ACStream: Enforcing Access Control Over Data Streams

Jianneng Cao #1, Barbara Carminati *2, Elena Ferrari *3, Kian-Lee Tan #4

#National University of Singapore, Singapore
1caojianneng@comp.nus.edu.sg
4tankl@comp.nus.edu.sg

*DICOM, University of Insubria, Varese, Italy
2barbara.carminati@uninsubria.it
3elena.ferrari@uninsubria.it

***Abstract*— In this demo proposal, we illustrate ACStream, a system built on top of StreamBase [1], to specify and enforce access control policies over data streams. ACStream supports a very flexible role-based access control model specifically designed to protect against unauthorized access to streaming data. The core component of ACStream is a query rewriting mechanism that, by exploiting a set of secure operators proposed by us in [2], rewrites a user query in such a way that it does not violate the specified access control policies during its execution. The demo will show how policies modelling a variety of access control requirements can be easily specified and enforced using ACStream.**

I. Overview

Data stream management systems (DSMSs) have been increasingly used to support a wide range of real-time applications that manage sensitive information. For instance, if you consider a battle field monitoring scenario, the positions of soldiers should only be accessible to the battleground commanders. Even if data are not sensitive, it may still be of commercial value to restrict their accesses. For example, in a financial monitoring service, stock prices are delivered to paying clients based on the stocks they have subscribed to. Hence, there is a need to integrate access control mechanisms into DSMSs.

Protecting data streams is quite different from protecting data stored in conventional DBMSs. As such, the many access control models and mechanisms that have been designed for traditional database systems [3] cannot be readily adapted to data stream applications. First, the long-running queries over unbounded data streams mean that access control enforcement is data-driven (i.e., triggered whenever data arrive). This implies that it is computationally expensive to enforce access control at query execution time, which is the common way to enforce access control in conventional DBMSs. Then, as data are streaming, temporal constraints (e.g., sliding windows) become more critical. Finally, operators are typically shared across multiple queries. Hence, access control can hardly be enforced at data stream or operator levels.

There is a further difference making the traditional access control not suitable for DSMSs. Indeed, in DBMSs the access control enforcement is based on the concept of authorized views. This implies that in order to grant the read privilege to a given user, say Bob, on a particular view, the security administrator (SA) has to create the corresponding view, say V1, and grant Bob the SELECT privilege on V1. As such, Bob is authorized to submit SELECT query on V1. Applying this enforcement in the context of DSMS will result in a very large number of views being generated. It is expected that the number of views generated in DSMS context will be very much larger than that in traditional DBMS. Indeed, the proposed access control model supports access control policies with temporal constraints. Thus, it is possible that for each specific time interval a user is allowed to see different parts of a stream or joint stream. As an example, consider the battle field monitoring scenario where it is reasonable to grant to each user/role a different view on the basis of the action taking place in the field. The possible huge number of views makes the views-based enforcement not suitable for DSMSs.

The goal of this demonstration is to present the main features of ACStream, a system for specifying and enforcing access control policies over data streams. The system is based on an expressive role-based access control model presented by us in [4] and specifically tailored to data streams. Relevant features of the model are the support for a wide range of access control granularity levels, specialized privileges for aggregate and join operations, as well as the support for various kinds of temporal constraints.

ACStream is based on the Aurora data stream model and algebra [5]. We have cast our work in the Aurora framework since there has yet no standard data model and query language for DSMSs and Aurora is up to now the most mature proposal. Moreover, Aurora has been successfully transferred to the commercial domain (i.e., the StreamBase engine [1]), and redesigned with distributed functionalities (i.e., Borealis [6]).

The ACStream access control enforcement mechanism is based on a set of novel secure operators that filter out from the results of the corresponding (non-secure) Aurora operators those tuples/attributes that are not accessible according to the specified access control policies (see [2] for their detailed description and formal semantics). Unlike conventional DBMSs, our access control mechanism operates at query definition time, and hence avoids run-time overhead. Whenever a user submits a query to the stream system, the reference monitor

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.25

checks the authorization catalog to verify whether the access can be partially or totally granted, or should be denied. In case of a partially authorized access request, the specified query is rewritten, by exploiting the secure operators, in such a way that it accesses only authorized data.

ACStream has a user-friendly GUI for both users and administrator: users are able to submit a query by simply considering the input streams without priori knowledge of the authorized views; for SA the specification of access control policies is as simple as defining a view in traditional DBMSs.

Data stream access control is a recent topic and few research has been so far performed in this area. Recently, Lindner and Meier proposed *Owner extended RBAC* (OxRBAC) security model to protect data streams from unauthorized accesses [7]. The main difference with respect to ACStream is that they apply a "post-processing" approach to enforce access control, according to which access control is performed only at the end of query specification. The main drawback of this approach is that many wasted computations may be performed, because it does not prune unauthorized tuples/attributes as early as possible. Moreover, it is possible for a user to remain "connected" to an output stream though he/she may not receive any output tuple (e.g., because his/her access rights have been revoked). By contrast, our mechanism statically rewrites the user query according to the specified policies before the query is registered into the system, it prunes unauthorized tuples as early as possible, and it avoids performing operations which at the end should be denied because of the specified access control policies. Additionally, the access control model on which ACStream is based is more expressive than the one illustrated in [7] in terms of the access requirements it can support. For instance, the mechanism described in [7] cannot handle certain queries that involve streams obtained from aggregating data over multiple streams. To the best of our knowledge ACStream is the first implemented prototype we are aware of supporting such a flexible and expressive access control model over data streams.

II. System Description

ACStream consists of two main components: a GUI for the specification and management of access control policies and a module performing query rewriting according to the specified access control policies. In the following, we briefly describe both of them. For details about the techniques please refer to the full technical paper [8].

A. Access control policies specification

The access control model adopted by ACStream is role-based and provides a wide range of granularity levels for access control. Objects to be protected are essentially *views* (or rather queries) over data streams. Access can be granted on whole data streams, as well as only on selected tuples and/or attributes of a data stream or selected attributes/tuples of joined streams. ACStream supports two types of privileges - Read privilege, for operations such as selection and projection of streams, and Aggregate privileges for operations such as Min, Max, Count, Avg, and Sum. In this way, it is possible to grant users the possibility of computing aggregate information over a data stream (e.g., the average of the heart beats of soldiers belonging to platoon X), without giving them the access to detailed data. Access to data streams can also be constrained to specified temporal intervals. Moreover, aggregate operations over the data may be constrained to specified time windows. This last feature is very relevant for data streams since we can customize aggregate operations according to data sensitivity, by regulating the size of the window over which the aggregate function can be computed. Moreover, the ACStream access control model allows one to specify policies that apply only on selected views resulting from the join of two or more streams. ACStream supports such wide range of granularity levels by using an approach similar to the one adopted by RDBMSs to define the protection objects on which a policy applies: define a view satisfying the access control restrictions and grant the access on the view instead of the base relations. However, since a standard query language for data streams has not yet emerged, we give a language independent representation of protection objects. Basically, we model a protection object by means of three components: (STRs,ATTs,EXPs), which correspond to the SELECT, FROM and WHERE clauses of a CREATE VIEW SQL statement. STRs denotes the streams over which the view is defined, ATTs denotes selected attributes within the streams composing the view over which the authorization is granted, and EXPs denotes the conditions, expressed in terms of predicates over the streams attributes, to be satisfied by the tuples belonging to the authorized view. Access control policies are stored into a system catalog, i.e., SysAuth, which is queried by the reference monitor (cfr. Figure 1) to decide the result of an access control request. ACStream provides a GUI to specify each component of an access control policy.

Table I presents an example of SysAuth catalog containing four access control policies defined for the Doctor role and referring to two streams: Position(ts, SID, Platoon, Pos) stream and Health (ts, SID, Platoon, Heart, BPressure) stream. In particular, SID and Platoon store soldier's and platoon's identifiers, respectively, both in the Position and Health streams, Pos contains the soldier's position, Heart stores the heart beats, whereas BPressure contains the soldier's blood pressure value.

The first access control policy in Table I authorizes doctors to access the position and id of soldiers belonging to their platoons (this condition is modeled as: Position.Platoon=self.Platoon[1]).

The second policy authorizes doctors to compute the average position of those soldiers not belonging to their platoons. This privilege is granted only during the time of action a. Moreover, this policy states that the average position can be computed only with windows of minimum 1 hour and with

[1]We assume that each user has an associated profile, i.e., a set of attributes modeling his/her characteristics, like for instance the platoon one belongs to.

sbj	protection object			priv	gtc	wtc	
	streams	attributes	expressions			s	o
Doctor	Position	Pos, SID	Position.Platoon=self.Platoon	read	-	-	-
Doctor	Position	Pos	Position.Platoon≠self.Platoon	avg	[start(a), end(a)]	1	1
Doctor	Health	*	Health.Platoon=self.Platoon	read	-	-	-
Doctor	Health, Position	BPressure, SID Pos	Health.Platoon≠self.Platoon ∧ Position.SID= Health.SID ∧ Pos≥Target(a)-δ ∧ Pos≤Target(a)+ δ	read		- - -	- - -

TABLE I

EXAMPLES OF ACCESS CONTROL POLICIES

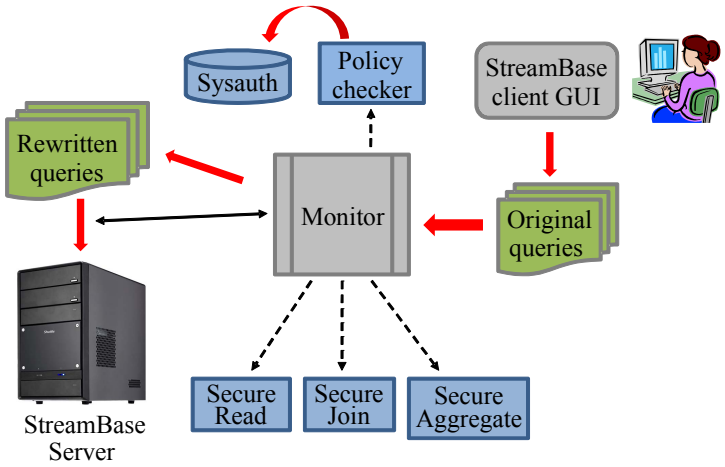

Fig. 1. `ACStream` architecture

1 hour as the minimum step. By the third policy doctors are authorized to monitor the health conditions (i.e., all attributes of Health stream) only of those soldiers belonging to their platoons. Finally, the fourth policy allows doctors to monitor blood pressure of those soldiers not in their platoons, but whose position is near to the target of action `a`, that is, whose position has a distance at most δ from action `a` target's position[2] (`Pos`$\geq$`Target(a)`$-\delta\wedge$`Pos`$\leq$`Target(a)`$+\delta$).

B. Access control enforcement

Aurora query processing exploits a data-flow paradigm, modelling queries as a loop-free direct graph of operations (i.e., boxes). Users specify queries by adding boxes to a graph. `ACStream` enforces access control at query definition time. Whenever a user tries to submit a query graph, the reference monitor checks the authorization catalog to verify whether the access can be partially or totally granted, or should be denied. In case of partially authorized access request, the graph is modified in such a way that it only outputs authorized data. The building blocks of the `ACStream` reference monitor are a set of *secure operators*, that filter out from the results of the corresponding (non-secure) Aurora operators those tuples/attributes that are not accessible by the requesting user according to the specified access control policies. The main secure operators are `Secure View`, `Secure Read`, `Secure Join` and `Secure Aggregate`. `Secure View` receives as input a stream and an access control policy and returns the view of the stream containing all and only those tuples/attributes which are accessible according to the policy's constraints. `Secure Read` makes use of the `Secure View` operator and returns, given a user u and a data stream S the view of S over which u can exercise the `read` privilege, according to the policies in `SysAuth`. Since `ACStream` allows the specification of policies for aggregate privileges, the suite of secure operators contains `Secure aggregate`. Given an aggregate operation over a stream S and a user u, this operator considers the policies specified over S for the requested aggregate operation that apply to u and returns the result of the aggregate operation only over the "view" authorized by these policies, by also taking into account the possible temporal constraints imposed by the policies. Finally, `Secure Join` is similar to `Secure Aggregate` but it takes into account policies that apply to joined streams. The graph corresponding to the user query is rewritten by using the secure operators to obtain a graph which outputs only authorized tuples.

[2]We assume a function Target() that returns the position of the target of a given action.

C. System architecture

ACStream has been implemented on top of StreamBase DSMS[3] [1]. In particular, as depicted in Figure 1, we exploit the Streambase GUI for users to express graph queries. Before submitting a query the user needs to log on to ACStream, which assigns to him/her the corresponding role(s). For internal processing, each query is then encoded into an XML document, where each box of the graph is translated into an XML node containing information on the input stream(s), the performed operations and the output streams(s).

```
1. <box name="filter_pos"   type="filter">
2.     <property name="last-editor-tab"   value="1"/>
3.     <input port="1"   stream="Position"/>
4.     <output port="1"   stream="out:filter_pos_1"/>
5.     <param name="expression.0"   value="pos&gt;100"/>
6.     ...
7. </box>
```

Fig. 2. XML encoding of an Aurora box

Let us consider a simple graph query to retrieve the information of soldiers whose position is greater than 100. The corresponding graph consists of a filter box applied on Position stream, to prune any tuple that does not satisfy the constraint on attribute Position. Figure 2 illustrates the XML node encoding this filter box[4].

Once the user submits the query, the Monitor parses the XML file of the query into a graph using DOM [9]. Then, it traverses the graph box by box recursively from the final output stream (the end) to the source streams (the start), and for each box it calls the Policy checker component to determine what policies in Sysauth have to be enforced. This is quite easy when the box is applied to an input stream; however, it is more difficult when the box is applied to an *internal stream*, that is, a stream output by another box. To facilitate the latter task, we model each internal stream *S*' with a protection object like representation, similar to the one used for access control policy specification. This makes it easier to determine the policies to be considered for the considered box. Thus, we model an internal stream as a triple (STRs,ATTs,EXPs). In general, if the box is applied to a single input stream, Monitor makes use of the Secure Read operator to enforce access control policies retrieved by Policy checker. If the box is applied to a single *internal stream*, it is not necessary to enforce any policy since all unauthorized tuples/attributes have already been pruned by previously inserted Secure Read operator. However, if the box denotes a join operator, whether it is applied on input streams, or internal streams, or a mixture of them, it is necessary to call Policy checker to determine the policies to be considered. These policies are then enforced by means of the Secure Join operator. The process for an aggregate box is similar to that for a join box. For any of other cases, we do not need to enforce any policy.

After the traversal and the enforcement of access control policies, the Monitor generates a new XML document encoding the rewritten query. The resultant query is then submitted to StreamBase Server for execution.

[3]StreamBase is a commercial product exploiting the Aurora data stream engine.

[4]In XML, '>' is the encoding of '>', so line 5 has to be read as pos>100.

III. Demonstration

The goal of the ACStream demonstration is to show the specification and enforcement of access control policies on data streams. The demonstration is based on the protection of streams in the military domain. The demonstration starts by showing how access control policies like those in Table I can be specified using the ACStream specification tool. Then, we run the ACStream reference monitor by using a sample SysAuth catalog for the target streams, containing policies modelling a variety of access control requirements for different roles (e.g., captains, soldiers, doctors) in the considered domain. In particular, we demonstrate the execution of each secure operator and how the operators can be used to rewrite the input query graph on the basis of the required box and the specified access control policies. We also show examples of detection of violations of the access control policies and the corresponding denial of access.

Acknowledgment

Jianneng Cao and Kian-Lee Tan are partially supported by a research grant R-252-000-307-112 funded by the National University of Singapore.

References

[1] Streambase home page. [Online]. Available: http://www.streambase.com/
[2] B. Carminati, E. Ferrari, and K.-L. Tan, "Enforcing access control over data streams," in *Proc. of 12th ACM Symposium on Acccess Control Models and Technologies (SACMAT)*, Sophia Antipolis, France, 2007, pp. 21–30.
[3] E. Ferrari and B. Thuraisingham, *Advanced Databases: Technology and Design*, 1st ed. London: Artech House, 2000, ch. Secure database systems.
[4] B. Carminati, E. Ferrari, and K.-L. Tan, "Specifying access control policies on data streams," in *Proc. of Database System for Advanced Applications Conference (DASFAA)*, Bangkok, Thailand, 2007, pp. 410–421.
[5] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Aurora: a new model and architecture for data stream management," *VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.
[6] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik, "The design of the borealis stream processing engine," in *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2005, pp. 277–289.
[7] W. Lindner and J. Meier, "Securing the borealis data stream engine," in *Proc. of International Database Engineering and Application Symposium (IDEAS)*, Dehli, India, 2006, pp. 137–147.
[8] B. Carminati, E. Ferrari, J. Cao, and K.-L. Tan, "A framework to enforce access control over data streams," *To appear in ACM Transactions on Information & System Security (TISSEC)*, 2008.
[9] Document object model. [Online]. Available: http://www.w3.org/DOM/