A Formal Framework for Reflective Database Access Control Policies

Lars E. Olson    Carl A. Gunter    P. Madhusudan

Speaker: Brian Wongchaowart
October 19, 2009
Motivation

- A database access control policy is *reflective* if it depends on the current state of the database...
- This paper was published in 2008. Why is this novel?
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;
  commit;
  return ' ';
end
Motivation

• “Existing implementations of RDBAC have no formal description.”
• This paper proposes Transaction Datalog as a formal framework for studying reflective access control policies.
Each rule in Transaction Datalog represents a procedure.
The head is the procedure name, and the body is the procedure definition.
Variables in the rule represent parameters that are given values at run time.
Transaction Datalog

- Transaction Datalog rules can not only read the current database state, but also modify it by inserting and deleting facts (\textit{ins.p} and \textit{del.p}).

```
hire(Name, Salary, Dept, Pos) :-
    \geq(Salary, 50000),
    ins.employee(Name, Salary, Dept, Pos).
```
• For each database predicate name $p$ with arity $n$, this paper defines three special predicate names: $\text{view} . p$, $\text{view} . \text{ins} . p$, and $\text{view} . \text{del} . p$, each with arity $n + 1$.

• Unlike $\text{ins} . p$ and $\text{del} . p$, these predicates are not part of the Transaction Datalog language.
Transaction Datalog for Access Control

- If $\text{view} \cdot p(U, T_1, \ldots, T_n)$ can be derived from the current database state, then user $U$ is granted read access to the values of $p(T_1, \ldots, T_n)$.
- The derivation of $\text{view} \cdot p(U, T_1, \ldots, T_n)$ may change the database state.

```prolog
view.employee(User, Person, Salary, Dept, Pos) :-
  employee(Person, Salary, Dept, Pos),
  =(User, Person).
```
Transaction Datalog for Access Control

• If \( \text{view.ins.p}(U, T_1, \ldots, T_n) \) can be derived from the current database state, then user \( U \) is allowed to insert the fact \( p(T_1, \ldots, T_n) \) into the database.

• If \( \text{view.del.p}(U, T_1, \ldots, T_n) \) can be derived from the current database state, then user \( U \) is allowed to delete the fact \( p(T_1, \ldots, T_n) \) from the database.

\[
\begin{align*}
\text{view.ins.employeee} & (\text{User}, \text{Person}, \text{Salary}, \text{Dept}, \text{Pos}) :&
\text{employee} & (\text{User}, _, \text{hr}, _), \\
& & \text{ins.employee} & (\text{Person}, \text{Salary}, \text{Dept}, \text{Pos}).
\end{align*}
\]
view.picnic(\text{User}, \text{Person}, \text{Assignment}) :-
  employee(\text{Person}, \text{Salary}, \text{Dept}, \text{Pos}),
  \text{ins.leaked}\_\text{info}(\text{Person}, \text{Salary}, \text{Dept}, \text{Pos}),
  \text{picnic}(\text{Person}, \text{Assignment}).
Access to the database for any non-administrator user must be restricted to using only the view predicates.

\[
\text{view.picnic}(\text{User}, \text{Person}, \text{Assignment}) :- \\
\text{view.employee}('bob', \text{Person}, \text{Salary}, \text{Dept}, \text{Pos}), \\
\text{view.ins.leaked_info}('bob', \text{Person}, \text{Salary}, \text{Dept}, \text{Pos}), \\
\text{view.picnic}('bob', \text{Person}, \text{Assignment}).
\]
Basic Privilege Rules

view.picnic('bob', Person, Assignment) :-
    picnic(Person, Assignment).

view.ins.picnic('bob', Person, Assignment) :-
    ins.picnic(Person, Assignment).

view.del.picnic('bob', Person, Assignment) :-
    del.picnic(Person, Assignment).
Security Analysis and Decidability

- The primary advantage of using Transaction Datalog is that the formal safety analysis of some classes of policies is decidable, in the sense that it can be determined whether a specified privilege (\texttt{view}.\textit{p}, \texttt{view}.\textit{ins}.\textit{p}, or \texttt{view}.\textit{del}.\textit{p}) over specified facts can ever be leaked to an untrusted user by executing Transaction Datalog rules.
- This requires examining not just the current system state, but all future system states.
Theorem 1

- There exists a set of non-recursive Transaction Datalog rules that can simulate the HRU model.
- Thus the security analysis of Transaction Datalog is undecidable in the general case.
Theorem 2

- Security analysis is decidable for a database with state $S$ and transaction base $P$ with all rules containing no side effects.
- Transaction Datalog reduces to Datalog when there are no side effects (database updates).
Rewriting Append-Only Policies

- Rules with only insertions and no deletions can be rewritten in plain Datalog.
- Let $r$ be the rule $p :- p_1, p_2, \text{ins}.p_3, p_4, \text{ins}.p_5, p_6$.
- Then rewriting $r$ produces the set of rules:
  
  $p_3 :- p_1, p_2$.
  $p_5 :- p_1, p_2, p_4$.
  $p :- p_1, p_2, p_4, p_6$. 

Theorem 4

- A set of Transaction Datalog rules is safely rewritable if rewriting them produces safe Datalog rules.
- Security analysis is decidable for a database with state $S$ and transaction base $P$ with rules that contain no retractions and are safely rewritable, given a finite number of users.
For a database with state $S$ and transaction base $P$ with rules that contain no retractions and are safely rewritable, if a given permission does not exist in the model of the Datalog database derived from the union of $S$ and the rewritten rules from $P$, then it will not be accessible in any future state of the current database if all rules are monotonic.
Subsequent Work

- Existing database management systems cannot evaluate access control policies written in Transaction Datalog.
- One solution is to compile Transaction Datalog rules into SQL view definitions.
Strengths

- The formal safety analysis possible for the reflective access control policies considered in this paper is an important step towards verifying that a set of policies actually corresponds to the intention of the policy author.
Strengths

- A safety guarantee proven using the methods in this paper applies not only to access control policies written in Transaction Datalog, but also to any equivalent access control policies written in a language supported by a real database management system.
- Thus reasoning with Transaction Datalog is useful even if the actual policies are never written in Transaction Datalog.
Weaknesses

- No proofs!
Weaknesses

- The security analysis in this paper only considers the effect of database state changes caused by an untrusted user executing Transaction Datalog rules. In reality, database state changes are part of the normal operation of a database.
Weaknesses

- There is no support for access control on update operations. Updates are clearly not equivalent to a deletion followed by an insertion from the point of view of enforcing constraints on the relationship between the old data values and the new values.
Summary

- This paper proposes Transaction Datalog as a formal framework for studying reflective database access control policies.
- Policies written in Transaction Datalog can not only read the current database state, but also modify it by inserting and deleting facts.
- Reasoning with Transaction Datalog makes the formal safety analysis of some classes of policies decidable.
Questions/Comments