

## Integration of a Security Product in Service-oriented Architecture

Aleksander Dikanski, Christian Emig, Sebastian Abeck  
*Research Group Cooperation & Management, Universität Karlsruhe, Germany*  
 {dikanski | emig | abeck} @cm-tm.uka.de

### Abstract

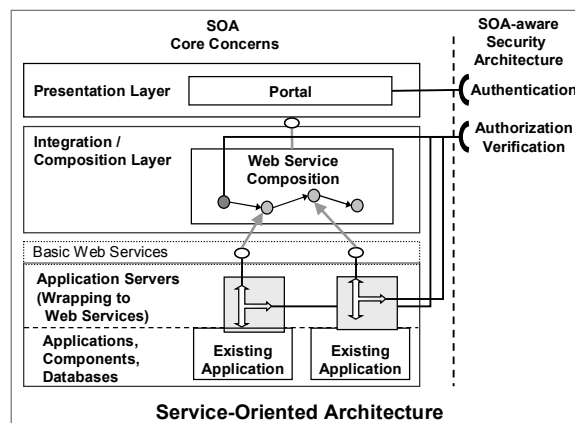
*The future of enterprise software development lies in the use of a service-oriented architecture (SOA) to support business concerns. Business services are using security services offered by service-oriented security architectures for security support. The question remains how to implement the security services using traditional security products and how to map security policies defined at service level to product-specific policies. In this paper we present an approach for integrating existing security products into service-oriented security architectures. We show how traditional security products can be adapted to fit into the overall service-oriented paradigm. We present a case study that applies our approach.*

### 1. Introduction

To tackle the increasingly complex requirements of IT systems, enterprises are adopting service-oriented architecture (SOA) to align their IT with their business processes. While Web service technology is commonly used to implement core concerns, cross-cutting concerns are hard to integrate into the overall SOA development process. Especially security is often an afterthought, considering the amount of overly complex Web service security standards. A service-oriented solution to this problem is to provide *security as a service*, i.e. offering a set of services, offering the central functionality of e.g., authentication, authorization and policy management. These services are part of a security architecture, of which we presented a blueprint of in [1].

Yet the critical task of the security services requests the reuse and therefore integration of existing security products into the security architecture. Additionally, focusing on access control, the respective policy models at service and at product level must be aligned. The integration requires two main tasks. At first, the

components of the security product need to be mapped to the logical building blocks of a security architecture. Design gaps between the provided service interfaces of a security architecture and the proprietary interfaces of the security product require adaptation. Secondly, access control policies for an SOA are specified in a language not supported by the security products and therefore need to be mapped to the internal policy language of the security product.



**Figure 1. Service-oriented Architecture requiring Security Services**

The contributions of this paper are:

1. We present an approach to transparently integrate existing security products into a security architecture, focusing on the access control service.
2. We show how access control policies for Web services can be mapped in order to be used by the integrated security product.

In order to achieve these goals, we built upon our previous works concerning service-oriented security architectures and access control policy models [1]. We demonstrate our approach by integrating the commercial of the shelf (COTS) security product Tivoli Access

Manager of IBM into our service-oriented security architecture and by mapping our service-oriented access control policies onto product-specific policies.

The paper is organized as follows: Section 2 presents approaches to security architectures focusing on access control and access control models. In Section 3 we present our approach to the integration of existing security products into a service-oriented security architecture. In Section 4 we show how to derive product-specific access control policies from product-independent ones. In Section 5 our approach is applied practically in a case study. A conclusion and an outlook on future research in this area close the body of this paper.

## 2. Related Work

### 2.1 Access Control Architectures

The access control architecture described in [2] uses multiple distributed access control processors (ACP) to control access to Web services. An ACP itself is a Web service which is responsible for access control decisions. A special ACP called gatekeeper is used to put the access control decision for a Web service into action, authenticating users and issuing security tokens. It also determines the necessary ACP to compute the access control decision for an access request. The amount and order of the ACP is an open design decision. The interceptor pattern [3] is used to implement the gatekeeper in order to intercept the SOAP messages sent to a Web service. The flexibility of the approach is also its greatest weakness when it comes to the implementation of the ACP using existing security products. Obviously each ACP could be implemented by a security product, but there is no need for all the ACP to be Web services themselves. A service should offer a coarse-grained interface which reduces the necessary invocations of that service, as these are expensive operations. Calling multiple ACP Web services leads to a tremendous performance loss.

In [4] an approach for a policy based access control architecture is proposed using an event-driven paradigm. The coordination between the security services takes place using the exchange of events. This allows the distribution of the services without a central control. The functional services themselves are distributed into a digital identity management, an authentication management as well as an access control service. Each service has a similar structure consisting of a decision point which evaluates related policies and an enforcement point which puts the decisions into place. In or-

der for each service to maintain a coherent view of all security relevant information and to be able to create a shared context, a notification service and a context service are introduced into the architecture. Questions still remain concerning the interaction between the services, the format of the events and the handling of event messages. This approach also suffers from the overuse of Web service interfaces. Additionally one can question the choice of an event-driven approach for the communication between the services. Events are typically used for asynchronous communication between distributed partners in an unreliable network. Assuming that one applies the security architecture in the controllable and reliable environment of an enterprise intranet the use of synchronous communication might be more applicable.

Although each of the security architectures tries to solve a certain problem, they are not suited for the task of integrating an existing security product efficiently. This is due to the lacking differentiation of distinguished service interfaces offered to the business services of a SOA and the internal component interfaces used by the security architecture itself.

### 2.2 Access Control Models

SecureUML [5] is an access control model based on an extended version of role based access control (RBAC). In the model, the nature of the protected object is undetermined and can be adapted to multiple use cases. This is demonstrated by defining policy languages for process and component systems. Policies contain the possible actions a subject can perform on an object and are supported by basic attribute based authorization constraints. Opposite to the basic RBAC, basic actions can be composed. A policy defined for a compound action is passed on to its basic actions. Although the approach is flexible enough to be used in the context of a service-oriented architecture (SOA), the usage in a security architecture and of a specialized security product is not the main focus. Instead the security infrastructure of an underlying development platform is utilized. Additionally, the complex context of an SOA, e.g., a composition of Web services, can not be represented with SecureUML.

RBAC is also the basis for SECTET-PL [6], a policy language for business workflows using Web services. It is based on the declarative Object Constraint Language (OCL) [7] and can therefore be used in the context of system models defined using the Unified Modeling Language (UML) [8]. Policies are specified using predicates and are attached to Web services defined in an interface view. This view contains descriptions of the partner services interface, service ex-

change documents as well as roles and permissions. The usage of the approach is demonstrated by mapping the policies to the policy language eXtensible Access Control Markup Language (XACML, [9]). Most of the arguments against SecureUML hold for SECTET-PL as well. Another constraining argument is the indispensable focus on OCL and UML to define the policies.

In summary comparing the approaches to the aim of integrating existing security products it can be seen that RBAC alone is not flexible enough for specifying service level policies [10]. The mapping of service level policies to product level policies has so far only been shown for ABAC policies [11].

### 3. Architectural Integration

Security is not a core concern of a service-oriented architecture (SOA). Yet it is an important aspect of practical usage. To cover this cross-cutting concern, security should be offered as a set of services itself. We proposed a well-defined set of security service interfaces to be used by Web services and in an SOA in [12]. Existing and field-tested security products should be used to implement these interfaces. In this Section we describe relevant parts of our security architecture followed by an overview of the IBM Tivoli Access Manager (TAM). Then we show how the components of the TAM fit into our architecture.

#### 3.1 Service-oriented Security Architecture

The security architecture we described in [1] and [12] consists of three logical layers. The first layer contains the well-defined and stable service interfaces to be used by other services. The authentication interface provides operations to authenticate a subject and to issue a temporary security token to be used for further access control. An access control decision can be delegated to the authorization verification service interface. Management of users, groups as well as access control policies is done through the administration service interface. We also presented a set of possible components, implementing these interfaces. In a functional layer, a secure token service component performs the authentication of subjects. A policy decision point (PDP) component encapsulates access control decisions logic. Lastly an administration component implements the administration interface. The functional components store and retrieve their data from components placed in the data layer below.

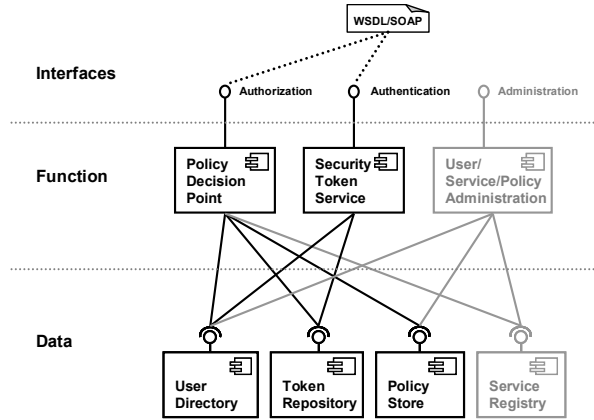


Figure 2. Blueprint of an SOA-aware Security Architecture

Notice that with the exception of the interface layer, the components and their respective communication technology are not fixed. They are only provided as reference components. As only the interfaces are determined, existing security products can be integrated. In the following we show how the security product Tivoli Access Manager, presented in the next Section, can be used to provide the required functionality.

#### 3.2 Overview of the Tivoli Access Manager

The Tivoli Access Manager (TAM) is a universal authentication and authorization framework used in several security products.

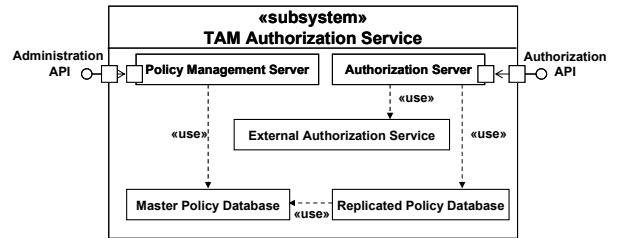


Figure 3. Architecture of the Tivoli Access Manager

The overall architecture of the authorization service provided by the TAM, shown in Figure 3, consists of a Policy Management Server, a Master Policy Database and one or more Authorization Servers. The Policy Management Server manages users, groups, policies and protected resources. The resources are defined for a security domain and are stored in the Master Policy Database. Distributed copies of the Database are used by the Authorization Servers which are deployed for localized access control decisions. Interfaces can be used by policy enforcement points to request an access

control decision from the servers. A more extensive overview of the TAM framework is given in [13].

### 3.3 Approach to Integration

Integration of the Tivoli Access Manager (TAM) into our security architecture requires a mapping of interfaces as well as data formats. The adapter pattern is an appropriate approach to perform this task [14], i.e. an additional component, converting the invocation of the service operations into the equivalent methods of the interface of the TAM, must be implemented in order to leave existing clients of the security services unchanged. This can lead to typical integration problems depending on the chosen service interface and the security product. Enterprise application integration (EAI) and other current software engineering approaches provide best practice advises for solving such an integration task.

Accessing the security products' authorization functionality is a typical problem. Two extreme cases of interface provisioning can be distinguished. Firstly, the security product could offer no explicit interface at all, in which case a change of the security product might be advisable. Secondly, the security product offers standardized and well documented interfaces, which simplifies the integration. Assuming a security product offers an interface, the next problem is to map the policy data formats. Service level security data provided via the service interface might be represented by multiple data structures, it might not map at all on the product side and vice versa. If possible one of the two data formats can be changed to match the other one. Otherwise a mapping must be developed which allows the transformation of the greatest possible subset of the data.

If the TAM is to be integrated into the security architecture the approach is less problematic. The simplified authorization verification service interface of our security architecture provides a single authorize-operation, which receives an authorization request message and returns an access control decision in form of a Boolean value. The request message contains the unique identifier of the requested resource, a session token issued to the user on authentication, as well as other security related data. In case of the TAM this represents the minimal information necessary to perform an access control decision. The TAM provides amongst others an object-oriented authorization interface consisting of several classes. Therefore the task of the adapter component is to map the invocation of the coarse-grained service operation to a sequence of fine-grained object operations. In order to use the interface the adapter component must be registered in the TAM

using provided configuration tools. Furthermore, the session token of the user needs to be issued by the same TAM instance so that the resource identifier is known to the TAM. Using these pieces of information the relevant objects of the TAM interface can be created and the access control decision be requested from the Authorization Server. Additional parameters can be sent too, using attribute objects as containers for name-value pairs.

With this an integration of the security product can be performed at the software level. Further integration is needed for the different access control models to align the access control policies at service and product levels.

## 4. Access Control Model Integration

Integration of a security product into a service-oriented security architecture leads to the necessity of managing different access control models. This is due to the different granularity and languages of service level and product level policies. This task of integration can be subdivided into two subtasks. Firstly, the access control models at both levels need to be analyzed. Secondly, rules can be defined which map policies defined at the service level to policies at the product level. Manual adaptation is an error-prone process which is not feasible because of the resulting security issues. Therefore the policy mapping should be automated.

In this Section we present an approach for integrating different access control models. In the following we firstly give an overview of an extended version of the Web Service Access Control Markup Language (WSACML). Afterwards we describe the product level access control model of the Tivoli Access Manager (TAM). Finally we define rules to map WSACML policies to TAM policies.

### 4.1 Service Level Access Control

In [12] we presented a conceptual access control model based on attribute based access control (ABAC, [10]). Based on this, we developing the Web Service Access Control Markup Language (WSACML) to express attribute based access control policies at the service level [11]. The conceptual model of WSACML contains policy elements which are attached to objects. These policies contain one or more rules which determine the conditions necessary for permitting access to the object. These rules use attributes of the accessing subject (e.g., the identifier, the credential and the session token), the object (e.g., possible input parameter

to a service operation) as well as attributes determined by the context of the access request. The object hierarchy is specialized using Web service collections, i.e. a set of unrelated Web services, and Web service compositions, i.e. a set of Web services used in a business process, allowing the attachment of one policy to multiple Web services.

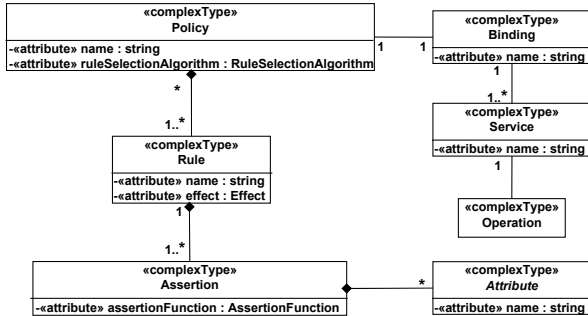


Figure 4. Web Services Access Control Markup Language (WSACML)

#### 4.2 Product Level Access Control

The access control model of the TAM, presented in Figure 5, consists of policy elements and a hierarchical arrangement of so called protected resources (PO). The latter represents the objects a user can access.

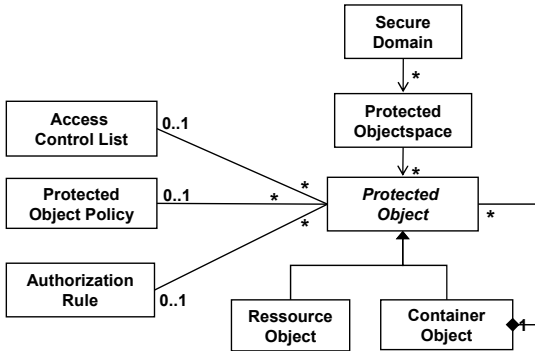


Figure 5. Authorization Metamodel for Tivoli Access Manager

All protected resources belonging to a particular organization unit are placed in a secure domain, which itself is partitioned into objectspaces. An objectspace is managed and secured by a policy enforcement point (PEP) and contains PO of the same type, e.g., Web based resources, services, message queues, file systems, printer etc. The PO can further be ordered by hierarchically arranging them using container objects. The PO can be explicitly secured by attaching policy elements to them or implicitly by attaching it to a container object.

TAM policies can be defined using access control lists (ACL), policies for protected objects (POP) and authorization rules. ACLs can be used for identity or role based policies. POP and authorization rules provide the possibility of defining policies related to the PO. Our main interest lies with the authorization rule policy element, which enable ABAC policies. The rules are based on an access decision information (ADI) document and are specified using the transformation language Extensible Stylesheet Language (XSL) [15].

#### 4.3 Mapping Rules

After analyzing the Web Service Access Control Markup Language (WSACML) and the Tivoli Access Manager (TAM) policy model, mapping rules between WSACML policies and TAM policies can be defined. A WSACML policy is attached to at most one object, which itself can be a container for other objects. E.g., a Web service collection is a container for a set of Web services and a Web service represents a collection of Web service operations. Such collections are mapped to TAM container objects. The objects in the collection are mapped to protected objects. All container and resource objects are placed in a predefined objectspace managed by a specialized policy enforcement point (PEP) for Web services, such as a secure service agent [16].

As WSACML policies are based on attribute based access control (ABAC), we make extensive use of the TAM authorization rule policy element. A WSACML policy can be represented by a single authorization rule using a XSL-choose construct. Inside this construct, the WSACML rules are mapped to XSL-when constructs. Using the assertion and assertion function elements of the WSACML policy a Boolean expression for the test attribute can be defined. The effect of a WSACML rule is mapped to the predefined return values of the TAM rule. The relevant data to be used by the TAM rule evaluation engine is provided by various data sources. Subject attributes can be retrieved automatically using the credential information of the requesting subject. Object attributes can be retrieved by an external provider using the TAM plug-in mechanism or alternatively by the PEP. The latter is used in the case of input parameters to a Web service operation. The same applies to environment attributes. The main problem is to determine the format of the ADI document used to evaluate the authorization rules. Using the above attributes types, a standard format can be chosen which provides the relevant data inside corresponding XML elements. E.g., the */subjectAttribute/role* expression refers to the *role*-element inside the

*subjectAttribute*-element. A tool to automate and support the mapping of service level to product level policies can be implemented on the basis of such a fixed set of mapping rules and the predefined format of the ADI document.

## 5. Case Study

We applied our approach to secure Web services used by a Web application. The application allows students to inspect their current academic record, a scenario typically found in the university domain. We used Web Service Access Control Markup Language (WSACML) policies to secure the access to the Web services by defining a service level policy with two rules. The first rule allows a user in the student role to review his/her and only his/her academic record. In this case the rule states that the unique identification number (UID) of the student must match the UID of the requested academic record and the requestor must be in the student role. A second rule allows a user in the counselor role to review all the academic records. In this case only the role membership is checked.

To secure the application we implemented an adapter for the Tivoli Access Manager (TAM) to encapsulate the transformation logic and to provide unified access as described earlier. For that we used stateless session bean components of the Enterprise JavaBeans (EJB) [17] component technology. The adapter component skeleton was created in a contract-first approach given the security service interface description provided in [12]. The Java classes provided by the TAM were used for accessing the authorization framework. The classes can be used either used directly or as an authorization provider in the authorization mechanism of the Java platform. The former approach turned out to be more usable in our case.

We further mapped the Web services and the WSACML policy for the application to equivalent TAM protected resources and authorization rules by applying the rules defined earlier. The resulting TAM rule consisted of two when-constructs, each representing an equivalent Boolean expression to the WSACML rules. These protected resources were stored in a container object inside a preconfigured objectspace. The policy rules were then linked to the container object.

To minimize possible errors introduced by manual mapping, we implemented a tool using Extensible Stylesheet Language (XSL) for our XML notation of WSACML. The tool produces an XML document describing the protected objects and the authorization rules in a format used by the policy import tool of the TAM. The import tool takes care of automatically con-

figuring the resources and policies. Defining the objectspace for Web service resources remains the only manual task.

## 6. Conclusion and Further Work

In this paper we presented an approach for transparent integration of existing security products into a service-oriented security architecture in order to use existing technology to secure a modern service-oriented IT infrastructure. The integration steps include software integration using adaptation as well as policy integration by mapping service level to product level policies. We showcased our approach by integrating the security product IBM Tivoli Access Manager into our security architecture and by mapping service level to technical product level access control policies.

We see two directions for further research. Firstly we aim to evolve our security architecture into a more distributed and decentralized one. Thereby increasing reliability as well as enhancing solutions to issues concerning privacy and distributed security information. Secondly, we are interested in bridging the gap between policies defined in the analysis phase and at the implementation phase of a software development process by applying the presented approach of automatic mapping of policies to the business level.

## 7. References

- [1] Ch. Emig, F. Brandt, S. Kreuzer and S. Abeck. "Identity as a Service - Towards a Service-Oriented Identity Management Architecture", *13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services (EUNICE 2007)*, Twente, Netherlands, July 2007.
- [2] R. Kraft, "Designing a distributed access control processor for network services on the Web", *Proceedings of the 2002 ACM Workshop on XML Security (XMLSEC '02)*, ACM, New York, NY, 2002, pp. 36-52.
- [3] Ch. Steel, R. Nagappan and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services and Identity Management*, Pearson Education, Upper Saddle River, N.J., 2006.
- [4] E. Bertino, L.D. Martion, "A Service-oriented Approach to Security - Concepts and Issues", *Proceedings of the 11<sup>th</sup> IEEE International Workshop on Future Trends of Distributed Computing Systems (FTGCS'07)*, IEEE Computer Society, Washington, D.C., 2007, pp. 31-40.
- [5] D. Basin, J. Doser, T. Lodderstedt: "Model Driven Security: From UML Models to Access Control

Infrastructures”, *ACM Transactions on Software Engineering and Methodology*, ACM, New York, NY, January 2006.

[6] M. Alam, R. Breu, M. Hafner: “Modeling permissions in a (U/X)ML world”, *First International Conference on Availability, Reliability and Security (ARES'06)*, IEEE Computer Society, Los Alamitos, CA, 2006. pp. 685-692

[7] Object Management Group (OMG): *Object Constraint Language*, [online] OMG, Available from: <<http://www.omg.org/docs/formal/06-05-01.pdf>>, [Accessed 4 Dec. 2008]

[8] Object Management Group (OMG): *Unified Modeling Language*, [online] OMG, Available from: <<http://www.uml.org/>>, [Accessed 4 Dec. 2008]

[9] Organization for the Advancement of Structured Information Standards (OASIS): *Extensible Access Control Markup Language*, [online] OASIS, Available from: <[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)>, [Accessed 4 Dec. 2008]

[10] E. Yuan, J. Tong: “Attribute Based Access Control (ABAC) for Web Services”, *IEEE International Conference on Web Services (ICWS 2005)*, Orlando, Florida, July 2005.

[11] Ch. Emig, S. Kreuzer, S. Abeck, J. Biermann, H. Klarl: “Model-Driven Development of Access Control Policies for Web Services”, *LASTED International Conference on Software Engineering and Applications (SEA 2008)*, Orlando, Florida, November 2008.

[12] Ch. Emig, F. Brandt, S. Abeck, J. Biermann, H. Klarl, “An Access Control Metamodel for Web Service-oriented Architecture“, *IEEE Conference on Software Engineering Advances (ICSEA '07)*, Cap Esterel, France, August 2007.

[13] G. Karjoth, “Access Control with IBM Tivoli Access Manager”, *ACM Transactions on Information and System Security (TISSEC)*, ACM, New York, NY, May 2004, pp. 232-257.

[14] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, Mass., 1995

[15] L. Quin: *The Extensible Stylesheet Language Family*, [online] World Wide Web Consortium (W3C), Available from: <<http://www.w3.org/Style/XSL/>>, [Accessed 4 Dec. 2008]

[16] Ch. Emig, H. Schandua, S. Abeck, “SOA-aware Authorization Control”, *International Conference on Software Engineering Advances (ICSEA '06)*, Tahiti, November 2006.

[17] Sun Microsystems, *Enterprise Java Beans Technology*, [online] Sun Microsystems, Available from: <<http://java.sun.com/products/ejb/>>, [Accessed 4 Dec. 2008]