentication of web application regarding the data of individual users OAuth. Our inspiration for the OAuth protocol is due to token exchange mechanism for authentication. However in the original OAuth protocol, both authentication and authorization of users are performed at a backend server side, therefore it does not fit the design goal of xDAuth. Permitme [16] has introduced the concept of a delegation authority for providing delegation permits to mashup web applications, for accessing user data at back end services. Regarding the introduction of a delegation authority, Permitme is similar to our approach. However there are some fundamental differences as Permitme assumes that the delegation authority and backend services share information regarding user data. This means that whenever there is a change in the user data, the backend service has to intimate the delegation authority regarding the change. In xDAuth a user has no account on the delegation service, then authentication takes place in her home domain only. Shibboleth [21] is the closest technology to xDAuth. As we have discussed in Section 3, there is significant difference on the design of xDAuth from Shibboleth, especially on the role of the DS, which makes xDAuth ideal for cross domain constraint enforcement. DAuth [25] is an extension of OAuth to split an access token into multiple sub-tokens and assign them to different components of a distributed web consumer. Therefore DAuth can support very fine-grained permission control for accessing user data in service providers. However, like OAuth, DAuth does not support cross domain access and delegation as xDAuth does.

8. CONCLUSION

In this paper, we have presented a cross domain access control and permission delegation framework called xDAuth for service oriented organizations. xDAuth leverages a trusted delegation service to serve as a decision making point for

cross domain access requests. Each resource sharing domain can publish security policies to the delegation service via open RESTful web service interfaces. Delegation in xDAuth occurs at two places: A local user or an administrator delegates rights on her owned resources to other domain users, provided that the delegation at this level is allowed by a delegation control policy of her domain. Secondly, each domain delegates the evaluation of cross domain authorization policies called xDAuth policies to a central policy decision point called delegation service. We implement xDAuth framework within a medical module in OpenERP, an open source ERP system. Currently, we are extending xDAuth framework for multi-step delegation and working on providing it as a open source module in the OpenERP project repository.

9. REFERENCES

[1] Authentication and authorization for google apis, `http://code.google.com/apis/accounts/docs/AuthForWebApps.html`.

[2] Medical–The Universal Hospital and Health Information System. `medical.sourceforge.net/`.

[3] OASIS, journal=OASIS: www.oasis-open.org/committees/xacml/repository/csxacml-specification-1.1.pdf, 2003.

[4] OAuth official web site. `http://www.oauth.net/code`.

[5] OpenERP systems. `http://www.openerp.com/`.

[6] Windows live id, `https://accountservices.passport.net/ppnetworkhome.srf?lc=1033&mkt=EN-US`.

[7] XACML 2.0 implementation in Python. `http://pypi.python.org/pypi/ndg-xacml/0.4.0`.

[8] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, 2005.

[9] E. Barka and R. Sandhu. A role-based delegation model and some extensions. In *Proceedings of National Information Systems Security Conference*, 2000.

[10] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009.

[11] M. Blaze, J. Feigenbaum, and A. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Security Protocols*, pages 625–625. Springer, 1999.

[12] D. Clarke, J.E. Elienb, C. Ellison, M. Fredette, A. Morcos, and R.L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[13] J. Crampton and H. Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2):123–136, 2008.

[14] S. Du and J.B.D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, 2006.

[15] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, (2), 2002.

[16] R. Hasan, M. Winslett, R. Conlan, B. Slesinsky, and N. Ramani. Please permit me: Stateless delegated authorization in mashups. In *Computer Security Applications Conference*, pages 173–182, 2008.

[17] K. Hasebe, M. Mabuchi, and A. Matsushita. Capability-based delegation model in RBAC. In *Proceeding of the 15th ACM symposium on Access control models and technologies*, 2010.

[18] J.B.D. Joshi and E. Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, 2006.

[19] N. Li, B.N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, 2003.

[20] N. Li, M.V. Tripunitara, and Z. Bizri. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information and System Security (TISSEC)*, 10(2), 2007.

[21] RL Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated Security: The Shibboleth Approach. *Educause Quarterly*, 27(4):6, 2004.

[22] T.H. Payne, D.E. Detmer, J.C. Wyatt, and I.E. Buchan. National-scale clinical information exchange in the United Kingdom: lessons for the United States. *Journal of the American Medical Informatics Association*, 18(1):91, 2011.

[23] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, 2002.

[24] D. Recordon and D. Reed. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, 2006.

[25] J. Schiffman, X. Zhang, and S. Gibbs. DAuth: Fine-grained Authorization Delegation for Distributed Web Application Consumers. In *Proc. of IEEE Symposium on Policies for Distributed Systems and Networks*, 2010.

[26] B. Shafiq, J.B.D. Joshi, E. Bertino, and A. Ghafoor. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE transactions on knowledge and data engineering*, pages 1557–1577, 2005.

[27] R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. In *Proc. of Computer Security Foundations Workshop*, 2002.

[28] J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems*, 32(3):365–384, 2007.

[29] L. Zhang, G.J. Ahn, and B.T. Chu. A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):404–441, 2003.

[30] X. Zhang, S. Oh, and R. Sandhu. PBDM: a flexible delegation model in RBAC. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, 2003.