Brief Announcement: Extending SQL Access Control to Derived and Distributed Data

Arnon Rosenthal
MITRE Corporation
arnie@mitre.org

Edward Sciore
Boston College
sciore@bc.edu

Access-control mechanisms in distributed systems can differ greatly. For example, a data warehouse materializes a separate database; the underlying source administrators then grant full privileges on their contributed data to the warehouse administrator, trusting that he will grant appropriate privileges on the warehouse data. On the other hand, federated database researchers have source and federation administrators collaborating to specify the rights of federation users. Research papers have proposed access-control models specifically for federations, distributed-object systems, XML systems, etc.

These proposed models, although academically interesting, are neither easy for administrators to use nor attractive for vendors to implement. Our approach is to extend SQL in three ways:

- inferred privileges on derived data;
- privileges for use in constructing derived data;
- factored privileges, to model different execution concerns.

By combining these features, we can represent many policies specifiable in other researchers' models, with significantly simpler administration and implementation.

We explain these three components of our model via a running example; refer to the full paper for details [RS]. Consider a national network of hospitals, each having the local table:

TREATMENT(HospitalID, Date, Category, PatientID, Cost)

There is also a data warehouse based on these source tables.

Privilege Inference:

Suppose the sources authorize the first three fields of TREATMENT to be publicly readable. The warehouse administrator (*WA*) creates a view TMT_INFO, which combines these values over all hospital tables. SQL's semantics cause two difficulties: First, the WA must have grant-option privileges on the underlying tables; this means that each hospital administrator must trust the WA to only grant privileges they would authorize. Second, the WA must actually manage the grants on the view, which can involve tedious, error-prone replication of the grants issued on the source tables.

We solve these problems by allowing privileges on the source tables to filter up to the view. Formally, a user can receive an inferred privilege on a derived object if both (a) the creator of the object authorizes the inference, and (b) the user would be able to get the same effect by creating the object himself. Consequently, the WA does not need special privileges to create a view. The WA grants inference privilege to selected users; a user who also has sufficient privileges on the source tables will automatically receive privileges on the view. If the WA also has grant-option authority on the view, he can additionally grant privileges to other users, beyond what can be inferred. This is an example of *joint privilege management.*

Use-Specific Privileges:

In order to create and manage a summary table of treatments and their cost, the WA must have grant-option authority on it. If the last two fields of TREATMENT are sensitive, the WA may not receive this authorization, and the view cannot be created.

We extend SQL so that source administrators can grant to the WA *use-specific* privileges on the source tables, which can be used only to evaluate the specified view. The WA will be granted the desired privilege on the view when he is able to obtain use-specific privileges on all tables underlying his view. Thus the local data is reasonably secure, since a user would have to subvert the DBMS to get at it.

Factored Privileges:

In SQL, a privilege is granted in a single command. However, the decision to grant a privilege may involve several independent factors, such as need to know, resource usage, security implications, and company policy. These factors should be separable, so that each can be made by the best informed person and changed independently.

We extend SQL so that each privilege is composed of smaller *privilege factors*, which can be administered separately and at different levels of granularity. Although the number of decisions will increase, each decision is simpler, less often changed, and more able to exploit inference. Training costs and total labor should decrease.

A privilege is granted only if a user has privileges for each relevant factor. For example, suppose the TREATMENT table has three factors: a machine-level factor ("only users having a hospital account can run queries on this machine"), a hospital-policy factor ("only doctors and medical researchers should see our information"), and a patient-consent factor ("medical researchers and grad students can see the information"). A privilege will be granted only for users satisfying all three conditions – in this case, medical researchers having an account on the local machine.

Privilege factors allow us to manage autonomy as a consequence of simple decisions. For example, suppose a local administrator wishes to restrict local execution. In our model, the local administrator only needs to assert positive privileges for his factor; if other factors change, the resulting full privileges will be adjusted automatically. Factors can be considered a structured way of implementing multiple signoff, but are managed essentially as ordinary privileges. Intuitively, they model desired autonomy, while recognizing that some factors are site-independent.

References

[RS] Rosenthal and Sciore, "Extending SQL Access Control to Derived and Distributed Data". www.cs.bc.edu/~sciore/papers/sql.pdf