A Formal Framework for Reflective Database Access Control Policies

Aaron Lovato

New Mexico Tech

April 26, 2010
1 Introduction
2 Problem Addressed
3 Background
4 Datalog
5 Policies
6 Security Analysis
7 Implementation
8 Conclusion
Introduction

Paper published in 2008

Written by Lars E. Olson, Carl A. Gunter, and P. Madhusudan

Basic Ideas
- Replace ACLs in databases with reflective queries
- Reflective access control refers to access control based on the existing state of the database
- Use Transaction Datalog for expressing policies
Problem Addressed

- More traditional access control paradigms overly complicate administration of common policies
- RDBAC is in a limited level of development
  - Current implementations don’t fully address problems with traditional access control paradigms or have limitations that can be overcome
    - E.g., Oracle’s VPD
  - Some open up security holes in the database
  - No formal definition of RDBAC
Examples

- The “each employee can view their own data” policy
  - Requires individual ACL entries for each employee
  - Management becomes increasingly tedious when adding users, implementing policies, or changing the schema

- The “managers can access the data of all employees in their department” policy
  - Roles can be used to provide a solution to this problem
  - However, roles describe extent rather than intent
    - A user’s role needs to be updated upon promotion to prevent leaving the database in an inconsistent state
Current Implementations

- Current implementations of RDBAC have at least three drawbacks
  - Privilege to define policies is considered an administrator privilege
  - Policies that refer back to the table being queried are not allowed
  - No formal definition
Examples of RDBAC

- VPD
  - Uses arbitrary code written as a stored procedure to implement policy
  - Example

```sql
create or replace function employeeFilter
(p_schema varchar, p_obj varchar)
return varchar as
begin
    return username = ||
            SYS_CONTEXT(userenv, SESSION_USER) || ;
end
```
Examples of RDBAC

- **VPD**
  - Rewrites `select * from employee` to `select * from employee where username = 'BOB'` when Bob is logged in
  - Similar policies can be defined in other commercial databases
  - Can be used to create a security loophole for users that can define their own policies
  - Example
create or replace function attackFilter
(p_schema varchar, p_obj varchar)
return varchar as
begin
    for row in (select * from alice.employees) loop
        insert into bob.leaked_info values(
            row.username, row.ssn, row.salary, 
            row.email);
    end loop;
    commit;
    return ;
end
Suppose Bob has a table that he can create the above policy for and creates a *leaked_info* table.

If another user executes a query on Bob’s table, her data is copied to Bob’s table.

Oracle’s response to this problem included the following:

- Since creating the policy is an administrator privilege, Bob could easily just drop the policy protecting another user’s data.
- Alice could change her policy to prevent this attack.
- There is always a danger that users can be tricked into executing someone else’s code.

One solution is to prevent policies from editing the database.
Datalog Syntax

- Three types of symbols: variables, constants, and predicates
- A literal is a string of the form $p(t_1, t_2, ..., t_n)$ where $p$ is a predicate and each $t$ is either a constant or variable
- A rule is a statement of the form $p : q_1, q_2, ..., q_n$, where $p$ and each $q$ is a literal
  - $p$ is the head of the rule and the $q$’s are the body
  - A rule with a head that contains no variables and an empty body is a fact (can be written without the colon and hyphen separator)
- A database is a non-ordered, possibly infinite set of rules
- Built-in rules exist that do not vary over databases
Datalog Example

employee(alice, 90000, hr, manager).
employee(bob, 70000, sales, clerk).
employee(carol, 90000, sales, manager).
employee(david, 80000, hr, cpa).
manager(Person, Dept) :- employee(Person, Salary, Dept, manager)
Datalog Semantics

- Simple inference system where predicates are inductively derived from facts and repeatedly using rules
- A rule derives the head if there is an assignment to the variables such that the body of the rule is conjunctively true with respect to this assignment
- Formal inference rules left out of the paper...
Transaction Datalog

- An extension to Datalog that allows rules to modify the underlying database
- RDBAC only uses serial conjunction
- Rules evaluated in isolation
- Two special predicate names defined
  - \( ins.p \)
  - \( del.p \)
  - Always true
- The head of a rule can not be one of these special predicates
- Inference requires two copies of the database
Transaction Datalog

- The sequence of states in the derivation of the body of a rule must be continuous.
- In case of a failure, changes are rolled back so database remains unchanged.
- Example

\[
hire(\text{Name}, \text{Salary}, \text{Dept}, \text{Pos}) :- \geq(\text{Salary}, 50000),\ 
\text{ins.employee}(\text{Name}, \text{Salary}, \text{Dept}, \text{Pos})
\]
Transaction Datalog Example

- Infer $\geq (60000, 50000)$
- Infer ins.employee(emily, 60000, support, service)
- Infer hire(emily, 60000, support, service)
Example Policies

1. view.employee(User, Person, Salary, Dept, Pos) :-
   employee(Person, Salary, Dept, Pos),
   =(User, Person).
2. view.employee(User, Person, null, Dept, Pos) :-
   employee(User, _, Dept, manager),
   employee(Person, _, Dept, Pos).
3. view.ins.employee(User, Person, Salary, Dept, Pos) :-
   employee(User, _, hr, _),
   ins.employee(Person, Salary, Dept, Pos).
4. view.picnic(User, Person, Assignment) :-
   employee(Person, Salary, Dept, Pos),
   ins.leaked_info(Person, Salary, Dept, Pos),
   picnic(Person, Assignment).
Corrected Policy

1. view.picnic(User, Person, Assignment) :-
   view.employee(bob, Person, Salary, Dept, Pos),
   view.ins.leaked_info(bob, Person, Salary, Dept, Pos),
   view.picnic(bob, Person, Assignment).
2. view.picnic(bob, Person, Assignment) :- picnic(Person, Assignment).
3. view.ins.picnic(bob, Person, Assignment) :- ins.picnic(Person, Assignment).
4. view.del.picnic(bob, Person, Assignment) :- del.picnic(Person, Assignment).
5. view.leaked_info(bob, Person, Salary, Dept, Pos) :- leaked_info(Person, Salary, Dept, Pos).
6. view.ins.leaked_info(bob, Person, Salary, Dept, Pos) :-
   ins.leaked_info(Person, Salary, Dept, Pos).
7. view.del.leaked_info(bob, Person, Salary, Dept, Pos) :-
   del.leaked_info(Person, Salary, Dept, Pos).
Security Analysis and Decidability

- HRU model
- Restrictions on policies needed to make decidable analysis algorithms possible
  - Untrusted users can not execute policies that have side-effects
  - Append-only policies with “safe rewriteability”
Implementation

- The authors implemented a POC query engine using SWI-Prolog
- Provides rudimentary database functionality
- Results show that RDBAC incurs a cost inherent to executing extra queries on the database
- Implemented Chinese Wall and security analysis checks
Model for RDBAC based on Transaction Datalog
- Clearly lays out policies, privileges, and extension to non-privileged users
- Many policies require operations that are not formally analyzed yet
- Decidable algorithms given
- Negations (extension to Transaction Datalog)
- Showed how unsafe information flow can be prevented
Discussion
Lars E. Olson, Carl A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies.