Enhancing Database Access Control with XACML Policy

Sonia Jahid, Imranul Hoque, Hamed Okhravi, Carl A. Gunter
University of Illinois at Urbana-Champaign

ABSTRACT

XACML is apparently the most convenient way to express attribute-based access control policies. Though XACML has been used in several access control areas, processing XACML policies for attribute-based database access control still has not been studied in depth. In this work we compile XACML policies, and utilize the underlying database access mechanisms such as ACLs to protect sensitive data. We use the attribute information residing in the database itself to define high level XACML policies and transform this policy to low level database access policies for access decisions on another part of the database. We implement and evaluate our idea over a synthetic database and come up with efficient policy compilation and verification time.

1. INTRODUCTION

eXtensible Access Control Markup Language (XACML) [2] has widely been accepted to specify attribute-based policies for access control in several paradigms. Though native database access control mechanisms like access control lists (ACL) are quite efficient to implement identity-based access control, they lack the flexibility and expressiveness provided by attribute-based access control (ABAC). In this work, we propose a concept to unify the flexibility of attribute-based access control and the efficiency of ACLs. We specify attribute-based policy in XACML and transform it into database ACLs to provide information protection within the database. The challenges here are to bridge the gap between high level ABAC and low level IBAC, and to maintain consistency in the target database ACLs when any policy or data changes.

Most organizational databases contain user information such as name, address, age, gender, etc. Database tables and columns can have attributes attached to them too. These user and resource attributes can be utilized to perform access control on the sensitive information contained in the database. Database access control mechanisms such as ACLs provide identity-based access control and lack the option for ABAC. We specify access control policies in XACML which defines the rules between user and resource attributes and compile these policies into database ACLs utilizing the information contained in the database itself.

This approach comes with several advantages: in general, attribute-based access control is much more expressive than identity-based access control for organizations where many users share same types of permissions on similar types of resources. It provides the intent rather than extent of the context. Second, several existing approaches perform access decision on the fly, thus slowing down the decision making process. Preprocessing ABAC policies speeds up the decision making process by utilizing the fact that when a user wants to access something, the result is already there. Though some recent efforts focus on preprocessing XACML policies, they are not specialized for database. Finally, database ACLs provide information protection from inside the database and support the principle of the least privileges. In application level policy enforcement, there still exist the possibility of bypassing the mechanism and gaining illegal access to the database

We describe our approach in section 2 and present evaluation results in section 3. Section 4 presents some related works and Section 5 concludes.

2. DESCRIPTION

The whole process in our approach can be divided into three steps:

Step 1: Policy Processing

Access rules define who can access what. Attribute based policies define similar types of permissions for a collection of subject and resources using attributes instead of individual identifiers. In this paper, we specify XACML policies using the user and resource attributes already existing in the database. This is an abstract form of access policy and so is instantiated by preprocessing. The following example clarifies the approach:

Example: 'employees ranked as professors or tenured assistant professors can select from, insert into, and delete from graduate admission related tables in the database'.

The policy has three parts:

Subject: $\langle professor \rangle, \langle tenured, assistant\ professor \rangle$
Resource: $\langle graduate\ admission\ tables \rangle$
Action: $\langle select, insert, delete \rangle$

These are extracted from the policy, and appropriate SQL statements are formed to extract the exact users and resource tables. We assume that the enterprise database contains tables to store user information. We call these tables attribute tables. Database servers such as MySQL [3] provides options to attach attributes to the data tables in the

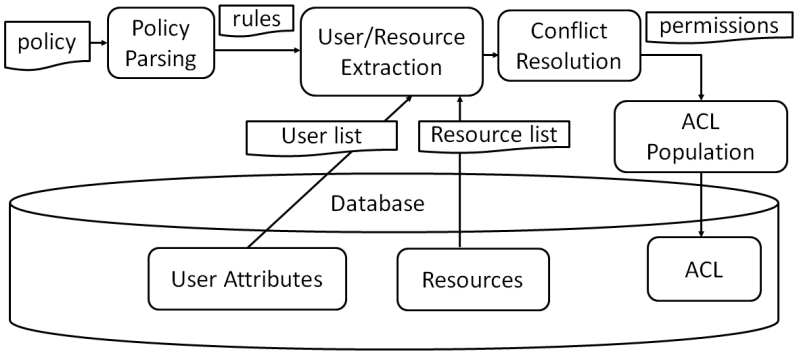

Figure 1: Policy Compilation Flowchart

form of metadata. The attribute tables and metadata are used to extract the users and resources from the database respectively. The following statements retrieve the users and table names (that satisfy the policy) from the database and pass them onto the next step.

```
SELECT username FROM employees.info
WHERE employee_type='Professor'
OR (employee_type='Assistant Professor' AND sta-
tus='Tenured');

SELECT table_name FROM information_schema.tables
WHERE table_comment='graduate admission';
```

Here, we assume that the database contains user attributes in `info` table of `employees` database, and table metadata in `tables` table of `information_schema` database.

Step 2: Policy Transformation

Database maintains a list of access rules in the form of who can access what (access control matrix) or what can be accessed by whom (access control list). Generally, individuals are added manually to a table or column ACL. In order to unify ABAC with IBAC, our approach builds up these ACLs with the list of users and resources obtained in Step 1. This step is basically instantiating the high level policies and transforming them into native database access mechanisms, thus bridging the gap between attributes and identifiers. In this step, `GRANT` SQL statements are constructed which populate the database ACLs.

Example: Users 'u_0, u_5, and u_7' and tables 't_5 and t_8' satisfy the attributes and are extracted by the SQL statements in the previous step. One SQL `GRANT` statement is executed for each table to populate the corresponding ACLs.

```
GRANT SELECT,INSERT,DELETE ON t5 TO u0, u5, u7;
GRANT SELECT,INSERT,DELETE ON t8 TO u0, u5, u7;
```

These update the underlying database access control mechanism, and hence policy is enforced in the database using database level policies, i.e., ACL.

Step 3: Policy Enforcement

Policy enforcement in our approach is performed within the database. If a user has some kind of access on any table, that user already resides in the ACL of that specific table. Unlike middleware based approaches such as Sun XACML implementation [8] where access decision is made and enforced at the application level, our approach protects the data from inside. If a user u_0 wants to read from table t_5 using the SQL statement '`SELECT * FROM` t_5', the policy is enforced at the moment she connects to the database.

Conflict Resolution

A policy conflict is resolved at data level during compilation. For instance, policy P_1 permits actions A_1 to users S_1 on resources R_1, and P_2 denies A_2 from users S_2 on resources R_2 where $S_1 \cap S_2 = S(S \neq \emptyset)$, $R_1 \cap R_2 = R(R \neq \emptyset)$, $A_1 \cap A_2 = A(A \neq \emptyset)$. This is a conflict for S over R for A. A redundancy occurs for same condition but when both P_1 and P_2 have the same effect, i.e., both of them permit or deny. The decision about which one should be active depends on the conflict resolution algorithms in the policy. In case of a permit-override, the conflicting permissions are added to the database ACLs. Deny-overrides, on the other hand, would not add the permissions to the database ACL, moreover, it will revoke the permissions if they already exist. A first-applicable will activate the permission that shows up first in a policy.

We get a set of permissions ⟨users, database resources, actions, effect⟩ to populate the database ACLs in Step 2 of our compilation. Suppose we get three sets of permissions $\langle u_0, t_0, select, permit\rangle$, $\langle u_0, t_0, insert, permit\rangle$, and $\langle u_0, t_0, select, deny\rangle$, and the combining algorithm is deny-overrides. The 2nd permission does not have anything in conflict and so it is added to t_0 ACL. Since the 1st and the 3rd ones are in conflict, the 3rd one is prioritized because of the combining algorithm. So, the final permissions are $\langle u_0, t_0, insert, permit\rangle$, and $\langle u_0, t_0, select, deny\rangle$. Hence, $\langle u_0, t_0, insert, permit\rangle$ is added to the database ACL using `GRANT` statement and $\langle u_0, t_0, select, deny\rangle$ is removed (if exists) using `REVOKE` statement. A pictorial description of our approach is shown in Figure 1.

3. EVALUATION

To evaluate our idea we implemented a prototype that compiles XACML policy into MySQL [3] ACLs. We designed a synthetic resource database and populated it with random data. The user attribute table consists of 50,000 users each with 100 attributes. We constructed XACML policies with

100, 1000, 2000, ..., 5000 rules. All experiments were carried out on a 2.40GHz Intel Core2 Duo with 3GB memory, and running Ubuntu 8.10. The database server was MySQL version 5.0.67-community-nt. The results are presented in Figure 2.

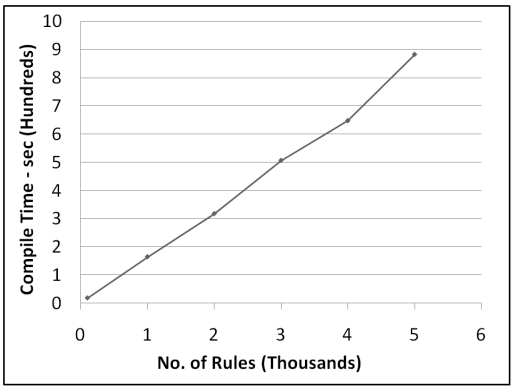

Figure 2: Policy Compilation Time

A 5000 rule policy compilation time required 882sec with the following breakdown - parse time 31, user retrieval time 720, and ACL population time 131. The compilation resulted in 34558 user extraction from the database, and after the conflict resolution, 170757 different permissions were added to the database ACL through 120 `GRANT` statements. Parse time is linearly proportional to the number of rules. User extraction depends on the complexity in the `WHERE` clause of a `SELECT` statement for each `Rule`, and the underlying data type. ACL population is a series of `GRANT` statements. If some deny permissions show up that already reside in the database ACL, `REVOKE` statements are used to remove them. ACL update time depends on the number of `GRANT(REVOKE)`s and the number of users per `GRANT(REVOKE)`.

When a user wants to access a database resource, she tries to establish a connection with the database. If she is in the database ACL, the connection is established, otherwise her access is denied. The access verification time that a user faces is the time to establish a connection which we tested to be 287ms in our experiments.

4. RELATED WORK

Database access policy enforcement has been interpreted in several works. Roichman *et al.* [7] have performed on-the-fly IBAC using parameterized views. Parameterized views are still not a part of current SQL servers. Besides, it requires instantaneous view creation on the tables to be accessed and queries to be re-writen on-the-fly. Our approach has no such requirement: it utilizes ACLs existing in almost all database access control mechanism. Olson *et al* [6] uses transaction datalog to describe policies for RDBAC. Policy checking and hence access control is done on-the-fly. This work also focuses on theory and does not address efficiency issues. The approach in [1] by Cook *et al.* uses a middleware named rule-engine to intercept user submitted query and change it if necessary to abide by rules. Our approach compiles policies into ACLs and does not need any query interception. The query is directly transferred to the database.

XACML policies are interpreted in some existing works such as Sun XACML implementation [8] since it is generalized for a wide range of platforms and access control domains. Another work such as XEngine [4] has preprocessed XACML policy to speed up the access control decision making process. Both of these approaches perform policy enforcement outside the database. An equivalent XACML request is generated from the user credentials when she submits an access request. Sun implementation does on-the-fly policy verification and XEngine prepares a decision table from before. If an access decision is positive, the user access is permitted.

Another approach to speed up XACML policy evaluation is proposed by Marouf et al. [5]. They use statistical analysis of the policy to determine which rules inside an XACML policy are encountered more often. Then using rule reordering and clustering techniques, they make the evaluation process faster and more efficient than Sun implementation.

None of these works are specialized for attribute-based database access control. Their focus is to speed up XACML policy processing. We take attribute-based XACML policies, instantiate the policies over existing data in the database, and enforce the policies using the database ACLs to perform ABAC.

5. CONCLUSION

We have presented an idea of compiling XACML policies over databases to perform attribute-based database access control. The novelty of our approach lies in combining policy pre-processing and transforming it into a low level database access control mechanism utilizing the information already existing in the database. We have implemented the idea as a proof of concept and evaluated it over a synthetic database.

6. REFERENCES

[1] W. R. Cook and M. R. Gannholm. Rule based database security system and method, November 2004.

[2] S. Godik and T. Moses. eXtensible Access Control Markup Language (XACML). Technical Report v1.1, OASIS, August 2003.

[3] Mysql. `http://www.mysql.com`.

[4] A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: A fast and scalable xacml policy evaluation engine. In *ACM Sigmetrics*, 2008.

[5] S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran. Statistics & clustering based framework for efficient xacml policy evaluation. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:118–125, 2009.

[6] L. Olson, C. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *The 14th ACM conference on Computer and communications security*, 2008.

[7] A. Roichman and E. Gudes. Fine-grained access control to web databases. In *12th ACM symposium on Access Control Models and Technologies*, 2007.

[8] Sun Microsystems, Inc. *Sun's XACML Implementation.*