Managing Enterprise Authentication and Authorization Permissions in Digital Ecosystem

Ilung Pranata, Member, IEEE, and Geoff Skinner, Member, IEEE

Faculty of Science and Information Technology, University of Newcastle, Callaghan, NSW, Australia, e-mail: ilungpranata@studentmail.newcastle.edu.au, Geoff.Skinner@newcastle.edu.au

Abstract—Intense interactions between resource providers and resource consumers in collaborating and sharing their resources frequently occur in an open and dynamic Digital Ecosystem environment. Such interactions contribute to several challenges in cybersecurity, particularly in protecting enterprise resources from various malicious attacks. The main challenge that occurs in protecting these resources is the ability to manage multiple unique authorization permissions over the enterprise resources. Unfortunately, the inadequacies of the current security mechanisms in addressing these challenges result in a slower implementation of a potentially highly beneficial Digital Ecosystem environment. Therefore, this paper proposes a distributed mechanism for individual enterprise to manage its own authorization process and resource access permissions, with the aim of providing rigorous protection of the enterprise resources in question.

Index Terms—authorization, authentication, access permissions, distributed resource protection, capability.

I. INTRODUCTION

Digital Ecosystem (DE) is a new promising concept which promotes a dynamic interaction and evolution mechanism amongst its member entities. This concept tremendously utilizes an open framework infrastructure to encourage a wide use of information in solving the digital divide issues of ICT adoption [1]. Further, DE also promotes a dynamic formation of knowledge based economy [2], an emerging economic model that would greatly contribute to the world economic growth by a creation of more jobs and a greater social inclusion [3]. The main attractiveness of a DE environment which generates interests from various established industries and academia is its support for an open, flexible and loosely coupled resource sharing and collaboration across multiple domains. These unique resource sharing characteristics are critical to preserve and improve the synergy among components and enterprises inside a DE environment [4].

Security infrastructure has portrayed a key role in the realization of a dynamic resource sharing environment. A strong and flexible security mechanism becomes a vital necessity to secure the communication and to provide mutual authentication and authorization services inside a DE environment. Fundamentally, the main functionality of DE security infrastructure is to protect the enterprise’s resources from any unauthorized access, and provides a solution for authenticating the resource consumers on every access request. Furthermore, a scalable security mechanism must also have a capability to comprehend the resource access permissions of each resource consumer. However, the adoption of security mechanism in providing these functionalities in the present DE implementation is woefully inadequate. There is still a huge necessity for a robust security infrastructure in delivering a rigorous enterprise authentication and authorization permissions, with the aim of safeguarding the resources from any unauthorized DE member entities.

Several challenges faced by the enterprises in identifying the authorized users and their individual permissions during the resource access. The first challenge is the dynamic situation where multiple resources such as the services, knowledge and information are being shared openly by each enterprise, while the second challenge is the ability of each resource being accessed by various resource consumers simultaneously. Therefore, in this paper, we attempt to deliver a comprehensive framework allowing an enterprise to manage its own resource access permissions with the aim to address the various security challenges occurred in a DE environment.

The reminder of the paper is structured in the following manner: section 2 lists all key definitions used inside this paper, section 3 discusses the background and related work, section 4 presents our detailed proposal in addressing the security challenges, section 5 explains our proposal implementation and finally section 6 provides a brief conclusion of this paper.

II. DEFINITION

Before entering into a detailed explanation of the proposed framework, a list of key terminologies used throughout this paper is presented below:

1. Resources: any services, knowledge, or information which is published, shared, or registered by an enterprise in a DE environment.
2. Resource provider: any entity or enterprise who provide resources in a DE environment.
3. Client: any user or enterprise who request for a specific resource provided by a resource provider.
4. Client profile: the identity of a client which provides the information about the client and the purpose of requesting a specific resource.
5. Permission: a special authorization rule which govern how a resource is being accessed by the client.
6. Capability: a mechanism that contains resource access permissions which is entitled to each client profile.

III. BACKGROUND

Providing a comprehensive and prevailing security infrastructure in a dynamic and open sharing environment such as DE is a trivial task for the enterprises. This is due to a huge number of enterprises who actively publish and consume the resources, and also the capability of a DE environment to evolve over the time. A few security mechanisms such as SOS (Self-Organizing Security) framework [5] provide a way to secure and optimize the
resource sharing and further promote the collaboration in a self-organizing environment.

In a DE environment, most resource providers are still facing a difficulty in assigning the correct authorization permissions to each client. The failure in assigning these authorization permissions and in identifying the authenticated clients could result the resources being compromised by the unauthorized parties. Therefore, a strong and flexible security mechanism must be enforced in addressing these failures. The current security framework in providing clients’ authentication and managing their identities in a DE environment is realized by the implementation of distributed identity management framework [6]. This framework adopts a trusted central Identity Provider (IdP) for authenticating and storing clients’ identities. The authentication is performed by transforming the client certificate token into an acceptable authentication assertions, utilizing the OASIS SAML [7] standard. Furthermore, the identity federation is enforced by Liberty Alliance [8] technology.

The other similar mechanism is Credential Server (CRES) repository framework [9] where client credentials can be stored by both local and remote servers, resembling the Grid Security Infrastructure (GSI) MyProxy Credential [10]. Generally, the authentication process of these mechanisms solely takes place in a credential provider/server where the certificate token is produced for the authenticated client, and a trust relationship is built between the resource provider and the credential provider/server. Several authorization frameworks from collaborative environments (CE’s), Community Authorization Service (CAS) [11] and Akenti [12] could be employed in a DE environment. This is due to the fact that both collaborative environments and Digital Ecosystems have a similarity in function and purpose: to encourage a collaboration and interaction between their components [2].

Although the mechanisms discussed previously work practically well, there are some apparent issues that may limit the ability of resource providers in providing a strong protection for their resources. Firstly, the current mechanisms store all identities and information about the clients in a central server where the authentication or authorization process is performed. This server is prone to a single failure which would greatly affect the whole environment. In a case that the server fails to perform the clients’ authentication or authorization process, no other mechanisms are in place to provide the process. This is in turn can contribute to having all environment resources becoming prone to malicious attacks. Although, it is possible to replicate the central server, the replication process will bring increase administrative issues and in addition to higher chances for compromising the resources considering the huge amount of data to be replicated.

Secondly, a centralized authorization server faces various challenges when it comes to assigning the access permissions to each DE client. This is due to the fact that a huge amount of resource providers are actively involved in a DE environment, and each resource provider is able to host and publish more than one resource. This condition becomes even more challenging as a single resource could be associated with multiple different access permissions, and each client may have different access permissions assigned to him. Therefore, in order to correctly assign the access permissions to each client, the authorization server must be able to answer at least the following questions: On which resource provider is the requested resource located?; What access permissions are defined for this resource?; Is the client allowed to access this resource?; and What access permissions are associated to this client for the requested resource?

Furthermore, as a DE environment grows in size and diversity, more and more clients and resource providers participate and interact in the environment due to the potential benefits that they can achieve. A central server will be increasingly experiencing serious administrative issues in trying to manage all client and resource providers’ accounts and permissions even with the possible use of super computers or grid collections of computers. Due to the identified issues on the current mechanisms from the literature, we argue that the resource protection mechanism in a DE environment must be a distributed process rather than a centralized process. The distributed process is implemented by allowing each resource provider to perform its own clients’ authorization and authentication process, and further facilitating the resource provider to assign the resource right permissions to its clients. To fulfill these requirements, we propose a Distributed Resource Protection Mechanism (DRPM). DPRM could effectively perform the authorization process and manage clients’ permissions in a DE environment. In the following section, we present and discuss a detailed explanation of our propose DRPM framework.

IV. DISTRIBUTED RESOURCE PROTECTION MECHANISM

Inside the DRPM framework, the concept of client profiles is used to identify and keep clients’ information. A new client who makes a request for a specific resource is required to provide their information which will then be stored in a client profile. Once a client profile is created, the resource provider will grant the resource access permissions to the client based on a list of permissions defined inside a capability [13]. This capability will always be used every time a client request for the same resource occurs.

A. Client Identification

In the present mechanism for service discovery inside a DE environment, a client searches for resources by utilizing a semantic discovery portal through their browsers or rich applications [14]. The discovery portal lists all client intended resources which are provided by DE resource providers. Once the client finds the resource that they want, they then contact the resource provider and requests for the resource. At this stage, the resource provider does not know any information about the client and their intended purpose on the resource. This may put the resource at risk as the resource provided by a resource provider may contain highly sensitive information which must be protected from any misuse and malicious act. Therefore, it is crucial for a resource provider to understand a requesting client’s information before any access to their resources is granted.

Taking this into consideration, we adopted a method of creation for a client profile that aims to capture all required, but voluntarily provided, information about a client. The
information which is contained in a client profile provides necessary data about who the client is and about their intentions and purpose for using the requested resources. The aim of implementing a client profile is to ensure the resource provider that resources are not going to the wrong entities and further impose the confidentiality and integrity of the resources. The use of client profiles also facilitates the auditing process of the clients accessing a resource. For example, there may be a situation where a resource provider needs to make a trace back to determine which client was delegated access to the resource in case there was an incident involving a dispute or counterfeiting of the resource in question.

In order to fully implement the client profile, it is necessary that a client registration portal is employed in our framework. A client profile is generated through this registration portal. It is built on the HTML language so that it can be universally accessed through a clients’ browser. Further, resource providers are able to customize the registration portal to contain only the information which is important to them. New clients wishing to access a specific resource are initially redirected into this portal. If they wish to access the resource, they must continue to fill in all the necessary information required by the resource provider to produce a client profile. Once it is produced, the client profile is stored in the resource provider repository and can be used for authenticating the client. On future access requests on the same resource, the client will provide their credential and it will be mapped to their client profile. If it is matched, the client will be granted another access to the resources. Utilisation of this functional procedure and process provides an additional and enhanced method for determining who is accessing a particular resource at a particular time within a DE.

B. Enforcing Access Permission

It is a challenge to enforce client access permissions on resources available within a DE environment. As discussed previously, a DE environment comprises various clients, and these clients could make the same request for a particular resource either at the same or at different time. Through our own investigations and as the literature indicates it is apparent that resource providers who publishes DE resources are having difficulties in managing the resource access permissions of the multiple and diverse range of DE clients. A further complication to the problem is that a resource provider is able to have multiple resources registered and published in a DE environment. This situation creates a complex and hard to define set of resource access permissions for each client. Therefore, a strong mechanism for managing the access permission of each client and their intended requested resource is crucial. This access permission mechanism must provide the ability for a resource provider to know which clients have the permission to access which resources.

To solve this issue, we utilize and evolve the concept of capability introduced by the CAS server used in a Collaborative Environment. In CAS, capability is used to store all access rights of a user which are determined by a community policy. However, the implementation of the capability in our framework is slightly different to the capability implementation in a CAS server. In our framework, capability contains all the necessary right permissions for each client to perform a set of operations on a particular resource. This capability is produced by the resource provider on which a particular resource is hosted. This capability is used by the resource provider to grant the client access to the resources and further provide the authorization process for the clients. After a new client profile is created, the resource provider generates a capability for this client profile that defines all resource access granted to the client. The capability limits the resource access of a particular client by listing all permissions which are granted to the client. Further, this capability is sent to the requesting client and will be used for future requests on the same resource. Every time a client makes a request to the resource provider for the same resource, the client sends back its initial configured capability to the resource provider. The resource provider then authenticates the provided client’s capability and grants the access permissions based on the permissions stored inside the client’s capability. Permissions and policies languages in a capability are expressed by using SAML Authorization Assertions [15]. Further, Public Key Infrastructure (PKI) [16] can be implemented in order to protect the capability by using the resource provider private key to sign the capability.

C. Registration and Authentication Process

On every authentication process, a client is required to provide his own certificate or credential to the resource provider. Resource provider then retrieves client information from the certificate and maps this information with their own client profile. A certificate or credential can be obtained by a client in two possible methods: 1) a certificate issued by external party or 2) a certificate issued by a proxy credential server. The clients may have already implemented their own certificates issued by their Trusted Third Party (TTP) or Certificate Authority (CA), such as VeriSign [17], TrustAlert [18], etc. This certificate is required to be used for each and every online transaction.

Additionally, a proxy credential server [19] could also be implemented in a DE environment to provide the certification services for the clients. In order to cope with those methods, the certificates issued by either external parties or a proxy certificate can be accepted as long as a trust relationship is built between the certificate issuer and the resource provider. A client is responsible for requesting a X.509 certificate [20] from his own CA or X.509 proxy certificate [21] from the proxy credential server. Further, the client must send this certificate to the resource provider for authentication on each time they make a request for a specific resource.

D. The Workflow Process

Having discussed all major components of our DRPM framework, we present two workflow process scenarios for when a resource is being requested. The first scenario shows a request process for those new clients who have never requested and consumed any resources provided by a resource provider. The second scenario presents the request process for those clients who have previously requested or consumed resources from a resource provider.
**Scenario 1: A new client request for a specific resource**
The following figure shows the workflow process model for a new client who has never requested or consumed any resources before.

1. A new client attempts to access a specific resource by making a request to the resource provider where the resource is hosted.
2. Resource provider asks for client certificate and resource capability token. Once client send the certificate, the resource provider will match the certificate with the client profiles stored in its repository. If the client profile is not found, the client will then be redirected to a registration portal. On this portal, the client is required to fill in his information.
3. Registration portal sends back the client information to the resource provider. Resource provider generates a client profile based on the information provided by the portal. Further, resource provider assigns the resource access permissions and policies of a requesting client into a capability token.
4. Resource provider stores the client profile and its capability inside its repository for future resource request. Further, the capability token is signed by the resource provider by using its private key and it is sent back to the client.
5. The client is granted the access to the requested resource based on the newly created capability token.

**Scenario 2: A registered client request for a specific resource**
The following figure shows the workflow process model for a client who has ever requested and consumed any resource provided by a resource provider.

1. A client who makes a request for a specific resource will retrieve his certificate from his own Certificate Authority (CA) or a proxy credential server who resides in DE environment.
2. CA or proxy credential server authenticates the client and sends his encrypted certificate back. Then, Client looks at his repository for the associated capability token of the requested resource.
3. Client will then send both certificate and capability token to the resource provider. Thereafter, the following steps are occurred:
   3.1 The resource provider verifies and authenticates the client certificate and further matches client information with the client profile stored in its repository. If the certificate information does not match with the client profile, client will not be allowed to access the resource.
   3.2 The resource provider decrypts and authenticates the capability token sent by the client.
   3.3 Resource provider will then authorizes the client and grants the access control to the resource based on the permissions defined in client’s capability token.
4. In a case that a client does not have the capability token for the resource as he has not previously consume this resource, client will only send his certificate to the resource provider. Thereafter, the following steps are occurred:
   4.1 The resource provider verifies and authenticates the client certificate and further matches client information with the client profile stored in its repository. If the certificate information does not match with the client profile, client will not be allowed to access the resource.
   4.2 The resource provider then assigns all resource access permissions and policies of the client on the requested resource. These access permissions are stored in a capability token. Then, this capability token will be signed using resource provider’s private key and sent to the client.
   4.3 The resource provider then authorizes the client and grants the access control to the resource based on the access permissions defined on client’s capability token.

**E. Granting Access to Different Entities**
As discussed previously, once a capability token is created by a resource provider, this token is sent to the requesting client. Further, this capability token is stored inside the client repository and is presented by the client on future requests for the same resource. This mechanism may be dubious as a question may arise on why there is a need to send the capability token back to the client while this capability token can be stored in the resource provider repository and used when the same client make the request on the same resource. Therefore, we attempt to provide a further explanation and discussion in addressing this question.

In a DE environment, there is a strong possibility that a client, which in this case could be an enterprise, is able to allow a number of its employees to request and consume the published resources using its client’s profile. These
employees are trusted by their enterprise, and they are representing the enterprise in doing online business negotiations and transactions. Although multiple employees use the same client profile, different access restrictions on the resources must be set accordingly. This is due to the fact that the enterprise as the owner of client profile has different policies specifying who should able to access which resources. Therefore, a number of cases show that an enterprise may want to grant different resource access to different employees based on its own policies. For example, enterprise allows employee A to use its client profile for accessing resources 1 and 2 on resource provider I however, enterprise only allows employee B to use its client profile for accessing resources 2 on resource provider I.

By sending the capability token to the client, it allows the client to allocate specific resource access to each employee. A client can further decide by itself which employee is able to access which resources by allocating the right capability tokens to the right employees instead of letting all employees access all resources using its generic client profile. The workflow process for the employee to access the specific resource using his enterprise client profile is detailed as follows: once an enterprise has granted and allocated the capability tokens to their employees respectively, an employee who wants to access the specific resource from the resource provider must ask for the capability token from his enterprise and send this token together with the enterprise certificate to the resource provider. Then the resource provider grants the resource access for the employee based on the received capability token.

V. CAPABILITY TOKEN IMPLEMENTATION

We have successfully implemented the capability token inside our DRPM framework. We adopted Java programming language to develop the capability token. The implementation of this capability token is using Metro web service stack [22] which is built on top of GlassFish web container [23].

The following sections describe our implementation design of the DRPM capability token, and further explain the way of exchanging the capability token between a client and a resource provider.

A. The Capability Token Design

A capability token describes all access permissions and policies on a specific resource given to a client. On the initial stage, a resource provider makes decisions on a set of authorization permissions allowed for a specific client, and embed these authorization permissions inside a capability token before it is sent back to the client.

We design a capability token to include all necessary major elements which are presented below:

- **Access permissions**: a set of authorization rights which are defined by a resource provider on a specific resources.
- **Client profile identifier**: an ID which describes and identifies a specific client profile.
- **Resource provider identifier**: an ID which describes and identifies a resource provider on which the resources are hosted.
- **Resource identifier**: an ID which describes and identifies a specific resource where the access permissions are assigned to.

![Fig. 4 The capability token design](Image)

Access permissions are the core element of a capability token and it must be expressed in a form of standard policy language that is understandable by both resource provider and client. In order to express the access permissions inside a capability token, we adopted an OASIS standard SAML language [7] which allows the interpretation of authentication and authorization policies for web service platform.

B. Exchanging Capability Token

All communication between resource provider and client is realized via Simple Object Access Protocol (SOAP) over HTTP binding request and response message. SOAP specifies the condition of information exchange inside the web service architecture. It heavily relies on XML for formatting the message and HTTP for transmitting the message. SOAP is an extensible lightweight protocol which is not bound to a specific programming language [24]. Therefore, both resource provider and client are able to interoperate each other in exchanging the request and response messages regardless of any web service frameworks that they use.

Due to great advantages of a SOAP envelope, we embed our capability token inside a SOAP-HTTP message. The SAML defined capability token is bound into a SOAP envelope. A SOAP envelope is divided into two main parts which are SOAP header and SOAP body. SOAP header may contain optional information about the message such as digital signatures or security requirements, while the SOAP body contains the actual resources. For binding purposes, the SAML SOAP binding specification [25] is utilized. SAML SOAP binding specification provides the rules to embed a SAML token into a SOAP header. It is used to bind a capability token which includes the authentication, authorization and attributes statements into a SOAP header.

Further, the resource which is requested by the client is carried by the SOAP body. Figure 5 clarifies the structure of SOAP-HTTP message used in carrying the capability token from resource provider to the client.
Fig. 5 Capability token in SOAP-HTTP binding message

Resource provider is responsible to bind the capability token into a SOAP envelope and further bind the SOAP envelope into a HTTP response message. To facilitate the binding process of SOAP envelope into a HTTP message, a Java technology JAX-WS [26] is adopted inside our Metro web service stack. JAX-WS could develop a SOAP envelope for web services and exchanging this SOAP envelope through a XML remote procedure calls. Once a client receives the HTTP message which contains the SOAP envelope, he then intercepts the HTTP message and disassembles the SOAP envelope. The disassembling process extracts the capability token assertions and these assertions can be stored in client repository.

VI. CONCLUSION

This paper describes the Distributed Resource Protection Mechanism (DRPM) framework in managing the enterprise authentication and resource permissions authorization. We have preliminary identified several security challenges faced by the enterprises in enforcing the resource access permissions. To address these challenges, we adopted client profile and capability token inside DRPM framework. Client profile permits an effective identification of each client before any access to the resources is granted by the resource provider.

The capability token stores all access permissions granted to a particular client on each resource. Further, it allows an enterprise to delegate the resource access permissions to each of its employees. We also discuss our implementation of the capability token in this paper. The present implementation of DRPM framework provides initial evidence that the solution is robust and highly flexible for an enterprise to enforce its resource access permissions.

VII. REFERENCES