# From MDM to DB2: A Case Study of Security Enforcement Migration

Nikolay Yakovets[1], Jarek Gryz[1], Stephanie Hazlewood[2], and Paul van Run[2]

[1] Department of Computer Science and Engineering, York University, Canada
Centre for Advanced Studies, IBM Canada
{hush,jarek}@cse.yorku.ca
[2] IBM Canada
{stephanie,pvanrun}@ca.ibm.com

**Abstract.** This work presents a case study of a migration of attribute-based access control enforcement from the application to the database tier. The proposed migration aims to improve the security and simplify the audit of the enterprise system by enforcing information protection principles of the least privileges and the least common mechanism. We explore the challenges of such migration and implement it in an industrial setting in a context of master data management where data security, privacy and audit are subject to regulatory compliance. Based on our implementation, we propose a general, standards-driven migration methodology.

**Keywords:** Master Data Management, Enterprise Security, Attribute-Based Access Control, Database Security, XACML, DB2.

## 1 Introduction

Today's enterprise data is complex and heterogeneous, and users who access it are diverse and belong to multiple domains. The access control models have evolved to fit these requirements. Traditional discretionary (DAC, [1]) and mandatory (MAC, [2]) access control models have been replaced by role-based access control (RBAC, [3]) and, more recently, attribute-based access control (ABAC, [4]). Unlike DAC, MAC and RBAC, the ABAC model defines permissions based on just about any security relevant characteristics of requesters, actions, resources, and environment, known as attributes.

The lack of native ABAC support by conventional DBMS has motivated many enterprise developers to implement access control checks at the application tier, bypassing the native database access control. In such architecture, the database connection is established on behalf of the application and is able to access the entire database, while application program logic itself is used to limit the privileges of the end-user. While this approach allows enforcement of more complex and flexible attribute-based policies, it also introduces several problems. First,

N. Cuppens-Boulahia et al. (Eds.): DBSec 2012, LNCS 7371, pp. 207–222, 2012.

because the database connection is at an elevated privilege, the enterprise system is prone to *privilege escalation attacks* [5], such as SQL injection [6]. Second, since the database is not aware of the identity of an end-user issuing the request, it may be difficult for application developers to provide the *end-to-end audit trail* which is essential to comply with regulations such as SOX [7], PCI [8], and HIPAA [9].

In this paper, we describe how to perform a migration of an existing enterprise security enforcement from the application tier to the database tier. In other words, we would like to be able to securely enforce the enterprise security policies at the database, while retaining the flexibility of their management at the application. We see several benefits of such database-level enterprise security enforcement. First, it helps protecting the enterprise system from privilege escalation attacks by complying with information protection principles of the least privileges and the least common mechanism [10]. Further, the placement of security controls on the database allows for full end-to-end audit trail as the identity of an end-user is not hidden behind a common authorization ID of an application layer.

We consider our approach in an industrial setting in a context of master data management (MDM) [11]. A wide variety of MDM solutions are available on the market today from major vendors such as IBM, Oracle, Microsoft and SAP. Such systems aim to provide a centralized environment for maintaining the *master* data about customers, suppliers, products and services. Master data plays an important role in both operational and analytical aspects of the organization operation. However, it is also a tempting target for attackers as it is captured by combining, cleansing and enriching highly confidential data across various data sources in the organization. Further, master data also contains personally identifiable information, making MDM solutions subject to regulatory compliance. Since the proposed migration aims to improve the security and simplify the compliance of the enterprise system, we believe it may be useful, in the first place, in the MDM domain.

In collaboration with IBM Centre for Advanced Studies, we used IBM InfoSphere Master Data Management Server (in short, MDM Server) as an implementation platform. MDM Server is a large scale industrial-grade MDM solution that enforces attribute-based security policies at the application tier while storing the master data in the relational database. This enabled us to implement the proposed migration in MDM Server to prove the feasibility and practical usefulness of our approach.

Lastly, we believe that our MDM Server migration experience may be helpful in similar migration projects in a variety of enterprise applications outside of MDM. To facilitate the implementation of such projects, we propose a general migration methodology that is standards-driven and vendor-independent.

The rest of the paper is organized as follows. Section 2 describes the challenges of the proposed migration, and Section 3 describes our industrial implementation. Section 4 discusses the lessons we learned, and Section 5 presents a general migration methodology. Section 6 discusses some of the related works, and Section 7 concludes.

## 2 Challenges

The proposed migration aims to place the security enforcement on the database, while keeping the security administration at the application. This comes with several challenges as both end-user identities and security policies need to be timely and efficiently *propagated* from the application tier down to the database. The following subsections provide more detail.

### 2.1 Identity Propagation

Request processing often involves the propagation of end-user's identity from one enforcement point to another within the enterprise environment. For example, consider a case where end-user is authenticated at a client tier. Then the identity is sent to an application tier for its own authorization and auditing. The application in turn generates the requests to query a back-end database potentially using yet another identity that is suitable to access a relational database on end-user's behalf.

The scenarios of such identity propagation differ by a degree of knowledge that the relational database has about the end-users. For example, an end-user may be mapped to a *system identity* - or the identity that is used to represent the application tier. This *many-to-one* mapping greatly simplifies the deployment topology, but effectively places all of the security controls such as authorization and audit on the application tier, as the relational database has no knowledge about the identity of the end-user.

On the other hand, an end-user may be mapped to a *functional identity* which can represent user's role or group in the organization. This *many-to-many* mapping allows placing some of the security controls on the relational database for the price of complicating the deployment topology. In this case, however, the database still does not know who exactly the end-user is. Therefore full *end-to-end* audit trail is impossible which can cause problems with regulatory compliance.

Lastly, an end-user may be mapped to her corresponding identity in the database. This *one-to-one* mapping allows the database perform authorization and audit using end-user's identity. This scenario requires the most difficult deployment topology, but allows placing most of the security controls on the relational database. Since this allows for full end-to-end audit trail, it greatly simplifies regulatory compliance of the enterprise system.

The proposed migration involves shifting from many-to-one to one-to-one identity mapping during the enterprise request processing. What makes this challenging is that the complication of the deployment topology that comes with one-to-one mapping is undesirable. Moreover, the identity propagation mechanism would require an identity provisioning component responsible for keeping identities in the database consistent with identities administered at the application. An optimal solution for handling the identity propagation would need to solve the above problems.

### 2.2 Policy Propagation

Application-to-database policy propagation is challenging because the underlying security mechanisms at the application are conceptually different from those at the database. Application-level authorization engine can use either proprietary or standards-driven approach such as OASIS XACML [12] to specify, distribute and enforce *high-level* attribute-based security policies. The flexibility of such policies comes from the ability to define the security rules based on attributes of users, resources and environment. For example, a single rule might permit access for all professors in the department of computer science to files pertaining to courses offered in that department. The access decision is made during the request execution time when the rule attributes are retrieved: the authorization IDs of current CS professors and table names of currently offered CS courses.

Relational database *low-level* security enforcement mechanisms are based on the assignment of privileges to users or roles. The privileges can be granted or revoked on complete tables and views. For example, a single database authorization might permit access for professor `JohnDoe` to table `CS101`. Clearly, such discretionary database policies are less expressive than attribute-based application policies.

The proposed migration would involve linking the security mechanisms at the application and at the database. A consistent mapping should be established between the security policies, such that the database privileges conform to semantics specified by permissions in the application. Mapping of a single application policy would require the understanding how application-level users and resources identified by their attributes translate to database-level authorization IDs and tables. Mapping of multiple application policies would require the understanding of policy combining and conflict resolution algorithms that are used at the application.

It is important to realize that updates to application security policies or to user and resource attributes may alter the established policy mapping in a non-trivial way. Some of the database permissions would need to be granted, while others would need to be revoked. The solution for policy propagation would need to provide the mechanism that will efficiently maintain the consistent relationship between the database permissions and the policies administered in the application.

## 3 Implementation

In this section we show how the proposed migration can be implemented in an industrial setting in MDM Server. MDM Server is a large scale enterprise application that runs on WebSphere Application Server (WebSphere) and uses IBM DB2 relational database (DB2) to store master data. First, we describe how the identity of an end-user of MDM Server can be efficiently propagated from WebSphere down to DB2. Next, we talk about how the attribute-based MDM Server security policies can be mapped to discretionary DB2 permissions.

Finally, we show how to efficiently maintain the consistency of DB2 permissions when MDM Server policies or attributes are updated.

### 3.1 MDM Server Identity Propagation and DB2 Trusted Context

In MDM Server access control checks are implemented at the application tier, and the application itself is able to access the whole master database by using an administrative system identity. The goal of our migration is to place most of the security controls on DB2, instead of relying on MDM Server to perform the security checks. To make an access decision, the database needs to know the identity of an end-user issuing the request that is effectively "masked" by the administrative system identity of the MDM Server application. One way to solve this problem is to modify the MDM Server application logic in such a way that each database connection is established using the end-user's identity who issued the request instead of the administrative system identity. However, this approach would involve the significant modification of the MDM Server code base, which is undesirable. Another, more efficient approach is to plug in a Java Authentication and Authorization Service (JAAS) login module to propagate the end-user identity to the database server. This approach maintains the end-user identity, but does not support connection pooling that is normally leveraged by MDM Server when many-to-one administrative system identity is used.

In our implementation, we utilize *trusted connections* instead of the manual mapping or a JAAS mapping. Trusted connections support client identity propagation and can also use connection pooling to reduce the performance penalty of closing and reopening connections with a different identity. Trusted connections leverage the DB2 trusted context object. The DB2 trusted context is an object that the database administrator defines and that contains a system authorization ID and a set of trust attributes. The trust attributes identify characteristics of a connection that are required for the connection to be considered trusted. The relationship between a database connection and a trusted context is established when the connection to the DB2 server is created. After a trusted context is defined, and an initial trusted connection to the DB2 database server is made, the application server can use that database connection from a different user without a full re-authentication. The database authenticates the end-user and then verifies the user authorization to access the database prior to allowing any database requests to be processed on behalf of that user.

When software stack is set up for use of the trusted context, the authorization of the end-user requests is carried out as follows. First, end-user is authenticated to MDM Client ①, and her identity is sent over to WebSphere which is runnung MDM Server ②. WebSphere initially establishes the connection to DB2 using underpriviliged "connection-only" user ③. However, on query execution time, the user identity is switched to the end-user who executes the query ④. Therefore, the query is executed and audit trail is logged at DB2 with the identity of the end-user ⑤.

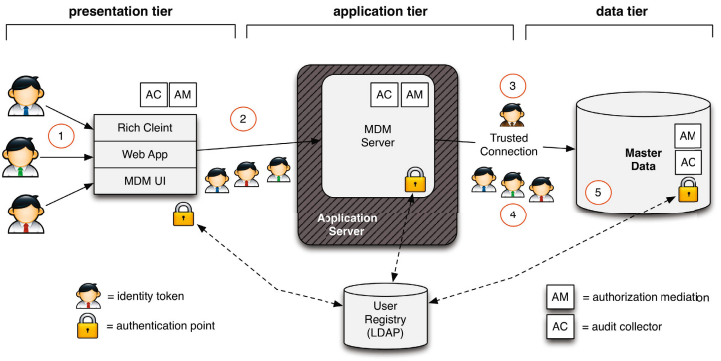


**Fig. 1.** Identity Propagation in MDM Server

To provision one-to-one identity propagation, database-level authorization IDs should be kept consistent with identity information stored in MDM Server according to the established one-to-one mapping. We solve this problem by using an LDAP [13] registry common to both MDM Server and DB2. Thus, the use of a trusted context and a centralized user registry allows us to establish an efficient one-to-one identity propagation that does not complicate the deployment topology of the enterprise system.

### 3.2 MDM Server Policy Propagation

To establish application-to-database policy propagation in MDM Server we need to provide a consistent mapping between MDM application security policies and their DB2 counterpart.

In DB2, we consider object-level and content-based authorizations that are granted as primary, secondary or public permissions. Such *discretionary* permissions are represented as access control lists (ACLs), where each object is associated with a list of subjects coupled with a set of actions they can perform on the object. The object, in this case, can be a table or a view, while the subjects are DB2 users (authorization IDs), user groups, user roles and public (all users). The access operations that subjects are allowed to invoke on objects are SQL operations that can be executed on tables: select, insert, delete, and update.

On the other hand, at the application tier, we have MDM *entitlements*, which are managed by Rules of Visibility and Persistency Entitlements components of MDM Server. In an entitlement, an *accessor*, which may be a user or a user group, is entitled to take an *action*, for example adding, updating, or viewing, on set of *business objects* or their *elements* which are identified by the *data association* resource attribute in the entitlement.

To establish a policy mapping, one should understand the semantics of MDM entitlements in the context of master data stored in the relational database.

In MDM, business objects map to database tables and elements that further refine those business objects correspond to the columns of its underpinning database tables. To enable full end-to-end audit, we establish one-to-one mapping between MDM accessors and database users and groups. Finally, MDM data actions are mapped to DB2 data manipulation operations (i.e. `update`, `select`, `insert` and `delete`).

For example, consider a pair of MDM entitlements shown in Fig. 2. First entitlement permits a particular student to view resources with attribute Data Association = {Course, Grades}. Second entitlement permits all professors updating resources attributed to Grades only. Through corresponding associated objects, COURSE database table is assigned Data Association = {Course, Grades}, and GRADES table has Data Association = {Grades}. According to our policy mapping, the first entitlement is associated with DB2 authorizations [1], [2] and [3], while [4] and [5] are associated with the second entitlement.

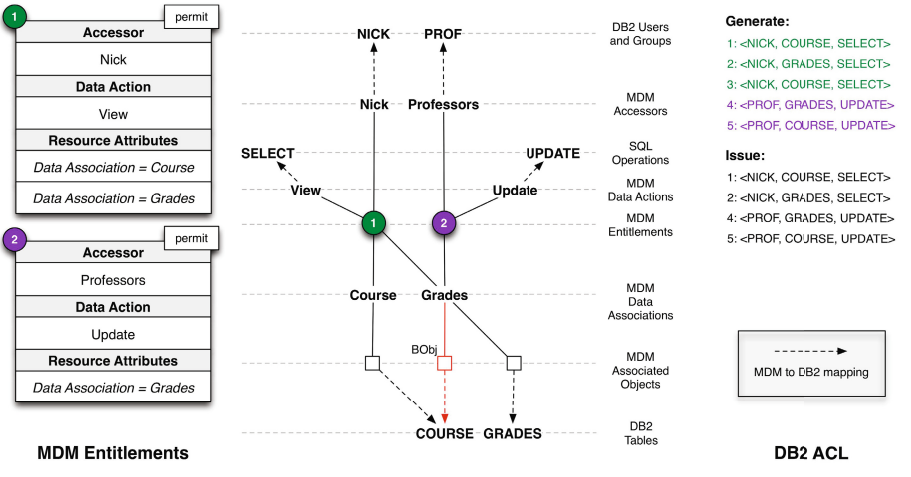


**Fig. 2.** Policy Mapping in MDM Server

To accommodate the propagation of MDM entitlements down to DB2 we modify MDM Server application logic by introducing Policy Propagation Point (PPP) component (shown in Fig. 3). Policy propagation is carried out in three steps. First step performs entitlement parsing: PPP interacts with existing MDM Server components (we logically group them as *MDM Policy Administration Point (PAP)*) to obtain the list of current entitlements. Each fetched entitlement is described by an accessor, a data action and a data association attribute.

Second step performs attribute extraction: PPP interacts with another group of MDM Server components (*Policy Information Point (PIP)*) to extract the resources that correspond to the attribute values of the parsed entitlements. In this step, data associations are associated with the list of associated objects they contain.

Third, and last, step builds DB2 authorizations and sends them to the database. For each associated object in an entitlement a single DB2 permission is generated in accordance to the established policy mapping. An accessor is mapped to a DB2 user or a group, a data action is mapped to a DB2 data manipulation operation and, lastly, an associated object is mapped to a DB2 table. The generated DB2 permissions are combined according to MDM Server policy combining and conflict resolution algorithms. In MDM Server a *union* approach between the entitlements is assumed (i.e. if a user belongs to a group which is allowed to see a data then, even though that user belongs to the other groups that are not allowed to see that data, the access will still be granted). No conflict resolution is required because the entitlements can only be granted and cannot be restricted (i.e., there are no *negative* entitlements) and the *default* policy is to restrict access unless it is specifically granted. Since DB2 employs the same policy combining and conflict resolution, we combine the generated DB2 permissions simply by issuing them directly to the database.

For each generated database permission, PPP logs all relevant information into the propagation log. This log contains, for each DB2 authorization, the reference to its parent entitlement, accessor, data action, data association and associated object. Information about the status of each DB2 authorization, whether it was issued to the database or was found duplicate, is also stored in this log. For example, for MDM entitlements that are shown in Fig. 2, PPP will generate and log five DB2 authorizations, however, only four of them will be issued to the database as one of the authorizations is a duplicate (assuming that no authorizations were issued for other MDM entitlements before).

Our solution also maintains the consistent relationship between MDM entitlements and DB2 permissions in case when either entitlements or their attributes are updated. A naïve approach to handle this problem would involve manual

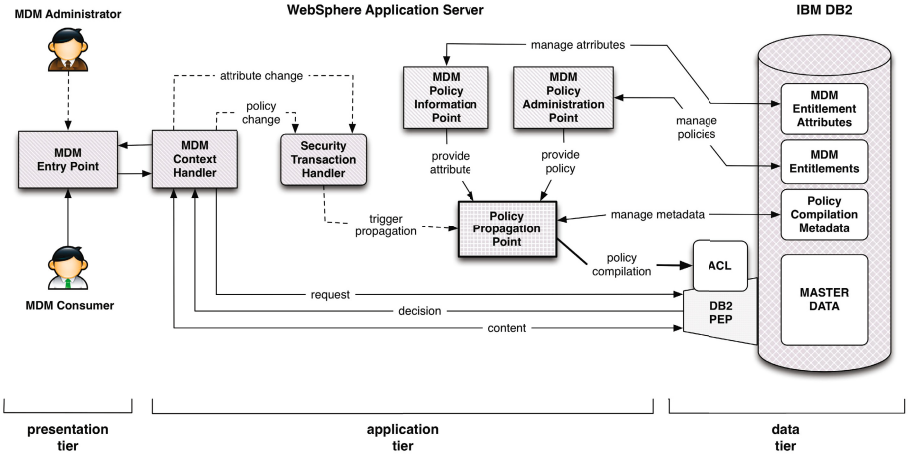


**Fig. 3.** Policy Propagation in MDM Server

periodical rebuilding of all DB2 permissions. Our *incremental propagation* approach aims to rebuild only a subset of DB2 permissions that is affected by the entitlement or the attribute update.

All administrative MDM transactions that deal with entitlements and their attributes are handled by MDM component called Security Transaction Handler (STH) (shown in Fig. 3). We modify this component in such a way that whenever it receives a transaction that deals with either entitlements or their attributes, it triggers PPP to *immediately* update the affected database permissions.

To illustrate how incremental propagation works consider the following example. Suppose that security administrator updates the data association attribute of the associated object BObj so that it no longer belongs to Grades (Fig. 2). STH notifies PPP that BObj has been updated. Then, PPP extracts from the transaction log all *old* DB2 permissions that were built from BObj. In this case, those are 1 and 5. Next, PPP checks the propagation log to see if there are unissued permissions that are duplicate to 1 and 5 and were not affected by the update. In this example, it finds permission 3. Therefore, only 5 is revoked and 3 is unmarked as being duplicate. Finally, PPP interacts with PIP and PAP to generate and issue to the database any *new* permissions resulted from the update.

## 4 Discussion

In this section we discuss the lessons that we learned from the implementation of the proposed security enforcement migration in MDM Server.

### 4.1 Integration Footprint

Industrial-scale enterprise applications have hundreds of thousands, sometimes even millions lines of source code. Due its large size a significant recoding of an initial implementation in order to integrate the proposed security enforcement migration is not feasible.

Our identity propagation implementation relies on trusted connections and trusted context features of DB2 and WebSphere. Trusted connections support client identity propagation and can also use connection pooling to reduce the performance penalty of closing and reopening connections with a different identity. Moreover, when WebSphere and DB2 are configured for use of trusted connections, WebSphere, seamlessly for the enterprise application, will propagate end-user identity to DB2 through the trusted data source. Therefore, the existing application code that uses WebSphere APIs to interact with DB2 through configured data sources does not need any modification. Thus, given a properly configured software stack, almost no initial code modifications are needed to implement the identity propagation in our approach, and it also doesn't incur noticeable performance degradation.

Our policy propagation mechanism is encapsulated in a single additional module which integrates seamlessly into the existing application security core architecture. It reuses existing application interfaces to obtain enterprise security

policies and attributes. Minor modifications to the initial code base were needed to enable propagation triggering mechanisms in existing application components.

Therefore, our approach has a small integration footprint on the existing code base, which makes it feasible to implement in a large-scale industrial-grade enterprise application such as MDM Server.

### 4.2 Policy Granularity

We can classify attribute-based security policies into two types based on the *granularity* of the data to which they apply. Specifically, if policy describes *object-level* authorizations, i.e. its resources are mapped to whole database tables, then we call such policy *coarse-grained*. For example, coarse-grained policy allows students to see *all course grades*, when course grades are contained in one, or several, database tables. On the other hand, if a policy describes *content-based* authorizations, i.e. its resources are mapped to specific rows of a table, then we call such policy *fine-grained*. For example, fine-grained policy allows students to see *only their own grades* that are contained in specific tuples of a course grades database table.

In our approach, we compile coarse-grained application policies into a set of database permissions. However, a different approach is needed to handle those application policies that are fine-grained, since individual tuples cannot be specified as objects in database authorizations. In theory, SQL standard provides a way to deal with fine-grained access control through view creation. For example, it is possible to create views for specific users, which allow those users access to only selected tuples of table. However, this approach is not scalable, and essentially impractical in systems with thousands or millions of users, since it would require millions of views to be manually created by a policy administrator.

For this reason, for the past several years, there have been much research on alternative solutions for fine-grained access control for databases. Implementations include *query rewriting* [14], *parameterized view creation* [15,16] and *extensions* to SQL [17–19]. *Hippocratic* databases use meta data stored in the database to make access decisions and are able to enforce cell-level policies. However, neither of those implementations are part of current publicly available commercial SQL servers.

The only currently publicly available implementation of fine-grained database access control (FGAC) is Oracle Virtual Private Database (VPD). This approach is based on Truman Models [15] and attaches predicates to user submitted queries to enforce row-level policies. However, in our implementation, we were limited to IBM software stack and thus were unable to use Oracle VPD. Use of Oracle VPD by switching to Oracle software stack, or a similar IBM offering when it becomes available, to map and synchronize fine-grained policies is a natural next step in our work.

### 4.3 Security vs. Performance

One of the aspects of policy propagation is the amount of time it takes the change in the enterprise policy, its attribute or attribute relationship to propagate from

the application to the database tier. If this time interval is relatively long, then the system security may be compromised. This time period is called the *window of access vulnerability* [20] which is caused by policy or attribute updates.

The duration of the window of vulnerability is determined by the implementation of policy synchronization engine. Generally, updates to database policies can be either *immediate* or *deferred*. In the immediate mode, whenever enterprise policies are updated, the updates to corresponding database policies are handled with the highest priority with no or relatively short delay. On the other hand, in the deferred mode, the updates to database policies are handled with lower priority and, therefore, the delay can be longer. The trade-off between immediate and deferred modes is the performance overhead imposed by the policy synchronization on the enterprise system.

In MDM domain, the confidential nature of master data has required us to handle the policy propagation in the immediate mode, however the implemented mechanism can be configured to operate in the deferred mode in order to help security administrators achieve a balance between security and performance of the system.

### 4.4 Mapping Correctness

It is important to establish application-to-database security policy mapping *correctly*, such that the permissions that are given in the database as a result of the policy propagation are indeed the permissions intended by a security administrator who defined them in the application. The understanding of privacy semantics of the application security policies is required in order to establish the correct mapping. Specifically, one needs to know both: how to map the *individual* policies and how to correctly *combine* them later.

In order to map the individual attribute-based policy to a set of database authorizations one needs to understand how application-level accessors, resources and data actions correspond to database users, database objects (such as tables and views) and SQL operations. Next, to combine the database authorizations that were produced by the compilation, one needs to use the *policy combining* and *conflict resolution* algorithms that were attached to the corresponding individual application-level policies.

In our implementation, policy combining was performed according to the union approach between MDM entitlements and MDM default no-access policy, hence no conflict resolution was required. Therefore, in MDM Server, it was sufficient to verify the mapping of individual application policies (which was correct given the object-relational design of MDM Server) in order to establish the correctness of the application-to-database policy mapping.

### 4.5 End-to-End Audit

An *audit trail* [11] is a process of secure recording of key system events and who initiated them in a chronological log. Auditing is used to reconstruct who-did-what after the fact. Typically, for a complete picture of a transaction on a data,

audit information must be collected at every enterprise system component along the transaction path - it is called an *end-to-end* trail. The ability of an enterprise system to provide *end-to-end* auditing facilities is essential for regulatory compliance.

When the enterprise security is enforced at the application tier, the requests to a database are made using a *system* identity - the identity that is used to represent the application layer, instead of the end-user identity. Since the end-user identity is not sent to the database it may be problematic to provide an audit trail at the database tier.

Following the proposed migration, the end-user identity is propagated down to the database tier and all the database requests required to perform a transaction are made using this propagated identity. This allows us to easily reconstruct the events after the fact by using native database auditing facilities instead of programming the application logic to collect the audit trail.

## 5 Migration Methodology

As we mentioned before, many application developers today choose to bypass database-level access control due to the lack of a native database ABAC support. Therefore, we believe that there are many enterprise applications out there which may benefit from our proposed application-to-database security enforcement migration.

Based on our case study, we propose a sound, methodological approach by which organizations can tackle migration projects. To keep our methodology as general as possible, we base it on a well-known industry standard for specifying and managing attribute-based policies - OASIS XACML. In addition to the policy language specification, XACML standard includes the description of the policy management architecture and its data flows (presented in Fig.4a). In our methodology, we propose the necessary changes to XACML reference architecture in order to accomodate the proposed migration and enable database-level enterprise security enforcement (Fig.4b).

We believe that ideal migration plan should be broken down into three phases. Known as analysis, implementation and integration, we define these phases in detail below.

In the analysis phase one should identify if consistent mapping can be established between enterprise attribute-based policies that are consumed by the PDP at the application tier and identity or role-based policies that are consumed by security mechanisms at the relational database. Similar to the migration in MDM Server, this phase might include establishing relations between enterprise data actions and database permission types, enterprise resources (such as business objects, their associations and hierarchies) and database tables, columns and rows, enterprise accessors (such as users, user groups, roles and role hierarchies) and database users, groups and roles.

Then, the core of the PPP is implemented. Given the attribute-based policies, enterprise accessors and resources that correspond to the policy attributes,

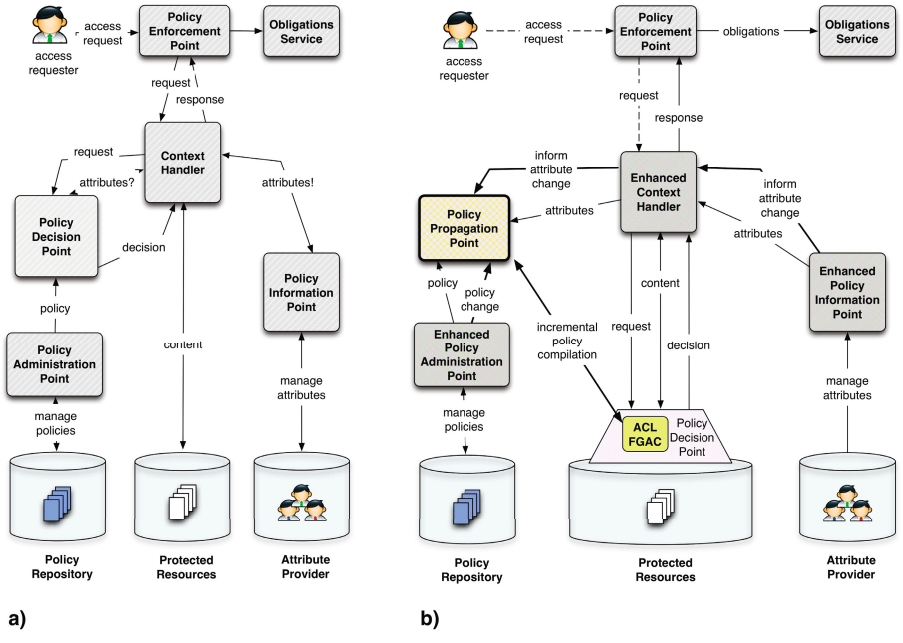


**Fig. 4.** Extended XACML Architecture

this component performs policy compilation and incremental updates according to the established mapping. It is important to realize that MDM Server's fairly straightforward and coarse-grained policy mapping and lack of negative authorizations allowed for relatively unproblematic implementation. However, finer-grained policies and presence of negative authorizations may considerably complicate this phase as necessary inconsistency and conflict resolution would have to be implemented.

Finally, the implemented engine is integrated into the enterprise system. Similar to the migration in MDM Server, two optimizations might be performed in order to keep the integration footprint small. First, PIP and PAP interfaces that supplied PDP with enterprise policies and evaluated the attributes are reused with PPP. Second, subject identity propagation is implemented with the help of a novel trusted context feature of an underlying software stack. Such feature are already supported by IBM and Oracle application servers and relational databases, with more vendors surely to follow. Finally, existing PIP and PAP are modified to enable PPP notification mechanisms necessary for timely incremental policy propagation.

The proposed migration has the following business benefits. First, it ensures better data protection in the enterprise as it eliminates the risks that are associated with using common application layer's authorization ID by providing an additional layer of security at the database tier. Second, by complying with the least common mechanism principle of protection of information in computer systems our migration eases end-to-end enterprise audit and, consequently, it

eases the organization's regulatory compliance. Finally, it may lower the overall transaction processing times due to efficiency of the database-tier security engine when compared to the application-tier one.

## 6 Related Work

In this section we discuss the related works in the areas of enterprise access control systems and attribute-based database access control techniques.

Enterprise access control enforcement has been interpreted in several efforts. Tivoli Access Manager [21] developed by IBM attempts to decouple the authorization logic from the application logic to allow security policy externalization by using a proxy that sits in front of the application server. IBM Tivoli Security Policy Manager [22] and Axiomatics Policy Server [23] build on that work by enabling enterprise architects to centrally define, manage and enforce XACML security policies for applications and data resources within the enterprise. Ladon [24] is Java-based API that enforces access control policies written in XACML by rewriting incoming SQL queries. However, in these products, the security policies are still enforced at the application level and inherit all disadvantages of this approach.

Little work has been done on the database-level attribute-based policy verification. Our work follows the recent work [20] which aims to compile attribute-based XACML policies into ACLs that are supported by conventional databases. The authors develop a toy prototype based on MySQL relational database to show that such compilation significantly improves attribute-based database access time with a price of reasonable off-line compilation time. Inspired by this work, we realized that such compilation can be used to provide an enhanced security, end-to-end audit and simplified compliance in many existing enterprise systems. Our work builds on this observation by describing the security enforcement migration in existing industrial-grade system and providing standards-driven migration methodology based on the enterprise software stack that enables database-level attribute-based policy enforcement in existing enterprise systems.

## 7 Conclusions

In this work, we presented a case study of migration of access control enforcement from the application to the database tier. This migration aims to overcome the security and audit concerns that are associated with the application-tier security mechanisms in an existing enterprise application. We considered our approach in the context of master data management where data security, privacy and audit are subject to regulatory compliance. We implemented the proposed migration in an industrial-grade MDM system to prove the applicability and usefulness of our idea. Finally, we proposed a general standards-driven and vendor-independent methodology that is designed to tackle similar security enforcement migration projects.

The proposed migration has two positive effects on the overall security and regulatory compliance of the enterprise system. First, our approach enhances the security by eliminating the vulnerability to privilege escalation attacks. Second, our approach eases the enterprise end-to-end audit by enabling native database-tier auditing facilities. Further, our approach has a small integration footprint by effectively reusing existing enterprise security components and novel features of an underlying software stack. Our reference model seamlessly integrates into the existing enterprise security architecture by requiring only minor modifications to the initial enterprise system code base, which makes it feasible to implement in a large-scale enterprise application.

There are two areas of further interest to us in this project. First, as we mentioned in Section 4.2, our approach handles only coarse-grained table-level policies. This is caused by the limitations of an underlying relational database, which in our case was IBM DB2. In the future, we plan to experiment with Oracle software stack to extend the policy propagation engine to handle fine-grained access control (FGAC). We envision that this can dramatically improve the enterprise transaction processing times since the selectivity of FGAC-transformed query is higher than that of the original query due to introduction of policy related predicates. Second, we plan to perform an evaluation of our approach in an actual production enterprise environment. The organization of the protected data and its accessors and the patterns in which the policies or their attributes are changed would help us understand the real effects our approach has on the enterprise system. This information will be invaluable in determining further optimizations to our engine.

## References

1. Scott Graham, G., Denning, P.J.: Protection: Principles and Practice. In: Proceedings of the Spring Joint Computer Conference, AFIPS 1972, May 16-18, pp. 417–429. ACM, New York (1972)
2. Jajodia, S., Sandhu, R.: Toward a Multilevel Secure Relational Data Model. SIGMOD Rec. 20, 50–59 (1991)
3. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based Access Control Models. Computer 29(2), 38–47 (1996)
4. Wang, L., Wijesekera, D., Jajodia, S.: A Logic-based Framework for Attribute Based Access Control. In: Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering, FMSE 2004, pp. 45–55 (2004)
5. Pfleeger, C.P., Pfleeger, S.L., Safari Tech Books Online: Security in Computing, vol. 604. Prentice Hall (2007)
6. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering Code-injection Attacks with Instruction-set Randomization. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 272–280. ACM (2003)
7. United States Code. Sarbanes-Oxley Act of 2002, PL 107-204, 116 Stat 745 (2002)
8. Security Standards Council. PCI DSS v2.0 (2010)
9. Allender, M.: HIPAA compliance in the OR. Aorn Journal (2002)
10. Saltzer, J.H., Schroeder, M.D.: The Protection of Information in Computer Systems. Proceedings of the IEEE 63(9), 1278–1308 (1975)

11. Dreibelbis, A., Hechler, E., Milman, I., Oberhofer, M., van Run, P., Wolfson, D.: Enterprise Master Data Management: An SOA Approach to Managing Core Information. IBM Press (2008)
12. Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/
13. Zeilenga, K., et al.: Lightweight directory access protocol (ldap): Technical specification road map. Technical report, RFC 4510 (June 2006)
14. Franzoni, S., Mazzoleni, P., Valtolina, S., Bertino, E.: Towards a Fine-Grained Access Control Model and Mechanisms for Semantic Databases. In: IEEE International Conference on Web Services, ICWS 2007, pp. 993–1000 (2007)
15. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending Query Rewriting Techniques for Fine-grained Access Control. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD 2004, pp. 551–562 (2004)
16. Roichman, A., Gudes, E.: Fine-grained access control to web databases. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, SACMAT 2007, pp. 31–40 (2007)
17. Stoller, S.D.: Trust Management and Trust Negotiation in an Extension of SQL. In: Kaklamanis, C., Nielson, F. (eds.) TGC 2008. LNCS, vol. 5474, pp. 186–200. Springer, Heidelberg (2009)
18. De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Trust management services in relational databases. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, pp. 149–160. ACM (2007)
19. Chaudhuri, S., Dutta, T., Sudarshan, S.: Fine grained authorization through predicated grants. In: IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 1174–1183. IEEE (2007)
20. Jahid, S., Gunter, C.A., Hoque, I., Okhravi, H.: MyABDAC: Compiling XACML Policies for Attribute-based Database Access Control. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy, pp. 97–108. ACM (2011)
21. Karjoth, G.: Access Control with IBM Tivoli Access Manager. ACM Transactions on Information and System Security (TISSEC) 6(2), 232–257 (2003)
22. IBM. Tivoli Security Policy Manager (2011), http://www-01.ibm.com/software/tivoli/products/security-policy-mgr/
23. Axiomatics. Axiomatics Policy Server (2011), http://www.axiomatics.com/products/axiomatics-policy-server.html
24. SourceForge. Ladon - XACML enforcement for DB2 (2009), http://xacmlpep4db2.sourceforge.net/