

Towards Separation of Duties for Services

Christopher Giblin¹ and Satoshi Hada²

¹ *IBM Zurich Research Laboratory, Switzerland*

² *IBM Tokyo Research Laboratory, Japan*

Abstract. Separation of Duties is an essential security control for managing the integrity of information technology systems and human processes. Due in part to recent emphasis on regulatory compliance and corporate governance, Separation of Duties has taken on fresh importance. However, support for specifying and enforcing Separation of Duties constraints remains a missing feature for many of today’s access control systems, sometimes leading implementers to embed such constraints in application logic. We present the Separation of Duties and Entitlements Analyzer, a system for defining Separation of Duties constraints across multiple systems through a simple constraint model and use of standard XACML policies. The ability to define meta-policies across systems in a common language is a viable approach for managing certain integrity concerns in a services environment.

1 Introduction

Environments organized according to the principles of service-oriented architecture (SOA) offer standardized interfaces to loosely-coupled computing capabilities inherent to an organization’s mission. An important goal of service-oriented architectures is the composition of these services into new or augmented functions. While abstraction and composition have obvious benefits, organizations must at the same time assure that security goals, such as auditable authorization, can be managed with equal flexibility and clarity.

Separation of Duties is an important security control which relates to authorization of business tasks. Interesting challenges arise in dealing with Separation of Duties since often it is required to constrain specific combinations of roles and task executions, a capability not necessarily satisfied by many of today’s platform security mechanisms. As Separation of Duties has historically dealt with human actions, the types of services most immediately relevant are human-based services as implemented by the IBM ®Human Task Manager [8], as proposed by the BPEL4People/WS-HumanTask specification [6], and potentially as implemented in RESTful services.

The Separation of Duties and Entitlements Analyzer project has addressed implementing Separation of Duties authorization controls across multiple systems. At the project’s core are the abstractions of constraint, based on simple conflict pairs, and authorization policy, based on the standard for authorization policy, XACML [5]. Since the constraints are a form of policy about authorization policy, they are sometimes referred to as *meta-policies*. The project has not

examined web services but rather web applications. Nevertheless, building blocks have been developed which can be extended towards supporting services.

The remainder of this paper discusses the SoD and Entitlements Analyzer and its potential application to services. Section 2 provides background on the concept of Separation of Duties. Section 3 describes the architecture and key features of the analyzer. Section 4 demonstrates the use SoD on an example application scenario. Section 5 discusses experiences and insights gained from the project. Section 6 presents conclusions and future work.

2 Separation of Duties

Separation of Duties (SoD) is a traditional security control predating information systems and is particularly well-known in financial accounting. Some of its more common forms are known by popular monikers such as "four-eye-principle" or "two-man rule". The essential aim of SoD is to recognize potential conflicts of interest in activities which could result in errors or fraud. Where potential conflicts of interest arise, activities are structured such that different individuals perform specific steps of the overall activity. As a simple example, in a purchasing process, the person who requests a purchase usually is not the same person who approves purchases. Distributing responsibilities reduces the impact that a single individual can have, requiring collusion to perpetuate a fraud.

We distinguish here between two kinds of SoD enforcement mechanisms: static separation of duty and dynamic separation of duty. One speaks of static separation of duty, abbreviated SSoD, if policies are enforced during design time. This is usually done by constraining the assignment of access rights or role assignments. Enforcement which depends on information available only at runtime is called dynamic separation of duty, for short DSoD. An example of DSoD is a system ensuring that a user attempting to execute a task, such as within a workflow instance, does not violate SoD constraints given the history of users who have previously executed tasks in that workflow instance. Most authors cite Saltzer and Schroeder [1] as the first authors who mentioned the concept of SoD, at that time under the name separation of privileges. Alternative names for SSoD and DSoD are strong exclusion and weak exclusion [1].

A control is a planned measure or countermeasure designed to mitigate a risk or assure the integrity of activities in pursuit of an organization's goals [9]. With specific regards to computing technology, security controls are measures taken "to protect the confidentiality, integrity, and availability of the system and its information" [7]. Security controls are commonly divided into two broad categories: detective and preventative. Preventative controls prohibit an action from happening while detective controls identify an action during or after the fact. A control can be constrained in its effectiveness due to system, financial or environmental limitations, prompting the deployment of one or more additional controls to compensate the initial control weakness. Such additional controls, whether preventative or detective, are termed compensating controls.

Separation of Duties controls can be preventative or detective. A component prohibiting administrators from assigning conflicting roles and permissions is an example of a preventative SoD control. However, some access control systems do not provide a generic means for enforcing SoD constraints. In such cases, log file analysis to identify SoD violations post facto represents a detective control. Managing SoD in complex environments often requires the combination of preventative and detective controls.

3 SoD and Entitlements Analyzer

The Separation of Duties and Entitlements Analyzer is part of an ongoing project at IBM® Research focusing on enterprise authorization services. The project has produced a framework built around the core concepts of system, constraint and analyzer. A web-based console is the user interface for authoring constraints and interacting with analysis and reporting components. Infrastructure components provide a repository for storing system information and another repository for storing SoD constraints. The console and framework are implemented together as a Java® Enterprise Edition application. Figure 1 illustrates the main architectural components.

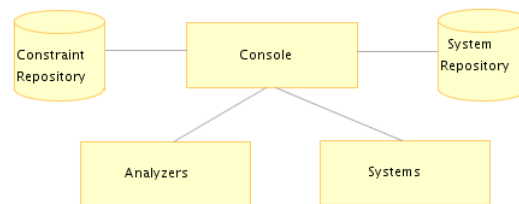


Fig. 1. Architectural Overview

An instance of this work is publicly available at the IBM alphaWorks site [2]. This version supports the specification and evaluation of SoD constraints for IBM's Tivoli® Access Manager and Tivoli Identity Manager™ products.

3.1 Constraints

SoD constraints have a type, a unique identifier, version information, an associated system, and a set of conflicts. Each conflict pair represents a mutual exclusion over a pair of entities. Entities can have an optional system identifier which permits defining conflicts spanning two systems. Otherwise, entities are assumed to belong to the system initially defined in the constraint. SoD constraints can

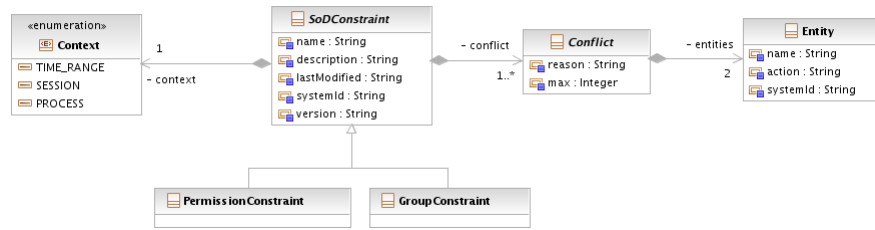


Fig. 2. SoDConstraint Model

also be defined for a specific context such as a session, period of time, or process instance. Figure 2 presents a UML class diagram of the constraint model.

An optional cardinality can be defined over the set of conflicts indicating the number of conflicts which must hold for a violation. For example, given a constraint with four conflict pairs and a cardinality of two, the constraint is violated if two or more of the conflicts exist during evaluation.

The framework currently recognizes two constraint types: group constraints and permission constraints. In the case of a group constraint, the conflicting entities are groups or roles which are mutually exclusive, that is, they contain no common members. For example, a constraint could state no user should have both an accounting and a purchasing role. Permission constraints express mutual exclusion of access permissions where a permission is defined as a resource/action pair. A permission constraint could state that no user can have, for instance, both the approve and pay permissions on purchase orders.

The console provides two alternative forms for authoring SoD constraints: matrix and list. The matrix representation is suitable for defining conflicts pairwise while the list is convenient for defining a group of conflicting entities. While the matrix form is more expressive, the list is more convenient for simple constraints. In both cases, constraints are stored as a set of conflict pairs.

3.2 Analyzers

Analyzers perform analytic functions relating to authorization. Those analyzers evaluating constraints on systems or policy include static SoD role and permission analyzers, a runtime DSoD enforcer and a log analyzer. Two other analyzers operate on policy without constraints: the entitlements analyzer and accessor analyzer. The entitlements analyzer reports on the resources accessible to a given user while the accessor analyzer reports on all users who can access a given resource. Figure 3 illustrates the analyzers in relation to their respective artifacts and constraints. The diagram also shows how policy analysis is actually performed on XACML translated from the authorization systems of specific systems.

All analysis on authorization policy is performed on XACML [5] representations with the corresponding user registry defining the users, group assignments

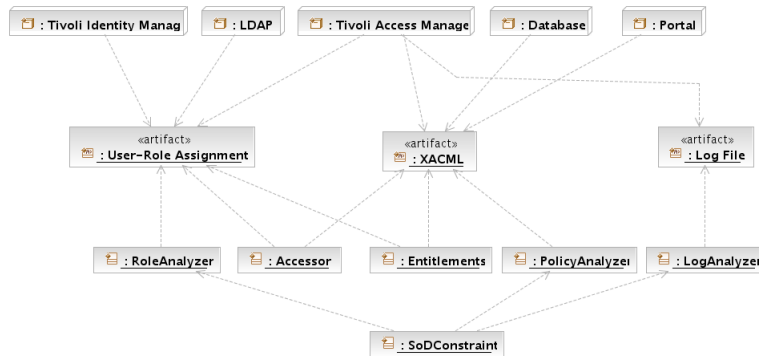


Fig. 3. Analyzers and respective artifacts

and role assignments. In order for a system to be supported by the analysis platform, an XACML translation of the system’s access control policy is required. For example, when the policy analyzer evaluates a permission constraint over Tivoli Access Manager, the native policy is retrieved from the Tivoli Access Manager installation and translated into XACML. The constraint is then evaluated against the equivalent XACML policy. The details of the Access Manager-XACML policy translation are presented in [3]. By performing policy analysis on the XACML standard, integrating other policy sources for analysis, such as the database and portal shown in Figure 3, is a matter of providing the corresponding XACML translation.

3.3 XACML Policy Analysis

Our approach to evaluate conflicting pairs of permissions on a given XACML policy occurs in two steps. **Step 1:** For each permission, we compute the set of subjects who has the permission. **Step 2:** We check whether the two subject sets contain common subjects. If this is the case, the common subjects are in conflict and represent a SoD violation.

Our algorithm for Step 1 works as follows. An XACML policy is a set of rules. Each rule has a rule effect (i.e., permit or deny) and a target which represents a set of tuples (*subject, resource, action*) applicable to the rule. In other words, given a rule and a permission (a pair of resource and action), we can compute the set of subjects which do or do not have the permission. We use the following general data structure to represent such a set of subjects: (*Permit, Deny, Type*), where *Permit* and *Deny* are a set of subjects. The possible values of *Type* are “PermitAny”, “DenyAny”, “NonAny”. “PermitAny” means that all subjects excluding the subjects in *Deny* have the permission. Similarly, “DenyAny” means that all subjects excluding the subjects in *Permit* does not have the permission. “NonAny” means that the subjects in *Permit* and *Deny* does and does not have the permission, respectively. Using this data structure, we describe

the algorithm as follows: For simplicity, we assume that (1) no rule uses conditions, (2) the policy has no target, and (3) the rule combining algorithm is “first-applicable”.

1. Input: an XACML policy and a permission.
2. Initialize ($Permit = \emptyset, Deny = \emptyset, Type = \text{“NonAny”}$)
3. For each rule, do the following:
 - (a) Compute the set S of subjects who does or does not have the permission.
 - (b) If S is AnySubject, then do the following and break the for loop:
 - i. If the rule effect is permit, then set $Type = \text{“PermitAny”}$.
 - ii. Otherwise, set $Type = \text{“DenyAny”}$.
 - (c) Otherwise, do the following:
 - i. If the rule effect is permit, then remove the subjects in $Deny$ from S and add the remaining subjects to $Permit$.
 - ii. Otherwise, remove the subjects in $Permit$ from S and add the remaining subjects to $Deny$.
4. Output: ($Permit, Deny, Type$)

Extension of the algorithm to the other XACML policy combination algorithms is straightforward.

4 Sample Application

A sample purchase-order processing application is described in this section to demonstrate the SoD analyzers. The sample application is implemented according to the Representational State Transfer (REST) architectural style [4]. The use of REST in this example does not imply advocacy for a particular architectural style, rather is one scenario illustrating the application of SoD constraints.

4.1 Resources

In this sample application, the HTTP POST method is used to create resources; PUT updates existing resources; GET retrieves resources. As REST applications do not maintain stateful communications, there is no notion of application session. Roles are not hierarchical.

The processing steps, required roles, methods and URIs in the purchasing process are summarized in the table below:

| Task | Role | HTTP Method | URI |
|-----------------------|------------|-------------|-------------------------|
| Request purchase | Employee | POST, GET | /purchase/request |
| Approve Purchase | Manager | PUT | /purchase/request |
| Create Purchase Order | Purchasing | POST | /purchase/order |
| Receive shipment | Inventory | POST, GET | /purchase/order/invoice |
| Approve payment | Accounting | POST, GET | /purchase/order/payment |
| Enact payment | Accounting | PUT, GET | /purchase/order/payment |

4.2 Runtime environment

As shown in Figure 4, the runtime environment consists of a server running the purchasing application; Tivoli Access Manager, an authorization server; WebSEAL, a reverse proxy which enforces policy defined in Access Manager; the Separation of Duties and Entitlements Analyzer for defining and analyzing SoD constraints. In XACML terminology, the authorization server is a Policy Decision Point (PDP). The reverse proxy downloads policy from the authorization server and functions both as a PDP and a Policy Enforcement Point (PEP).

Log files are produced by Access Manager, the reverse proxy and the HTTP-based purchasing application, all of which can be processed by the log analyzer. Somewhat uncharacteristic of REST applications is the authentication session maintained by the reverse proxy. However, this session is not an application session and is not maintained or perceived by the purchasing application. Rather it is a configurable security construct supported by Access Manager. An interesting capability of the WebSEAL reverse proxy is the ability to extend runtime authorization decisions over a programmatic interface, and so add SoD constraint enforcement. In the absence of the reverse proxy, this extension could feasibly be implemented in the front-end web server of the purchasing application.

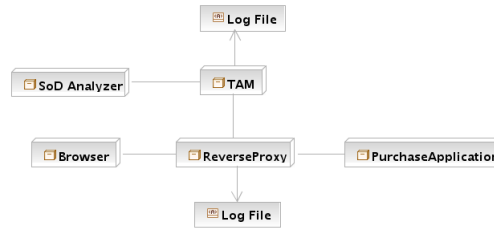


Fig. 4. Runtime environment

4.3 Static SoD Policies

The purchase order, shipment reception and payment tasks are to be kept strictly distinct. This is accomplished through both group and permission SoD constraints. A single group constraint defines the required roles as mutually exclusive. Likewise, a permission constraint defines those permissions required to execute these tasks as mutually exclusive. For these group and permission constraints, the list representation is the more intuitive representation and is shown in Figure 5.

4.4 Dynamic SoD Policies

Members of the Accounting role have permission to approve and enact payments, but should not approve and enact payments for the same purchase order. In the

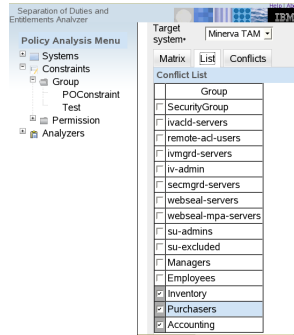


Fig. 5. Defining mutually exclusive roles in a list

example system, the runtime SoD component will enforce this constraint at the session level. That is, no user can approve and enact payment in the same session. Thus this constraint is defined as a conflict between approve and pay permissions with context set to session. As a compensating control, dynamic SoD log analysis on the web server logs is performed.

5 Discussion

The SoD platform presented here could feasibly be used in a services environment. Many aspects would remain unchanged, namely the accessing of user registries and authorization policies for performing analytics against meta-policies. Extensions would involve developing translations of service authorization constraints to XACML. As service descriptions are not annotated with authorization policy, this usually will require deploying an adapter on systems where authorization over service executions is performed, thus "behind the scenes" of a service. BPEL4People, on the other hand, defines user/group assignments and activity permissions which could map naturally into the same concepts in our platform. The SoD platform could be used to verify consistent SoD across multiple processes.

Enforcing SoD constraints during runtime as an extension to authorization poses both performance and resource challenges. Dynamic SoD constraints are stateful, monitoring long running instances such as sessions, processes and objects. For decision points with high volume, the amount of active state can become quite large. Therefore DSoD constraints require a persistent store for managing memory and surviving system restarts. This could mean that database lookups would become unavoidable during decision computation, thereby increasing decision response time.

Where back-end services support user applications, SoD can be most effectively enforced at the service if the identity of the requesting user is propagated

to the service. This is contrary to current practice where applications often use a dedicated user in accessing back-end services and databases.

Meta-policies can play an important role in maintaining integrity over services for governance objectives. As such, SoD policies do not seem suitable as WS-Policy assertions. SoD occurs at a level higher, taking as input user registries, roles, authorization policy, and in a services environment, WS-Policy assertions and BPEL4People activities and expressions. While many SoD use cases can be captured in conflict pairs, more sophisticated scenarios do exist which require a more powerful constraint language. As environments continue to become larger and more complex, while simultaneously meeting growing requirements for transparency and auditability, the need for advanced authorization policy analytics becomes inevitable.

6 Conclusions and Future Work

We have presented a system of services for the analysis of authorization policies across multiple platforms and applications, with a particular focus on Separation of Duties constraints. Centrally authored constraints are processed by components called analyzers which implement preventive and detective controls at various points in the runtime lifecycle. All policy-based analysis is performed on authorization policies represented in the XACML standard. A sample scenario based on REST services was presented to demonstrate a number of Separation of Duties controls.

Future work includes extending the systems supported for policy analysis by defining further translations of native authorization policy to XACML. Particularly business process models involving human tasks and separation of duties are of interest such as the BPEL4People/WS-HumanTask [6] proposal. In addition, work continues on algorithms for analyzing very large user and role databases.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Program (FP/2007-2013) under grant agreement no. 216917.

The work described in this paper was conceived and developed together with our team colleagues, Günter Karjoth, Andreas Schade and Yukihiro Sohda. We extend our gratitude to Michiharu Kudo and Andreas Wespi for their valuable contributions. Finally, we thank Samuel Burri and Samuel Mueller for their stimulating discussions.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.

References

1. J. Saltzer and M. Schroeder: The Protection of Information in Computer Systems. Proceedings of the IEEE, 63():1278-1308, September 1975
2. C. Giblin, S. Hada, G. Karjoth, A. Schade, Y. Sodha and E. Van Herreweghen: Separation of Duties and Entitlement Analyzer for Tivoli Access Manager. IBM alphaWorks, 2008. <http://www.alphaworks.ibm.com/tech/sod4tam>
3. G. Karjoth, A. Schade and E. Van Herreweghen: Implementing ACL-based Policies in XACML. To appear in 24th Annual Computer Security Applications Conference, December 8-12, 2008, Anaheim, California.
4. R. Fielding: Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000.
5. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. OASIS Committee Specification (T. Moses, editor). http://www.oasis-open.org/committees/documents.php?wg_abbrev=xacml.
6. OASIS: WS-BPEL for People (BPEL4People). Proposed technical OASIS committee, 2008. <http://xml.coverpages.org/bpel4people.html>
7. National Institute of Standards and Technology, U.S. Department of Commerce: Recommended Security Controls for Federal Information Systems, NIST Special Publication 800-53, December, 2007.
8. M. Kloppmann, S. Liesche, G. Pfau, and M. Stockton: Human-based Web services, IBM Developerworks, October, 2005. <http://www.ibm.com/developerworks/library/ws-soa-progmodel8/index.html>
9. The Institute of Internal Auditors: Glossary. Viewed 2008. <http://www.theiia.org/guidance/standards-and-practices/professional-practices-framework/standards/standards-for-the-professional-practice-of-internal-auditing/?search=glossary>