Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 197 (2008) 31–43
www.elsevier.com/locate/entcs

Infrastructural Support for Enforcing and Managing Distributed Application-Level Policies

Tom Goovaerts[1] Bart De Win[2] Wouter Joosen[3]

DistriNet Research Group
Department of Computer Science
K.U.Leuven
Leuven, Belgium

Abstract

State-of-the-art security mechanisms are often enforced in isolation from each other, which limits the kinds of policies that can be enforced in distributed and heterogeneous settings. More specifically, it is hard to enforce application-level policies that affect, or use information from multiple distributed components. This paper proposes the concept of a Security Service Bus (SSB), which is a dedicated communication channel between the applications and the different security mechanisms. The SSB treats the security mechanisms as reusable, stand-alone security services that can be bound to the applications and it allows the enforcement of advanced policies by providing uniform access to application-level information. This leads to a security infrastructure that is more flexible and more manageable and that can enforce more expressive policies.

Keywords: security policies, policy management, security enforcement, distributed systems

1 Introduction

Distributed applications consist of interacting components that are deployed on various locations in the network. Nowadays, instead of being programmed from scratch, applications are often built by composing heterogeneous, reusable components. When security becomes a priority, several techniques exist to bind security mechanisms to these components. For instance, security mechanisms can be supported by the middleware such as an application server or a virtual machine, they can be injected in the code by security automata [20] or aspect-oriented programming techniques [6], or they can simply be embedded in the code of the components.

[1] Email:tom.goovaerts@cs.kuleuven.be
[2] Email:bart.dewin@cs.kuleuven.be
[3] Email:wouter.joosen@cs.kuleuven.be

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2007.10.012

In either way, the distribution of the functional components implies that the security mechanisms are spread throughout the infrastructure.

An important problem with the spreading of security mechanisms is that the enforcement of application-level policies affecting multiple locations (i.e. components) becomes difficult. Most mechanisms are not designed to support different locations – nor should they be – and while a number of specific technologies exist that might solve this problem to some extent (e.g., security protocols, JACC [21] or XACML [18]), they are by no means a complete solution. Extra infrastructural support is needed to connect the different, heterogeneous components and mechanisms together in a unified way. This will increase the necessary level of control of security policies in nowaday's applications and the management thereof.

The main contribution of this paper is the discussion of the Security Service Bus (SSB), a dedicated communication channel that interconnects the functional components and security mechanisms within a certain trust domain. The SSB treats security mechanisms as first-class, reusable services that can use information from components and that are bound to the components. The advantages of the SSB are the potential to enforce a broader spectrum of security policies, the unified view on the security infrastructure and the gained flexibility and manageability. Remark that this paper does not provide a concrete and finalized SSB solution. Rather, it elaborates on the requirements, advantages and challenges of the SSB concept.

In Section 2, we illustrate the need for the SSB by means of a concrete example. In Section 3, we discuss the SSB concept in more detail. Section 4 shows how to the SSB facilitates the enforcement of the policies from Section 2 and points out some other interesting applications of the SSB. Section 5 discusses related work and Section 6 concludes the paper.

2 Motivation

In this section, we discuss why the fact that security mechanisms operate in isolation from each other limits the enforcement capabilities of the security infrastructure.

2.1 Example

Suppose we have an enterprise that offers an online shopping application consisting of three main components (see Figure 1). The web site that presents the shop to the user is implemented in the Web Shop servlet component on a Java EE [22] web container. This component interacts with the Shopping Cart Enterprise Java Bean (EJB) which is deployed on an EJB container on another host and maintains the items that the user is going to buy. The web shop and the EJB component communicate with each other over the RMI/IIOP protocol [11]. The Shopping Cart component in its turn requires a Payment component which is deployed on a third host, but is implemented in the .NET framework. All communications between the Shopping Cart and the payment component are realized over the SOAP protocol [25].

The platforms that host each of these three components enforce a set of security

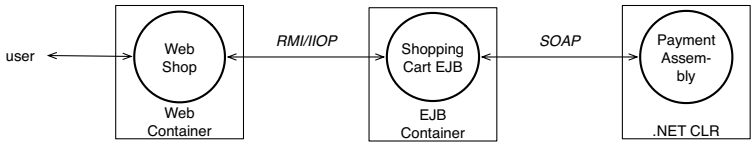

Fig. 1. The example scenario consisting of three interoperating software systems.

mechanisms and policies at runtime. Typical examples of security mechanisms that are enforced are user component-authentication, access control and auditing. Now, consider the following policies:

(i) At the web shop, deny all access to customers that are registered as bad payers at the Payment component.

(ii) At the host that runs the Payment component, ensure that the payment of a user is equal to the amount in its shopping cart.

(iii) When an unauthorized user repeatedly tries to access the administrative interface of the web shop, increase the level of auditing at all three sites.

The first and second policies illustrate that a policy might need access to information that is not local to the enforcement point for that policy. In the first policy, the access control mechanism of the web container needs information from the .NET component and in the second example, the access control mechanism of the .NET environment needs information about the EJB component. The third policy illustrates that security policies might affect several distributed components: the detection of an event at the web container triggers a change in the policies of different security mechanisms.

Without any further support, supporting policies like these is not straightforward. The first and the second policies depend on information that is located in other (remote) components. Some information about the caller (mainly its identity and/or its credentials) is usually propagated over communication protocols such as IIOP and SOAP, but evidently it is not feasible to propagate all kinds of information the security mechanisms will ever need, especially when this is application-level information that is not always known beforehand. One way of implementing these policies without any further support is to make all the components interoperable using a common protocol such as SOAP. Another approach is to let some components be proxies for components they are connected to. For instance, the EJB component could expose the bad payer attribute from the payment component to the web shop. For supporting the third policy, the auditing policies of the different middleware platforms would need to be exposed to the web container and the audit mechanism on the web container would need to be able to detect the attempted access to the management interface.

Implementing these policies in an ad hoc way may work to some extent but is far from optimal: dependencies on specific components need to be injected in the code of the components or into the security mechanisms. It is clear that these approaches are hard to manage and do not scale well if the information that is required changes/grows and if the policies change.

2.2 The Missing Link

The three policies from the previous section are hard to implement because they require application-level information from several distributed components, and because the security mechanisms operate in isolation from other mechanisms and components. This is especially hard when these components are heterogeneous. The sketched solutions always require the introduction of some form of communication channel between the security mechanisms on one hand and the functional components on the other hand. Once such a channel is put in place, the example policies can be supported much easier. The SSB precisely provides such an information channel, and it does so in a generic way.

While there is no generic approach for sharing security information, some specific solutions and building blocks do exist. In the first place, various dedicated and low-level security protocols exist that serve specific purposes such as key distribution or mutual authentication. These protocols allow the communication between two or more security mechanisms but they only focus on one specific security concern. Secondly, the sharing of security information is supported to some extent in the communication protocols that are used between the different parts of a distributed application. Several middleware protocols such as IIOP and SOAP have support for piggybacking security information to messages, for example in the form of certificates or assertions. These protocols are usually generic and they can use standardized representations such as WS-Security [17] and SAML [16], but they are only building blocks in the sense that the applications and the security mechanisms still have to agree upon which information should be propagated and under which form. E.g., it is trivial to propagate the user's role, but it is less straightforward to determine the fact that precisely the role needs to be propagated.

The increased distribution of applications over the past decades is being followed by an increased distribution and centralization of security mechanisms themselves. Traditionally, security mechanisms have been tightly coupled with the applications that need to be secured: applications either implemented the security mechanisms themselves or they made use of software libraries that provide security mechanisms. This monolithic design has evolved towards offering the security mechanisms in the middleware that connects the applications, such as virtual machines, object request brokers and application servers. Typical examples of concrete middleware-level security mechanisms are the CORBA Security Service [10] and implementations of the Java EE security model. The inevitable and ongoing next step in the sharing of security logic is to pull certain security mechanisms out of the middleware and to offer them as reusable services, that can be managed centrally and can be reused by many heterogeneous components. This evolution can be seen by the appearance of various security services such as Kerberos [15], Akenti [23], Tivoli Access Manager [13] and Shibboleth [1] to name a few.

The use of centralized audit and authorization services in our example would make supporting the required policies easier since we do not need to make the platforms dependent on each other. However, if the number of components and security services grows, the situation quickly becomes hard to manage and inflexible,

because each of the security services has their own way of communicating with the applications. The SSB aims at solving these issues by connecting the mechanisms and the applications using an intermediary abstraction layer.

3 The Security Service Bus

We propose to interconnect the different components and security mechanisms by means of a dedicated communication channel for security-relevant information such as security policies, security-relevant events and contextual information. In this section, we elaborate on this approach which we call the Security Service Bus. First, we discuss the most important requirements. Subsequently, we present the core concepts and components of the SSB and discuss their goal. Finally, we explore which kinds of security information can be exchanged on the SSB.

3.1 Requirements

The SSB is a dedicated communication channel that interconnects the different components of a distributed application and the security services. In order to guarantee the benefits of this approach, the following requirements need to be taken into account.

Flexibility Since the SSB needs to provide a more flexible security infrastructure, it must be flexible as a platform itself.

Security Besides giving increased flexibility, the SSB can also make the environment more secure, because there is a dedicated channel for sensitive security traffic. However, if the SSB is compromised, the consequences can be disastrous. Every single security service an application would be exposed to the attacker. Therefore, it is extremely important that the bus itself is highly secured.

Performance The more information security mechanisms can use, the larger the performance overhead can become, especially when the information is distributed. Because security mechanisms often block functional components (e.g. while making an authorization decision), the introduction of the SSB can have far-reaching performance consequences. Therefore it is important to keep the performance overhead under control. In a second step, the SSB can also be used to make the security infrastructure more efficient.

3.2 Conceptual Overview

The SSB is a dedicated security-specific communication channel that lets the components and the security mechanisms exchange information with each other. Based upon this communication channel, the SSB brings the established principle of separating functional logic and security logic to a distributed setting. It does so by virtualizing the security infrastructure as a set of interoperating components and security services that can be accessed, managed and bound to each other in a uniform way. Each component in a distributed infrastructure that needs to be secured

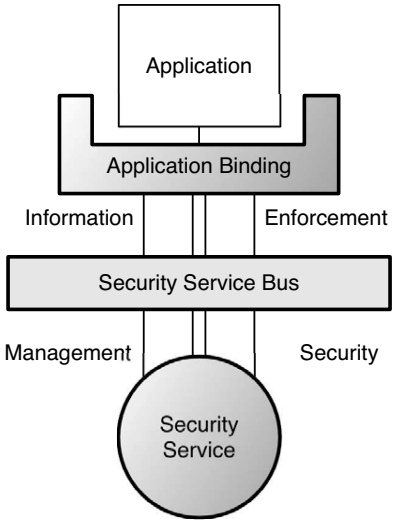

Fig. 2. Application bindings and security services.

and each security mechanism that provides security logic is connected and deployed on the SSB together with an exact specification of what the application or security service requires and provides in terms of security information and security logic. We assume that the SSB is deployed within a single trust boundary.

Security Services

The SSB treats security mechanisms as first-class reusable services that can be invoked, managed and composed with applications and with each other. The security services do not have to be completely independent components, they can also belong to existing middleware platforms. Security services are the components that contain the security logic and they implement a *security interface* and a *management interface* (see Figure 2). The security interface specifies the core security functionality. For instance, for an audit service, the security interface contains operations for auditing events and for an authorization service, it contains operations for making authorization decisions given certain contextual information. The management interface specifies operations that can be invoked to configure an manage the security service, such as loading a certain policy or enabling/disabling the service. The SSB keeps track of all security services that are registered and can it invoke their functionality when it is needed. The security services themselves can use the SSB for obtaining application-level information.

Application Bindings

Applications are bound to the SSB by means of an *application binding*. This is a wrapper component that presents an abstract view on an application to the SSB. The SSB aggregates all these views on the applications and provides a uniform abstraction layer to the security services that is independent of all application-specific details and that contains all application-level information available. When a

security service needs a particular piece of information, the SSB mediates the request and ensures that the application binding that offers the information is queried. The model that is used to specify the interfaces of the application binding consists of:

(i) A set of subjects S and subject attributes SA. Subjects represent the active entities in the application, typically the users.

(ii) A set of resources R and resource attributes RA. Resources represent the assets of the application that need to be protected. We assume that resources are organized hierarchically.

(iii) A set of actions A and action attributes AA. Actions represent the operations that can be executed on the resources. Each action is associated with one resource and one resource has many actions.

(iv) Three binary relations "." that associate subjects, actions and resources from S, A and R with their attributes from SA, AA and RA respectively.

The application binding implements of an *enforcement interface* and an *information interface*. The enforcement interface specifies all resources that are contained in the application and lists all the actions that can be performed on these resources. Resources and actions can correspond directly to low level concepts in the implementation of the application (such as a class and its methods), but they can also represent more abstract entities. The application binding guarantees that the behavior of the security services can be invoked each time an action on a resource is called. The security services can then use the SSB for obtaining information about the subject that wants to execute the action, the action itself and the resource on which the action is executed.

The information interface specifies which attributes about the subjects and resources of the application are made available to the SSB. Besides using information about a local invocation of an action, security services can make use of the information of other application bindings for obtaining additional information. The information interface consists of S, SA, R, RA and their association relations and of the hierarchy of R. In this case, the subjects do not only represent active subjects, but rather group all static and dynamic information an application has about a certain user, for instance, a user's age or telephone number. R represents all resources the application exposes such as an account or a form. The information that can be represented in the information interface either belongs to subjects or to resources. For some kinds of information it is possible to model it as a subject attribute or as a resource because it relates to both. For instance, a particular users' account at the payment component can be an attribute of the a subject or it can be a resource with the subject's name as an attribute.

A particular deployment configuration of the security bus will consist of a number of security services, a number of applications and a number of application bindings. One of the big advantages of the explicit notion of application bindings is that the correctness of the configuration can be verified by checking whether the appli-

cation bindings match the policies of the security services in terms of attributes and operations they must provide. In that sense, the main challenge in (re)configuration is to maintain this condition as an invariant.

3.3 Classification of Security Information

The definition of the security bus is intentionally kept rather broad. This section explores the possible kinds of security-related information that can be shared on the SSB. Three main categories of security information are distinguished.

Policies

Most security services are configured by a declarative policy that specifies the expected behavior of the service. Security policies are used to make easy adaptations to the behavior of the security service due to changing security requirements. Administrators of the SSB can change or query policies over the management interfaces of the security services, but security services can also transfer the policies themselves. For instance, an authorization service can reason about its policy and send an optimized sub-policy to another authorization service for performance reasons. Two major kinds of security policies are distinguished:

- *Configuration policies* specify various configuration parameters of a security service such as the location of an LDAP server, the use of specific keys or initialization vectors, or the kind of authentication mechanism that has to be used.
- *Security Policies* are specific to a security mechanism and are written in a language that is based on a particular security model such as RBAC [7]. These kinds of policies are more abstract than configuration policies. Typical examples of high-level security policies are access control policies, username-password mappings and role assignment policies.

Events

In the context of security information, events are things that happen in a system that are relevant for security. Often, an event is a trigger of security logic. Usually, an event is triggered by the invocation of an operation or the sending of a message. Security events are found at two levels:

- *Security-specific events* are events that are generated by security services themselves. Examples are authentication decisions or authorization decisions.
- *Security-relevant events* In most cases, security logic is initially triggered by an application-specific event. An application-specific event is an event about an application- or platform-level abstraction. An example of a generic security event is the invocation of an operation.

Contextual Information

This is application-level information that is used in various security policies. This information is represented in the form of attributes and can relate to subjects

or resources. Attributes can be application-specific, such as the bad payer attribute in the example, or they can be security-specific, such as the credentials or the role of a user.

4 Application of the SSB

In this Section, we show how the SSB can be used to enforce the concrete policies from Section 2.1 and we show how the SSB facilitates solving some more advanced scenarios.

Figure 3 illustrates how the SSB can be used to support the policies from the example in Section 2.1. The three components are now bound to the SSB. The interfaces of the application bindings are shown in Table 1. Because attribute mappings and resource-action mappings are trivial, they are omitted. Attribute values are not shown either. The enforcement interface of the web container lets security services intercept HTTP messages and the enforcement interfaces of the EJB container and of the .NET runtime can intercept method calls. The bindings are responsible for translating application state to the enforcement and information interfaces.

The audit and access control mechanisms from the three middleware platforms have been replaced by two security services, of which the security interfaces are shown in Table 2. Each of the middleware platforms now write their audit records to the audit service and query the authorization service for authorization decisions. The authorization and audit services now hold a policy that states which operations are permitted or audited respectively. These policies are expressed in terms of actions of the enforcement interfaces. When a user invokes an action on a protected resource (directly or indirectly) at one of the components, the middleware platforms notify the SSB of this event. The SSB then notifies each security service that needs to know this event, in our case both of the security services. [4] When the authorization service needs to make a decision for the first and second policies, it queries the information interfaces of the other applications via the SSB in order to obtain the *badpayer* attribute from the Payment component and the *totalprice* attribute from the cart of the active subject. If an authorization decision is made, it is returned to the application binding and another event is placed on the SSB. The audit service inspects the authorization decision events and when it detects a repeated denial event for the management interface of the Web Shop component, it raises makes its audit policy more strict.

This example only illustrates the basic functionality of the SSB but once the SSB is put in place, more advanced scenarios can be supported. For instance, the SSB can offer a session management service that allows different applications to share

[4] While it is true that this would generate an enormous amount of events, we stress that this discussion is held at the conceptual level. An implementation of the SSB can realize the same overall effect much more efficiently.

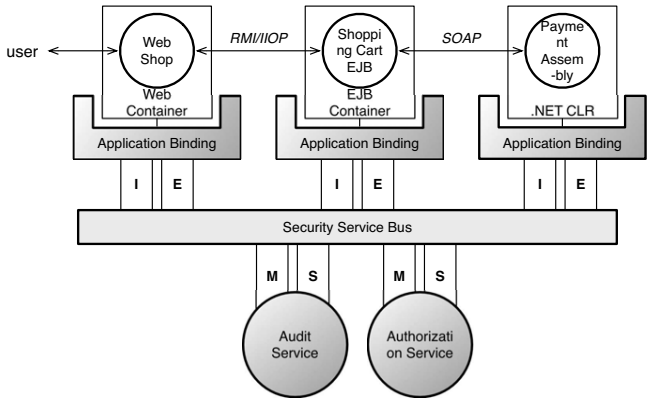

Fig. 3. The Example Revisited.

Table 1
The Application Bindings for the Example.

	Information Interface	Enforcement Interface
Web Shop	$S = \{alice, bob\}$ $SA = \{cert_a, cert_b\}$ $R = \{webshop, adminintf\}$	$S = \{alice, bob\}$ $SA = \{cert_a, cert_b\}$ $R = \{webshop, adminintf\}$ $A = \{post, get\}$
Shopping Cart	$S = \{alice, bob\}$ $SA =$ $\{nbitems, totalprice, role\}$ $R = \{cart_a, cart_b\}$	$S = \{alice, bob\}$ $SA =$ $\{nbitems, totalprice, role\}$ $R = \{cart_a, cart_b\}$ $A = \{add, remove, checkOut\}$
Payment Com-ponent	$S = \{alice, bob\}$ $SA = \{badpayer\}$ $R = \{account_a, account_b\}$ $RA = \{balance\}$	$S = \{alice, bob\}$ $SA = \{badpayer\}$ $R = \{account_a, account_b\}$ $RA = \{balance\}$ $A = \{pay\}$

Table 2
The Security Interfaces for the Example.

Audit Service	$audit(Event\ event)$
Authorization Service	$bool\ isAuthorized(Subject\ s,\ Action\ a,\ Resource\ r)$

a common security context which can hold, for example, information about the usage history of a principal. Another application is policy distribution: a centrally specified policy can be decomposed and distributed to decision points that are nearer to the resources that need to be protected.

An important assumption in the preceding discussion of the SSB is that it is deployed within a single trust domain: all components in the example reside in the same company or division and the security requirements originate from a single authority. Relaxing this assumption complicates the SSB, but also reveals some

interesting problems that can be tackled more easily when a SSB is provided. Suppose that we have a single security domain consisting of many subdomains. In such a case, the flow of security information needs to be controlled very tightly. Guaranteeing this without an SSB would be very cumbersome. However, when applications within these domains would be connected by a SSB, they could explicitly advertise the information flow policies and rely on the SSB for enforcing them in a uniform way.

5 Related Work

We are not aware of any work that proposes to use a global platform for security services. However, research does exist that links various security services to each other. Foley [8] introduces a framework that allows the sharing of access control policies between different middleware platforms. All policies can be encoded in a format that is based on Keynote credentials. The framework supports configuration, comprehension, migration, maintenance and decentralization. McDaniel [14] proposes a flexible security enforcement architecture for a group communication system called Antigone. The architecture enforces low level *session policies* that specify the security properties of a group session. Flexibility is achieved by using an event bus for communication between the API and the security mechanisms.

Several authors have explored the enforcement of advanced security policies that can take into account more information than the typical user/action/resource attributes. For instance, the dimension of time can be included and decisions can be made depending on previous events [20,2] or future events [12,9]. In the field of access control, several authors have proposed ways of representing, using and obtaining application-level information for use in access control policies [3,24]. These advanced policy enforcement mechanisms work well in local and homogeneous environments, but because of the lack of a uniform communication channel, it is hard to apply them in a distributed and heterogeneous setting. The work that perhaps comes closest to the SSB is Tivoli Access Manager [13], but this approach only considers access control and as such it does not address the problem of generically binding the applications with security services.

The security mechanisms within some platforms are architected with flexibility in mind in the sense that third parties can develop pluggable modules that extend the security functionality. For instance, the Java Authentication and Authorization Service (JAAS) [4] and the Java Authorization Contract for Containers (JACC) [21] allow customization with new authentication mechanisms and authorization engines respectively. The SSB can be seen as a generalization of these approaches that is inherently distributed.

Existing ways of sharing security information are mostly found in middleware protocols such as IIOP, .NET Remoting or SOAP. When propagation at the protocol-level is not possible, alternative solutions are needed. In the literature, different approaches exist for attaching security metadata to an execution context at a lower level than the protocols. Stateful Distributed Interposition (SDI) [19] and Cause-

way [5] provide system-level architectures for automatically propagating metadata between different distributed processes. These systems use an instrumented kernel and copy metadata at transfer points in the call flow of the application. The difference with our work is that we envision a more generic form of information sharing: instead of only pushing security attributes along with the application flow, like these systems do, the SSB supports more types of information and is based on a pull model where the security services request the information themselves. However, these systems can be useful instruments for implementing certain parts of a SSB, especially keeping track of sessions.

6 Conclusion

In this paper we have motivated the Security Bus concept as a way of interconnecting security services and we have illustrated how to apply this idea to the enforcement of a set of policies that span multiple distributed applications. This work discusses the general concept of an SSB rather than a fully worked-out architecture. We are currently designing the first version of our architecture in detail and we aim to validate it in a prototype.

References

[1] Internet 2. The shibboleth project. http://shibboleth.internet2.edu/.

[2] M. Abadi and C. Fournet. Access control based on execution history. *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 107–121, 2003.

[3] K. Beznosov. Object security attributes: Enabling application-specific access control in middleware. *Proceedings of the 4th International Symposium on Distributed Objects & Applications (DOA)*, 2002.

[4] Lai C., L. Gong, L. Koved, A. Nadalin, and R. Schemers. User authentication and authorization in the java platform. In *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999. http://java.sun.com/products/jaas/.

[5] K. Chanda, K. Elmeleegy, A. Cox, and W. Zwaenepoel. Causeway: Support for controlling and analyzing the execution of multi-tier applications. In *Middleware*, volume 3790 of *Lecture Notes in Computer Science*, pages 42–59. Springer, 2005.

[6] B. De Win. *Engineering Application-Level Security through Aspect-Oriented Software Development*. PhD thesis, Katholieke Universiteit Leuven, 2004.

[7] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[8] S. Foley, T. Quillinan, M. O'Connor, B. Mulcahy, and J. Morrison. A framework for heterogeneous middleware security. *Parallel and Distributed Processing Symposium*, 2004.

[9] P. Gama and P. Ferreira. Obligation policies: an enforcement platform. *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 203–212, 2005.

[10] Object Management Group. CORBA security service spcification, version 1.8. http://www.omg.org/cgi-bin/apps/doc?formal/02-03-11.pdf, 2002.

[11] Object Management Group. Common object request broker architecture specification, version 3.0.2. http://www.omg.org/cgi-bin/apps/doc?formal/04-03-01.pdf, 2004.

[12] K. Irwin, T. Yu, and W.H. Winsborough. On the modeling and analysis of obligations. *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 134–143, 2006.

[13] G. Karjoth. Access control with IBM tivoli access manager. *ACM Transactions on Information and System Security*, 6(2):232–257, 2003.

[14] P. McDaniel and A. Prakash. A flexible architecture for security policy enforcement. *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, 2, 2003.

[15] B.C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, 1994.

[16] OASIS. Security Assertion Markup Language Specification, Version 1.1, 2003.

[17] OASIS. Web Services Security: SOAP Message Security, Version 1.0, 2004.

[18] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, December 2005.

[19] J. Reumann and K.G. Shin. Stateful distributed interposition. *ACM Transactions on Computer Systems*, 22(1):1–48, 2004.

[20] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[21] Sun Microsystems. Java authorization contract for containers (JACC) version 1.0. http://java.sun.com/j2ee/javaacc/index.html, November 2003.

[22] Inc. Sun Microsystems. Java enterprise edition. http://java.sun.com/javaee.

[23] M. Thompson. Akenti: Distributed access control. http://dsd.lbl.gov/Akenti/.

[24] T. Verhanneman, F. Piessens, B. De Win, and W. Joosen. Uniform application-level access control enforcement of organizationwide policies. *Proceedings of the 21st Annual Computer Security Applications Conference*, pages 431–440, 2005.

[25] W3C. Simple object access protocol, version 1.2 recommendation. http://www.w3.org/TR/soap/.