

# Securing VO Management

Florian Kerschbaum<sup>1</sup>, Rafael Deitos<sup>2</sup>, and Philip Robinson<sup>1</sup>

<sup>1</sup> SAP Research, Karlsruhe, Germany  
firstname.lastname@sap.com

<sup>2</sup> Automation and Systems Department  
Federal University of Santa Catarina, Florianópolis, Brazil  
deitos@das.ufsc.br

**Abstract.** In this paper we propose a security architecture and mechanism for Virtual Organizations (VO) for businesses. The VOs we consider are based on web service technology to address interoperability issues and cater for future business software, and are dynamic, i.e. their membership may change frequently throughout their lifetime. We improve over previous approaches in the following aspect: We have designed, implemented and evaluated a comprehensive security mechanism for our architecture that can protect both the web services in the VO and the VO management services. The security policies of VO management are enforced by inspecting the request for the encodings of parameters that are relevant to the policy decision. The basic idea may be applicable to other web service based software with data-dependent security policies, e.g. databases.

## 1 Introduction

In collaborative world-wide business processes scenarios a large number of organizations interact dynamically sharing their resources. These collaborations can be arranged in Virtual Organizations (VO). A VO consists of a collection of individuals and institutions defined according to a set of resource sharing rules [4]. This definition covers the technical view point favored by the Grid community, while in the business community, the purpose of the VO is emphasized as in the following definition: A VO is a temporary coalition of geographically dispersed individuals, groups, enterprise units or entire organizations that pool resources, facilities, and information to achieve common business objectives [12].

The work in this paper considers VOs with the following properties:

- *dynamic*: VOs evolve during operation, e.g. allowing member replacement.
- *business process driven*: VOs where the interactions are defined by a business process (choreography).
- *web service-based*: VOs where the shared resources are web services.

A (business process) choreography is the description of the flow of visible interactions, i.e. web service calls, of the participating organizations. A standard language for choreography of web services is the Web Services Choreography

Description Language (WS-CDL) [13]. Orchestration based VOs can also be represented by a choreography where the orchestrating party acts a separate role in the choreography. In this paper choreography is therefore seen as the more general concept encompassing both concepts of regular choreography and orchestration.

A VO management system facilitates the administration and management of such VOs. It enables the creation, deletion and other operations on the state of a VO. This functionality is encapsulated in a component called lifecycle manager. Furthermore, VO management assigns individual organizations roles in the business process thereby implicitly adding, removing and replacing members of the VO. This functionality is encapsulated in a component called membership manager. Each component is a separate web service that can be called by any type of client that wishes to establish or administer a VO. For details of lifecycle and membership management see [12].

Securing VOs (or similar forms of collaborations) has been considered before [2, 3, 10, 16]. The focus of previous work is to secure the interactions of the organizations, i.e. the authentication and access control decision for the web services. The security of the VO management services (referred to as infrastructure services), i.e. the access control policies of the administration interface, are not considered. We propose a security architecture and mechanism that addresses both, the security of the VO operation, as well as its management. The unique outstanding feature is that our architecture uses one security mechanism deployed before resources and VO management services.

The advantages of having one security mechanism are two-fold:

First, there is reduced security engineering effort in implementation and design of security critical components. The number and size of these security critical components decreases by combining the security mechanisms for the resources with those of VO management. Furthermore, the security of VO management services has been externalized to the security mechanism, such that VO management services can now be implemented without security in mind and without embedded security checks. In summary, our overall implementation effort was reduced with this security architecture.

Second, the administration of the overall system becomes easier, since the administrator has to deal with and learn only one security policy language. Instead of a set of heterogeneous policy languages and mechanisms, one mechanism can handle all. This in turn can increase the verifiability and auditability of the system, since consistency check can be performed more easily. In conclusion, our system is more easy to administer and control.

Our particular security mechanism has further advantages:

First, the policy language makes administration of VO management security more flexible. Hard-coded, mandatory security checks have been replaced with configurable, policy-driven checks. Our system can therefore cater for a variety of VO management security requirements.

Second, our security mechanism can be deployed in front of any web service without changes to that web services. Any service implemented according to web

service standards can be retrofitted with security and placed into a VO. This makes use and setup of VOs more comfortable and opens it to wider range of legacy services.

The contributions of the paper are

- the design of a VO security architecture and mechanism that allows for distributed control using one security mechanism for VO management services and resources. This includes the design and specification of a set of VO management policies and the security token mechanisms and key distribution to establish trust relations.
- an implementation and an evaluation of the security mechanism. We implemented the security mechanism as part of a larger research project and evaluated its impact on web service invocations.

The remainder of the paper is organized as follows: Section 2 gives an overview of related work, before Section 3 describes the components in the system. Section 4 describes the security components and algorithms and Section 5 summarizes the performance evaluation. Section 6 concludes the paper.

## 2 Related Work

Other approaches for the authorization challenge in VOs have been proposed in the literature which we will review next. KeyNote and PERMIS do not address the challenge of distributed management of policies, but were rather designed for one trust domain. Akenti focuses on the policy decision as well and leaves the administration question open, but offers the possibility for delegation. CAS and VOMS are designed to be used in distributed administered VOs, but neither considers yet the question of how to secure the infrastructure services themselves.

- **KeyNote:** KeyNote is a trust-management system that defines a language to specify security policies, actions, principals and credentials, and uses a compliance checker to validate access requests [1].
- **Akenti:** The Akenti authorization system [16] is a security model and architecture that uses authenticated X.509 identity certificates and distributed digitally signed authorization policy certificates to make access decisions about distributed resources.
- **PERMIS:** There are three main components to the PERMIS implementation: the authorization policy, the privilege allocator (PA) and the PMI application programming interface (API) - details can be found in [2].
- **CAS:** The Globus team first proposed a security architecture for grids in [5]. This architecture did not yet address the problem of distributed administration of resources, but those of protecting user credentials and identity mapping which are not covered in our architecture. The Community Authorization Service (CAS) proposed as a solution for specifying and enforcing community (VO) policies allows resource owners to grant access to blocks of resources to a community as a whole, and let the community itself manage fine-grained access control within that framework [10] using capabilities [6].

Granting access to a community gives total control of the local resources to the CAS server administrator [9,10,18]. Furthermore, the problem of securing the infrastructure services, such as the lifecycle and membership manager or the CAS server itself have not been addressed.

- **VOMS**: The Virtual Organization Membership Service (VOMS) can be considered a specialization of CAS [3].

### 3 System Architecture

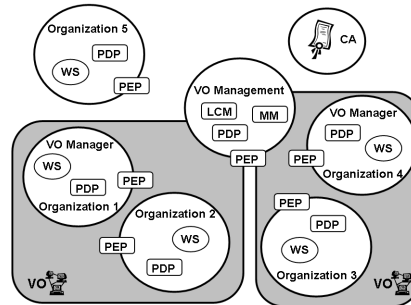


Fig. 1. System Architecture

Before joining a VO, an organization must obtain a global identity certificate. The certificate authority (CA) issues this certificate to the organization signed with the private key of the trusted root certificate in a public-key infrastructure (PKI).

The VO infrastructure consists of the following components:

- **Web Service (WS)**: This is the resource (web service) offered by an organization to other participants in a VO.
- **Policy Enforcement Point (PEP)**: The policy enforcement (PEP) is a reference monitor that intercepts every web service call to an organization's web services. The PEP authenticates the caller's identity and verifies all supplied credentials (e.g. in a challenge-response protocol). It then forwards the caller's identity, the attributes of the credentials and the details of the web service call to the policy decision point (PDP). The separation of the access control function into policy enforcement and decision point has been suggested in [11]. After receiving the PDP's access decision it either blocks or forwards the call to the web service (WS). Our security mechanism is a special PEP that not only protects the web services of the VO, but also the VO management web services, i.e. it intercepts calls to both.

- **Policy Decision Point (PDP)**: The policy decision point (PDP) receives the caller’s identity, attributes and the web service call details from the PEP. It then evaluates the stored policies and returns either a grant or deny decision to the PEP. All our policies are implemented using XACML [8] in a role-based access approach [15]. For a good survey of access control see [14]. Access control policies for web services are generated using the approach described in [13].
- **Lifecycle Manager (LCM)**: This service is part of the VO management and it allows the creation and deletion of VOs. It also stores the choreography of the VO. The lifecycle manager issues attribute credentials to the creator of a VO.
- **Membership Manager (MM)**: The membership manager, as part of the VO Management, assigns organizations to business roles. The security significance of the VO management services are detailed in the next section.

### 3.1 VO Management Interface

The VO management interface is described only briefly due to space requirements. The lifecycle and membership manager offer methods for VO creation, VO deletion, choreography retrieval, role listing, role assignment (and removal) and member replacement.

## 4 Security Architecture

### 4.1 Security Model for VO management

The current model for security of VO management is simple. Each VO has a VO manager that is responsible for all administration tasks. The VO manager can access all administration services (MM and LCM) and perform the necessary operations. VO members may use the query management services (MM) to get information about the VO they are part of, i.e. the list of the roles, etc. On the other hand, non-members should be prohibited from obtaining any information about the VO. There is a trifold security hierarchy of manager, member, and everybody, similar to UNIX file system access. To ensure the consistency of the VO management model, a set of management policies is generated according in role-based access control.

### 4.2 Roles

The roles in the policies of the security architecture are the business roles (*BP-ROLE*) as described in the choreography of the VO. To support VO management security, this has to be extended with a second role type for the VO manager (*VOMANAGER*).

In the context of web service security, RBAC policies allow one organization to limit the access to only that role. Compared to capability-based approaches, such as CAS, where the service is exposed to all members of a VO, the organization can now limit the access to a subset of them.

### 4.3 VO management policies

Besides the policies generated for the security of the interactions, there are policies that need to be enforced for the security of the VO management. The security mechanism not only verifies access, but also the content of the web service call, greatly enhancing its capabilities. The set of (allow-)policies derived from our security architecture can be represented as follows in Figure 2.

```
{
  Role: *
  Target: Lifecycle Manager
  Operation: createVO()
}
{
  Role: VOMANAGER
  Target: Membership Manager
  Operation: assignRole()
}
{
  Role: VOMANAGER
  Target: Lifecycle Manager
  Operation: deleteVO()
}
{
  Role: VOMANAGER
  Target: Membership Manager
  Operation: removeRole()
}
{
  Role: BP-ROLE
  Target: Lifecycle Manager
  Operation: getChoreography()
}
{
  Role: BP-ROLE
  Target: Membership Manager
  Operation: getRoles()
}
```

Fig. 2. Policies for VO Management Security

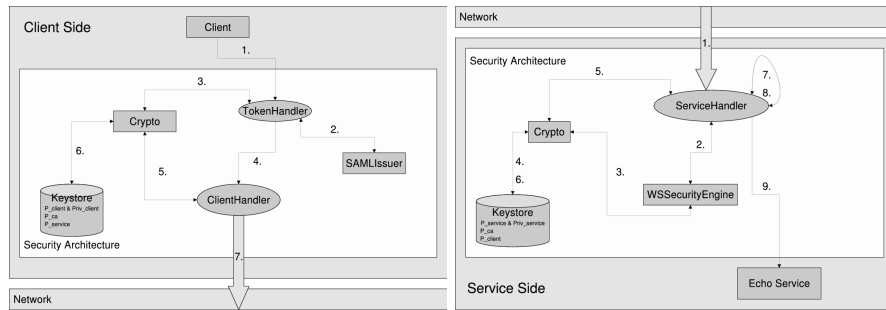
### 4.4 Role Assignment

This section clarifies the distribution of administration tasks between the different security administrators. The LCM issues an attribute credential attesting the role *VOMANAGER* to the creator of a VO (as a return value of the web service call). This credential can be either signed by a key belonging to the LCM only (with a certificate signed by the CA) or signed by the CA itself. Then the VO manager assigns the roles in the business process choreography (*BP-ROLE*) to specific members. For each assignment the VO manager creates an attribute credential with the *BP-ROLE*, the member's identity and the VO identifier. This credential is issued to the member, i.e. all role assignments are done by the VO manager.

### 4.5 PEP implementation

The PEP implementation is able to authenticate the caller's identity, verify all supplied credentials, forward the information to the PDP and grant/deny access. Our implementation is based on the Tomcat server with the Axis web service framework and supporting libraries for XML security and SAML [7]. For encryption, two distinguished methods are used: a symmetric and an asymmetric key algorithm.

The architecture works as an external layer, or module, that can be configured for any web service, setting handlers to intercept all the messages and perform the security tasks. Figure 3 depicts the use of the security layers for a simple echo service. This web service echoes any received message back to the sender. The security layer can be abstracted from the service layer. The service need



**Fig. 3.** Overview of the client and server interactions in the implementation

not know about the security handlers and only care about its own logic, i.e. any legacy web service can be secured and used in a VO. The implementation of the PEP is divided into three handlers: *TokenHandler* and *ClientHandler* on the client's side and *ServiceHandler* on the server's side.

**Outgoing Message** The steps for an outgoing message are depicted with numbers on the left side of Figure 3. Every time a client (using the security mechanism) sends a message to the WS (step 1), the message is intercepted first by the handlers. Two distinct actions take place: first, the token processing, and later, the signature and encryption processes.

Firstly, on step 2, the *TokenHandler* uses a *SAML Issuer* object to add signed tokens to the header of the message. The tokens (SAML assertions) considered must contain one subject, identified by a X.509 certificate, and, at least, one attribute statement with two attributes: role name and VO identifier (VO-id). It is possible to chain tokens, i.e. the VO manager can be the subject of the token issued by a CA granting the *VOMANAGER* role. Then, the VO manager can issue tokens (signing them with his own key), having others as subjects, granting the businesses roles, creating a chain of tokens.

To prevent unauthorized access to information that only belongs to a specific VO, each VO has an associated symmetric key, shared by the partners. So, after the token insertion, the role name attribute is encrypted (step 3) with the corresponding VO key. This key is currently not being renewed, when the membership of the VO changes, since it is assumed that the party leaving the VO will not receive future messages and therefore encrypted tokens.

After the token processing is completed, the message is intercepted by the *ClientHandler* (step 4) that signs the message's body (steps 5 and 6). Then, the handler encrypts the message's body with the public-key of the recipient before it is sent over the network (step 7).

**Incoming Message** The steps for an incoming message are depicted with numbers on the right side of Figure 3. The incoming message is intercepted by the

*ServiceHandler* (step 1), which coordinates the verification process. The first verification stage is executed by the *WSSecurityEngine* (step 2). This engine decrypts the message's body with the private key and then verifies the message's body signature (step 3). Next, for each token, the role name attribute is decrypted using the symmetric key associated with the VO-id specified in the corresponding attribute. In case the receiver does not have the key, the attribute is left encrypted.

The next stage is executed by the *ServiceHandler* itself and consists of three major tasks: checking the certificate authenticity (steps 5 and 6), completing verification of the tokens (step 7) and the VO-id parameter verification.

The VO-id parameter verification (step 8) ensures the scope of the message (what VO) and extracts the right permissions (attributes) for the PDP. It is that enables the PEP to secure the VO management services and externalize their security enforcement. Note that there could be more than one VO, and the two partners could be in more than one VO together, i.e. the client could have different permissions in each VO, e.g. in one he is the VO manager and in the other he plays a business role. So it is important to extract the correct attributes for the specific call, in particular for calls to VO management services.

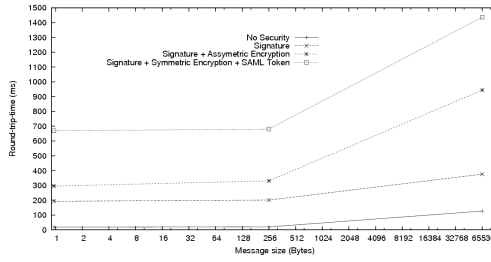
Every message to a VO management service contains the VO-id parameter in the SOAP body element, which is sufficient to detect the scope of the message, although, this information alone is not enough to enforce the policies. The VO-id parameter in the SOAP body is identified by flexible parsing and its special, fixed encoding in its namespace. The VO-id parameter from the SAML assertion (in the SOAP header of the message) is also extracted. The assertion is located and the attribute statement for the VO-id is retrieved. Having both VO-id parameters, it is possible to correlate messages to a VO. By extracting the name of the role from the same assertion, it is possible for the PDP to evaluate the access request against the policies of Figure 2 for this message (in the correct VO scope). If all the verifications are passed successfully, the message reaches the target service (step 9).

## 5 Performance Evaluation

We measured the overhead created by the insertion of the security mechanism in a web service invocation. The echo service was used for the evaluation and four test cases were defined:

- the message is exchanged with no security
- the message, on both sides, is signed by the sender and then verified by the receiver
- the message, on both sides, is signed and encrypted by the sender, and then verified by the receiver
- a token is added, the attribute statements are encrypted and later, the message is signed and encrypted; at the receiver side, the message is verified.

To compare the different test cases, the round-trip time (RTT) of the messages is used. On each test case, the size of the message's content assumed three



**Fig. 4.** Graphical representation for time measurement

different values: empty string, 256 bytes and 64 Kbytes of data. For each size on each test case, a sample of 110 values (RTT) was gathered. Three rounds of measurements were done. First of all, the size of the resulting *SOAPMessage* at the TCP layer was measured, and is depicted in Table 1.

	Empty String	256B	64KB
<b>No security</b>	371	627	65907
<b>Signature</b>	3163	3426	68707
<b>Signature + Encryption</b>	4777	5127	93311
<b>Signature + Encryption + SAML token</b>	14866	15215	103395

**Table 1.** Message size in bytes for the test cases

We can conclude that the overhead of the architecture in the size of the message is a linear function, e.g. the size of the message is approximately 1.5 times the size of the content plus a constant overhead. Figure 4 is the graph for time measurement. It represents the average values for the three rounds from the sample. The four different curves represent the four test cases for a different amount of data.

## 6 Conclusions

We present a security architecture and mechanism for VOs in business. It uses one security mechanism for protecting the VO management services as well as the resources offered in the VO. The advantages of having one security mechanism are reduced implementation effort due the reduction of security critical components and easier administration, since the administrator needs to deal with only one security mechanism. The basic idea of verifying the parameters of a web service call to the supplied credentials may be useful in other data-dependent web services, e.g. databases. Our evaluation of the implementation showed that the overhead introduced by the security architecture is acceptable for practical deployment. Future work is to extend the infrastructure services, e.g. by reputation services, but still secure them with the same security mechanism.

## References

1. M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *1998 Security Protocols International Workshop*, 1998.
2. D.W. Chadwick and O. Otenko. The permis x.509 role based privilege management infrastructure. In *Future Gener. Comput. Syst.*, volume 19 (2), pages 277–289. Elsevier Science Publishers B.V., 2003.
3. Y. Demchenko, L. Commans, C. de Laat, M. Steenbakkens, V. Ciashini, and V. Venturi. Vo-based dynamic security associations in collaborative grid environment. In *Workshop on Collaboration and Security (COLSEC) of The 2006 International Symposium on Collaborative Technologies and Systems (CTS)*, 2006.
4. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. In *International Journal of High Performance Computing Applications*, volume 15 (3), pages 200–222, 2001.
5. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *P5th ACM Conference on Computer and Communications Security*, 1998.
6. S. Mullender and A. Tanenbaum. The design of a capability-based distributed operating system. In *The Computer Journal*, volume 29 (4), pages 289–299, 1986.
7. OASIS, 2002. Security Assertion Markup Language (SAML) 1.0 Specification.
8. OASIS, 2005. eXtensible Access Control Markup Language 2 (XACML) Version 2.0 Specification.
9. L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The community authorization service: Status and future. In *Conference for Computing in High Energy and Nuclear Physics (CHEP)*, 2003.
10. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *IEEE Workshop on Policies for Distributed Systems and Networks*, 2002.
11. R. Guerin R. Yavatkar, D. Pendarakis. A framework for policy-based admission control. In *RFC 2753*, 2000.
12. P. Robinson, Y. Karabulut, and J. Haller. Dynamic virtual organization management for service oriented enterprise applications. In *1st International Conference on Collaborative Computing*, 2005.
13. P. Robinson, F. Kerschbaum, and A. Schaad. From business process choreography to authorization policies. In *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2006.
14. P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design*, 2001.
15. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role based access control models. In *IEEE Computer*, volume 29 (2), 1996.
16. M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. In *ACM Transactions on Information and System Security*, volume 6 (4), pages 566–588, 2003.
17. M. Thompson, S. Mudumbai, A. Essiari, and W. Chin. Authorization policy in a pki environment. In *1st Annual NIST workshop on PKI*, 2002.
18. V. Welch, R. Ananthakrishnan, S. Meder, L. Pearlman, and F. Siebenlist. Use of saml in the community authorization service. In *Computing in High Energy and Nuclear Physics*, 2003.