White Paper

Enforcing Segregation of Duties (SoD)

A real problem growing out of proportion with poorly designed access controls
Enforcing segregation of duties

In recent years many IT organizations have struggled to identify potential Segregation of Duties (SoD) violations within their IT systems.

A violation of this kind occurs if a user is given permissions which combined can be used to subvert a business critical process or in other ways cause harm in breach of corporate policies. A common example is the combination of permissions to manipulate vendor master data and permissions to approve invoice payments to a vendor. This combination leads to risk exposure since a user fraudulently could register a vendor and then approve fake invoices for his own benefit.

The effects of a SoD analysis often leads to costly remediation efforts since conflict resolution may require altering role definitions and related business processes. Alternatively, auditors may agree to accept mediating controls, meaning the risk is accepted but managed through manual controls at regular intervals to verify that permissions in breach of corporate policies are not abused.

An inherent RBAC weakness?

Interestingly, the need to handle SoD conflicts seems to have dawned quite suddenly upon the industry. The annual Hype Cycle report for Identity and Access Management published by Gartner in 2005 did not mention SoD at all whereas the 2006 edition placed “SoD for ERP” as being already past the peak. By 2007 “SoD for ERP” was in the Trough of Disillusionment and through 2008 and 2009 it matured, Climbing the Slope in Gartner terms.

This rapid hype career reflects real costs and money spent by customers. “The market [Segregation of Duties Controls Within ERP] was about $150 million in 2006 and is forecast to grow to $200 million in 2007 and to more than $250 million by 2008”, said another report which updated in 2008 predicted $300 million for 2009. The actual outcome of these forecasts is difficult to verify since many of the vendors specializing in “SoD for ERP” later turned up in reports labeled “Yesterdays cool vendors” – their products were typically merged into the IAM or ERP suites of larger vendors.

However, customers continue to pay for tools, for audits in which they are used, and for remediation of continuously reoccurring SoD issues.

One may wonder why SoD risk treatment so rapidly became top priority and why customers invested so heavily to get them fixed. Were the risks always there although ignored until Sarbanes Oxley and other regulations made them top priority? Or is the problem simply proportionate to the successful market penetration by ERP vendors?

If the ERP system could be configured never to allow approval of payments to those specific vendors whose master data have been created or edited by the approving user, then an enforced mitigating control would be built into the system and a SoD conflict would not occur. But ERP systems mostly utilize Role Based Access Control (RBAC) models for access control. With RBAC, business rules are cast into roles leading to authorization decisions that either say “yes, you may because it’s in your role” or “no, you may not because it is not in your role”.

What we need to express, however, is a conditional rule that says “yes, you may do A, provided you did not previously do B”.
A rule of this kind expresses conditions based on the context in which access is requested. And this, by necessity, is typical for most situations in which SoD needs to be enforced. SoD rules are never derived from absolute truths. Maintaining a sound division of job tasks between staff members is a means to handle business risks. And the business risk emerges out of the context of the individual users’ actions. The RBAC model, however, is not well suited to handle contextual conditions. Quite the opposite, it assumes we can make predefined decisions based on preconfigured roles regardless of the specific context in which a user later chooses to use this role.

The “SoD in ERP” hype is therefore in reality an effect of poorly designed access controls lacking the ability to dynamically adapt to conditional business rules.

The ABAC approach

With Attribute Based Access Control (ABAC), by contrast, the objective is to express business rules rather than to label a set of static user permissions with a role name. In ABAC, attributes describing the subject, the action, the resource and the context of an access request are used to define hierarchies of rules.

Sticking to our example, the following attributes would be of interest in a rule definition:

1) Describing the subject (the user approving a payment) a list of identifiers pointing to vendor records manipulated by the user should be attached, an attribute we could call user.vendorlist.
2) The action would be approve.
3) The resource would be a payment – a record or transaction in the system – with a recipient attribute identifying the vendor to whom money will be sent.

A policy can state something like “if action=approve, resource=payment and resource.recipient not in user.vendorlist, then permit else deny”, a rule easily defined using XACML, the policy language used in ABAC based authorization.

Furthermore, let’s assume the rule needs to be extended, for instance to enforce constraints on the amounts a user can approve depending on role, level or delegation principles. With XACML, new rules can easily add further conditions to extend our existing policy – where the RBAC model instead would risk role explosion problems in order to capture new requirements.

Finally, ABAC rules can be managed centrally and enforced in multiple applications which is practical if integrated business applications are used to support one business process for which segregation of duties needs to be enforced – again something that is almost impossible to achieve with RBAC due to role explosion effects.

Thus, ABAC assists in the implementation and enforcement of access controls derived from actual business rules, including rules mandating dual controls or segregation of duties.
Segregation of Duties – RBAC vs. ABAC

Segregation of duties to minimize risk exposure remains important regardless of what type of authorization technique is used. Enforcing SoD is therefore equally important with ABAC as with RBAC. But our examples reveal some important differences as illustrated below:

With RBAC the violation occurs when permissions contained in two different roles are in conflict. Users having both Role 1 and 2 in the above figure are affected whereas those who have either Role 1 or Role 2 in combination with other roles are not. Altering Role 1 and/or 2 therefore often leads to cascading role redefinitions and/or role explosions. Unaffected users in group 1 and 3 may be impacted by measures taken to handle a conflict within group 2. The role concept may thus expand the effects of SoD resolution into “innocent” parts of the user population.

The role concept also expands the problem across unaffected permissions. Although only individual permissions within Role 1 and Role 2 are in conflict they “contaminate” their respective roles. With SoD enforced on these roles, none of the permissions they bundle can be combined even if most combinations do not represent a business risk.

By expanding the SoD conflict in either or both dimensions – affecting “innocent” users and/or “innocent” permissions – RBAC generates considerable administrative costs.

With ABAC, the failure to capture a business requirement for SoD would instead be the result of imprecise or erroneous rule definitions. A rule definition saying “Yes, you may approve payments, provided you have a driver’s license”, as illustrated above, would not adequately capture the business risk we want to avoid. Furthermore, this type of rule error may also affect an “innocent” user. A legitimate transaction could be blocked because the user does not have a driver’s license, which from a business risk perspective is rather irrelevant.

However, provided we have efficient means to detect and avoid rule errors as well as efficient means to verify that all of our actual business rules have been properly captured in the policies of our authorization system, problem resolution is simple and isolated to our inadequate or missing rules.
Conclusions:

- SoD requirements on authorization require the ability to provide conditional decisions based on contextual data. The RBAC model as opposed to the ABAC model is not context-aware and thus not well suited to handle SoD requirements.

- In RBAC SoD conflicts expand across a larger set of users and permission than those initially affected. Problem resolution may require changes beyond the immediate scope of the identified problem. Problem resolution may be time consuming, laborious and expensive.

- In ABAC an unmanaged business requirement for SoD is the effect of rule design failures. Altering failing rules fixes the problem. Provided rule design flaws are detected in a timely fashion, problem resolution is fast, simple and inexpensive.

Policy Governance for ABAC

As seen by the above discussion, quality assurance in policy modeling becomes crucial when you use Attribute Based Access Control (ABAC). We need to verify that business rules are being adequately captured in policies and that all policies actually have been properly implemented.

Axiomatics product suite therefore offers a broad range of features to assist in the creation, evaluation and auditing of authorization policies.

- Axiomatics Policy Server comes with an IDE for policy development, where different types of policy requests can be simulated and analyzed with a debugger which will reveal exactly how policy resolution resolves to a "permit" or "deny" based on existing policies.

- Axiomatics Policy Auditor is a separate auditing tool that can be used to ask questions such as "who has access to resource R and under what conditions"?

You can find more information about these products on the Axiomatics web site’s Product section and in the following white papers:

- Axiomatics Policy Server White Paper
- Axiomatics Policy Auditor White Paper