Imposing Fine-grain Next Generation Access Control over Database Queries

David Ferraiolo, Serban Gavrila, Gopi Katwala, and Joshua Roberts

National Institute of Standards and Technology
Gaithersburg, Maryland 20899
{dferraiolo, Serban.gavrila, gopi.katwala, joshua.roberts}@nist.gov

ABSTRACT

In this paper, we describe a system that leverages ANSI/INCITS Next Generation Access Control (NGAC) standard called Next-generation Database Access Control (NDAC) for accessing data in tables, rows, and columns in existing RDBMS products. NDAC imposes access control at the data level, eliminating the need for implementing and managing access control in applications and/or through the use of proprietary RDBMS mechanisms. Consequently, the same policies can protect multiple databases from queries sent from multiple applications. Furthermore, NDAC not only provides control down to the field level, but to varying fields of select rows. NDAC is unique in achieving this granularity of control without the use and coordination of multiple ptotection mechanisms. Operationally, users issue wide sweeping queries, and NDAC allows access to the optimal amount of data permissible for the user. The method includes an Access Manager for trapping and enforcing policy over SQL queries issued by applications as well as a Translator for converting SQL statements to NGAC inputs and converting NGAC authorization responses to either an access Deny or one or more permitted SQL statements.

Keywords

ABAC; NGAC; Policy Machine; DBMS; Access Control

1. INTRODUCTION

Relational Database Management Systems (RDBMSs) do not typically impose access control directly on its data. To restrict access to sensitive data that might reside in a RDBMS, controls are typically implemented at the application level or through propriety RDBMS methods such as views. These controls take on many forms including role-based access to "screens" with parameters that can be characterized and subsequently used to formulate and issue SQL queries. SQL queries comprise four basic types of operations–Select, Insert, Update, and Delete–that respectively read, create, write, and delete data in tables. An important feature of RDBMSs is that they are able to specify criteria and extract and/or alter data that might reside in one or more tables with great efficiency. For example, "give me all the employees over 50 years old that live in Virginia".

ABAC'17, March 24 2017, Scottsdale, AZ, USA
ACM 978-1-4503-4910-9/17/03
DOI: http://dx.doi.org/10.1145/3041048.3041050

In this paper we describe a method that leverages ANSI/INCITS Next Generation Access Control (NGAC) standard [1] [2] called Next-generation Database Access Control (NDAC) for imposing access control over database queries at the data level, independent of the application and with minimal impact on performance. As a result, the same policies can protect multiple databases from queries sent from multiple applications.

NDAC's method of protection begins with automatically generated composite objects in the form of object attributes from a database schema and the expression of access control policies in terms of those attributes. NDAC uses NGAC as an authorization engine to manage access control policies (through its Policy Administration Point (PAP)) and compute authorization responses (through its Policy Decision Point (PDP)). [3] provides an open source for such an engine. The method also includes an Access Manager (a customized NGAC Policy Enforcement Point (PEP)) for trapping and enforcing policy over SQL queries issued by applications and a Translator for converting SQL statements to NGAC inputs and converting NGAC authorization responses to either an access Deny or one or more permitted SQL statements.

Furthermore, NDAC provides control down to the granularity of select rows with varying fields. Operationally, users issue wide sweeping queries, and NDAC allows access to an optimal set of permissible data. Although other technologies (see section 6–Related Work) achieve a similar granularity of protection through the combined use of multiple protection mechanisms, NDAC is unique in its use of just one policy store. The principle advantage is that NDAC does not need to maintain and coordinate multiple access control schemes and can use the same policy store to protect non-RDBMS resources, such as files, using an NGAC standards PEP.

To demonstrate viability and assess performance, we have created an NDAC prototype/experimental implementation using Harmonia 1.6–an NGAC reference implementation that uses MySQL for its access control database [3]. For purposes of computing a decision or reviewing access rights, all information that is needed resides in memory. Harmonia 1.6 access control information is loaded from disk into memory when the PDP is initialized and updated when an administrative change occurs.

The remainder of this paper focuses on the method rather than the NDAC prototype/experimental implementation due to its early stage of development.

2. NGAC OVERVIEW

The Policy Machine (PM) [4] is an access control framework that

served as the basis for the development of an ANSI/INCITS standard call Next Generation Access Control (NGAC). NGAC consists of:

- a standard set of data elements and relations that can be configured to express arbitrary access control policies in support of a wide variety of data services and applications
- a generic set of operations that include read/write operations that can be performed on resource data as well as administrative operations for configuring (creating and deleting) the data elements and relations that represent policies
- a standard set of functions for computing access control decisions and enforcing policy over user access requests to perform read/write and its administrative operations

NGAC is a flexible access control framework in that it can be molded in support of multiple combinations of diverse access control policies. NGAC can often provide much of the same data service functionality that is supplied by existing application products and system utilities (e.g., file management, workflow, internal messaging) and with similar performance [5]. An advantage of NGAC is that access control policies are comprehensively enforced over its data services, while non-NGAC data service counterparts lack such faculties. Although it is possible to develop a NGAC relational DBMS data service with features similar to today's commercially available RDBMS products, the NGAC data service performance would pale in comparison. Furthermore, the NGAC-enabled RDBMS data service could not directly accommodate the broadly recognized SQL standard for accessing databases.

3. NDAC

NDAC provides a means of leveraging NGAC for expression and enforcement of access control policies over SQL queries for accessing data in tables, rows, and columns in existing RDBMS products. By leveraging NGAC, the method provides a means of access control policy support that goes beyond state-of-the-art with minimal impact on performance. It can impose forms of mandatory, discretionary, and history-based access control policies [6]. Architecturally, NDAC could be deployed externally to RDBMS, thereby providing a general solution for a variety of RDBMS products, or it could be implemented as a database-kernel module.

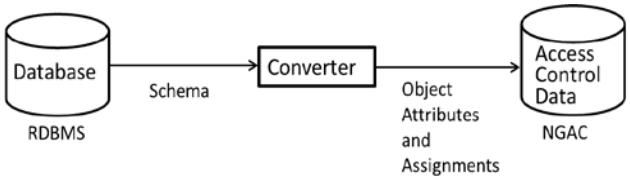

Figure 1. Converting Database Schema to NGAC Access Control Data

Included among NGAC's data elements and relations used to express and enforce policies are Object Attributes. Object Attributes are containers that group and characterize data objects in diverse ways. Data objects and object attributes are placed into containers through an assignment relation. Vis. Figure 1, the NDAC process for expressing access control policies begins with an existing RDBMS schema, which includes columns and tables that are automatically converted into NGAC-corresponding object attributes and assignments.

Given that rows are also object containers, existing rows could be automatically converted as well. NGAC data elements and relations also include User Attributes, a generic set of operations, and three types of relations for specifying an access policy. Once the RDBMS schema has been converted, NGAC relations are configured in formulating policy in terms of the created object attributes and assignments using NGAC's administrative API. The resulting data elements and relations are stored as NGAC Access Control Data. In addition to the conversion and supplementary data elements and relations, NDAC includes an Access Manager for trapping SQL queries from applications, a Translator for converting SQL queries along with a user identity to NGAC inputs, and NGAC authorization responses to those inputs to either an access deny or permitted SQL queries.

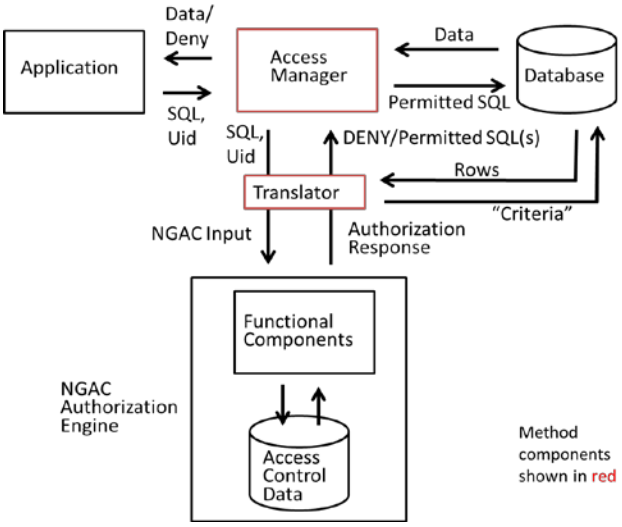

Figure 2. Placement of NDAC with respect to existing components

Figure 2 shows the placement of NDAC's Access Manager and Translator in an authorization flow that involves Applications, a target Database, and an NGAC authorization Engine. The authorization flow is as follows:

(1) The SQL statement from a user of the Application is intercepted by the Access Manager and sent to the Translator.
(2) For Select, Update, and Delete statements, using a separate transaction, identify the set of rows that meet the criteria included in the SQL statement. For Insert, this step is not used.
(3) The Translator converts the SQL statement from the user into NGAC inputs that are fed to an NGAC implementation (engine).
(4) Using its Access Control Data, and the rows identified in (2), the NGAC implementation computes and renders an Authorization Response that is sent back to the Translator.
(5) The Translator converts the Authorized Response into either an access DENY or one or more SQL Statements that are permitted for the user and sent back to the Access Manager.
(6) The Access Manager submits the Permitted SQL Statements to the Database.
(7) In the case of a Select operation, Data extracted from the database is sent back to the Access Manager and forwarded to the Application and user.

Depending on the type of query (Select, Update, Insert, or Delete) the Translator issues different inputs to the NGAC Authorization

Engine. These details are discussed later in the paper.

4. EXPRESSING POLICIES

4.1 Basic Elements, Containers, and Relations

NGAC access control data includes users, data objects, generic operations, and user and object attributes among its elements. NGAC treats both user attributes and object attributes as containers. Containers are instrumental in both formulating and administering access policies and attributes. NGAC expresses access policies through configurations of relations that include assignments (define membership in containers), associations (to derive privileges), and prohibitions (exceptions to privileges).

User attribute containers characterize their members. These containers can represent user names, roles, affiliations, or other common characteristics pertinent to policy such as security clearances.

Object attribute containers characterize data by identifying collections of objects such as those associated with certain projects, applications, or security classifications. Object containers can also represent tables, columns, and rows.

NGAC uses a tuple (x, y) to specify the assignment of element x to element y. The assignment relation always implies containment (i.e., x is contained in y).

Users and objects may be contained in one or more containers, and containers may be contained by or contain other containers of the same type. For object containers, this allows for the representation of complex data structures such as relational database tables with distinguished fields. Rows of a table may be expressed as containers of data objects corresponding to the row's fields, and columns may be expressed as containers of data objects corresponding to column fields. Figure 3(b) illustrates a table using ovals to represent containers and dots to represent individual data objects. The vertically oriented ovals represent columns (Name, Phone, SSN, and Salary), the horizontally oriented ovals represent rows (AliceRecord, BobRecord, and TomRecord), and their intersections represent fields in one or more tables. Figure 3(b) further illustrates a container of rows (Gr2Records) and two containers of columns (Public and Sensitive). All rows and all columns are represented by the object container EmployeeTable.

Note that for this example, the containers shown in red are the object attributes that were automatically created by the Converter (see figure 1). All other NGAC elements and relations are assumed to be created through an NGAC administrative API by an authorized user. This authorized user may be a policy administrator or, as we discuss later, the user submitting Insert or Delete SQL queries.

Figure 3(a) illustrates user containers (also called user attributes) for the grouping and characterization of users. The container named Staff includes three users (u1, u2, and u4), and the container HR includes two users (u3, and u5). Employee is a container of containers (HR and Staff). In addition, figure 3(a) shows three containers—Bob, Alice and Tom—that respectively contain u1, u2, and u4. Finally, figure 3(a) shows Gr2Mng containing user u2.

NGAC recognizes a generic set of operations that include basic input and output operations (i.e., read and write) that can be performed on the contents of data objects as well as a standard set of administrative operations that can be performed on NGAC data elements and relations that represent policies and attributes.

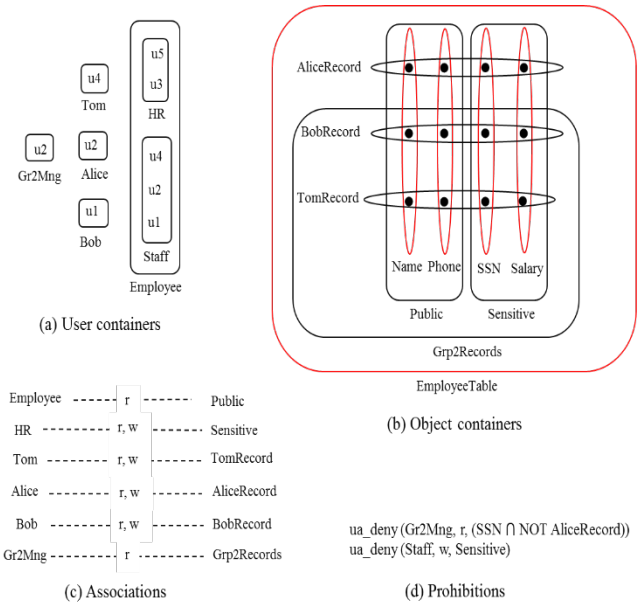

Figure 3. Example Policy Configuration

To carry out an operation, one or more access rights are required. As with operations, two types of access rights apply: non-administrative access rights and administrative access rights.

4.2 Associations

Access rights to perform operations are acquired through associations. An association is a triple, denoted by *ua---ars---pe*, where *ua* is a user attribute, *ars* is a set of access rights, and *pe* is a policy element that may comprise either a user attribute or an object attribute. The policy element *pe* in an association is used as a reference for itself and the policy elements contained by the policy element. The meaning of the association *ua---ars---pe* is that the users contained in *ua* can execute the operations enabled by the access rights in *ars* on the policy elements referenced by *pe*. The set of referenced policy elements are dependent on (and meaningful to) the access rights in *ars*.

Figure 3(c) lists six association relations in terms of the user and object attributes (containers) illustrated in figures 3(a) and 3(b). The set of referenced policy elements are dependent on the access rights in *ars*. Note that the policy element of each association is an object attribute and the access rights are read/write. In the association HR---{r, w}---Sensitive, the policy elements referenced by Sensitive are data objects (dots) contained in Sensitive, meaning that user u3 and u5 can read and write those objects. If we had an association HR---{create assign-to}---Sensitive, where "create assign-to" is an administrative access right, then the policy elements referenced by Sensitive would be Sensitive, SSN, and Salary, meaning that users u3 and u5 may create assignments to Sensitive, SSN, or Salary.

The access policy specified by the list of associations in figure 3(c) is as follows:

- Employee users can read Name and Phone fields of all records in EmployeeTable

- In addition to being able to read Name and Phone fields, HR users can read and write SSN and Salary fields of all records in EmployeeTable
- Bob, Tom, and Alice can read and write all fields (SSN, Salary, Name, and Phone) in their own record (respectively, BobRecord, TomRecord, and AliceRecord)
- Gr2Mng can read all fields (SSN, Salary, Name, and Phone) of all records in Gr2Reccords (i.e., BobRecord and TomRecord)

4.3 Prohibitions

In addition to assignments and associations, NGAC includes three types of prohibition relations. In general, prohibition relations specify privilege exceptions. One of these relations is user attribute-deny. The user attribute-based deny relation is denoted by ua_deny(*ua*, *ars*, *pes*), where *ua* is a user attribute, *ars* is an access right set, and *pes* is a policy element set used as a reference for policy elements contained by the policy element(s). The meaning of the relation is that the users assigned to *ua* cannot execute the operations enabled by the access rights in *ars* on the policy elements in *pes*.

Figure 3(d) lists two prohibitions. The first prohibition specifies that users assigned to Gr2Mng cannot read objects in SSN with the exception of objects in AliceRecord. The second prohibition specifies that users assigned to Staff cannot write to objects in Sensitive.

The prohibitions listed in figure 3(d) further constrain the access policy as follows:

- Staff users can read Name and Phone fields of all records in EmployeeTable
- In addition to being able to read Name and Phone fields, HR users can read and write SSN and Salary fields of all records in EmployeeTable
- Bob, Tom, and Alice can read all fields (SSN, Salary, Name, and Phone) and write to Name and Phone fields in their own record (respectively, BobRecord, TomRecord, and AliceRecord)
- Gr2Mng can read all fields of all records in Gr2Records with the exception of the SSN field

An example set of Employee Records with data content is shown in the top table of figure 4 under the object containers depicted in figure 3(b). The bottom three tables show the access capabilities for users u1, u2, and u3 under the access control policy expressed in figure 3, where read access is highlighted in black, and read/write access is highlighted in red.

5. TRANSLATOR

As discussed in section 3, the NDAC includes a Translator. On one side, the Translator converts an SQL statement generated by an application and the identity of the application's user to an NGAC input. On the other side, the Translator takes an NGAC authorization response to the input and converts it to either one or more permitted SQL statements; an access DENY in the case of a Select statement; or to a GRANT or DENY status in the case of an Update, Insert, or Delete statement. The Translator treats Select and Update operations differently than Insert and Delete Operations since Select and Update operations are directly mapped to NGAC read and write operations on data. Alternatively, Insert and Delete operations are mapped to create and delete administrative operations on NGAC object containers that correspond to rows.

Some Employee Records

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067	304-75-3995	$62,550

u1 (Bob, Staff)

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042		
Tom	301-976-2067		

u2 (Alice, Staff, Gr2Mng)

Name	Phone	SSN	Salary
Bob	301-976-4454		$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067		$62,550

u3 (HR, Staff)

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067	304-75-3995	$62,550

Figure 4. Example set of records with data content and the access capabilities for users u1, u2, and u3 under the access control policy of figure 3

5.1 Select and Update

Select SQL statements include a specification of one or more tables and one or more columns from those tables along with criteria for identifying rows from the table(s). Update SQL statements include a specification of one table with one or more columns with criteria for identifying rows. The method for translating a user's requested Select statement to one or more permitted SQL statements or an Update statement to a GRANT or DENY result is based on NGAC's ability to review the access capabilities of users. See [7] for a linear time algorithm and method for reviewing NGAC user capabilities. In particular, NDAC identifies a set of objects that are accessible to a user for either read for Select or write for Update as well as attributes that contain those objects. In the algorithms that follow, the terms "row," "column," and "table" refer to object attributes that correspond to those entities. Possible algorithms for Select and Update are as follows:

For Select:

(1) Using a separate transaction, identify the set of rows in the SQL database that meet the criteria included in the Select SQL statement.

(2) For each row identified in (1), identify a maximal set of columns that are a subset of the columns in the Select statement, and each identified column contains an object (for which the user has Read access) that is also contained in the row. These columns are said to be associated with the row.

(3) For each row, column association, remove the columns that are also included in any DENY relation for the user with respect to Read.

(4) For each subset of identified rows so that each row in the subset has a common associated set of columns, generate a Select SQL statement for that set of columns with the original table and original condition augmented by a condition that limits the Select to the subset of identified rows.

(5) If the set of rows or columns are empty, the Translator issues a DENY response.

For Update:

(1) Identify the set of rows in the SQL database that meet the criteria included in the Update SQL statement.
(2) Identify a set of rows in the table of the Update SQL statement containing objects accessible by the user under the write operation.
(3) If the rows identified by (1) are a subset of those identified in (2), proceed to (4). Otherwise, DENY access.
(4) For each row identified in (1), verify the existence of objects common to the row and the set of columns included in the SQL Update statement. If the condition fails, DENY access. Otherwise, proceed to (5).
(5) For the columns included in the SQL Update statement, verify that the columns are not included in any deny relation for the user. If the condition holds, GRANT the SQL Update Statement. Otherwise, DENY access.

To provide a sense of potential performance, preliminary data shows that the NDAC prototype/experimental implementation currently computes and displays the results of authorizations of 100 records with 6 fields in 4 seconds and 1,000 records in 40.2 seconds.

5.2 Delete and Insert

The execution of an SQL Delete statement removes one or more rows from a table in accordance with criteria included in the statement. NDAC Grants or Denies a user's request to delete one or more rows in a database table, and, in the case of a Grant, subsequently deletes the corresponding NGAC object attributes and relations. The execution of a SQL Insert statement creates a new row with specified column values in a specified table. The method either Grants or Denies a user's request to insert a row in the database, and, in the case of a Grant, subsequently creates an NGAC object attribute corresponding to the row, creates objects (representing the values), and assigns those objects to the row attribute and appropriate column attributes. A user's capability to perform an SQL Delete or Insert operation is dependent on the existence of administrative privileges.

The creation and deletion of objects, object attributes, and assignments is achieved through the execution of administrative operations. A user's capabilities to execute administrative operations are established through administrative privileges.

5.2.1 Administrative Operations

Administrative operations in NGAC are implemented using parameterized routines, prefixed by a precondition, with a body that describes how a data set or relation (denoted by Y) changes to Y′. The precondition tests the validity of the actual parameters. If the condition evaluates to false, then the routine fails:

Rtnname (x_1, x_2, …, x_k) {
…*preconditions*…
{
Y′= f(Y, x_1, x_2, …, x_k)
}

Consider as an example the administrative operation CreateOinOA shown below, which specifies the creation of an object x and assigns the object to an object attribute y. The preconditions here stipulate that the x parameter is not a member of objects (O), and the y parameter is a member of object attributes (OA). The body describes the addition of the x to the set of objects (O), which changes the state of the set to O', and the addition of the tuple (x, y) to the set of assignments (ASSIGN) relation, which changes the state of the relation to ASSIGN'.

CreateOinOA(x, y)
x ∉ O ⋀ y ∈ OA
{
O' = O ⋃ {x}
ASSIGN' = ASSIGN ⋃ {(x, y)}
}

Each administrative routine entails a modification to the NGAC configuration.

5.2.2 Administrative Privileges

In order to execute an administrative operation, the requesting user must possess appropriate access rights. Just as access rights to perform read/write operations on data objects are defined in terms of associations, so too are capabilities to perform administrative operations on policy elements and relations.

For example, consider the following two associations in support of the configuration depicted by Figure 3(b):

TableAdmin---{*create-oa*, *create-o*, *create ooa*}---EmployeeTable

TableAdmin---{*delete-o*, *delete-oa*, *delete-ooa*, *delete-oaoa*}---EmployeeTable

The meaning of the first association is that a user assigned to TableAdmin can:

(1) create an object attribute (e.g., corresponding to a row) assigned to an object attribute (e.g., EmployeeTable) in EmployeeTable
(2) create an object assigned to an object attribute (e.g., an existing row) in EmployeeTable
(3) create an object to object-attribute assignment from an object (e.g., an object in a row) to an object attribute (e.g., corresponding to a column) in EmployeeTable

The meaning of the second association is that a user assigned to TableAdmin can:

(1) delete an object to object-attribute assignment (e.g., delete object assignments to attributes corresponding to a row and column) in EmployeeTable
(2) delete an object in EmployeeTable
(3) delete an object-attribute to object-attribute assignment (e.g., a row assigned to EmployeeTable) in EmployeeTable
(4) delete an object attribute (e.g., corresponding to a row) in EmployeeTable

5.2.3 Administrative Routines

The administrative operations necessary to insert or delete an object container corresponding row in another object container

corresponding to a table do not need to be executed on an individual basis, but instead can be executed as an NGAC administrative routine.

An *administrative routine* consists mainly of a parameterized interface and a sequence of administrative operation invocations. The body of an administrative routine is executed as an atomic transaction—an error or lack of user privileges that causes any of the constituent operations to fail execution subsequently causes the entire routine to fail, producing the same effect as though none of the operations were ever executed.

The following routine (in the context of figure 3(b)) creates an object attribute (corresponding to a row) assigned to EmplyeeTable, creates new objects (corresponding to values), and assigns those objects to object attributes (corresponding to columns) and the object attribute corresponding to the row. Assume the columns Name, Phone, SSN, and Salary already exist and are assigned to the object attribute EmployeeTable.

```
Insert_Row_in_EmployeeTable(row, name, phone, ssn, salary)
{   CreateOAinOA(row, EmployeeTable)
    CreateOinOA(name, row)
    Assign(name, Name)
    CreateOinOA(phone, row)
    Assign(phone, Phone)
    CreateOinOA(ssn, row)
    Assign(ssn, SSN)
    CreateOinOA(salary, row)
    Assign(salary, Salary)
}
```

Although the Insert routine applies to the object attributes corresponding to the example schema of figure 3, a similar and corresponding routine could automatically be created for each table of an RDBMS schema, or a generic Insert routine could exist that uses a template specific to each table.

An administrative Delete routine could be used to delete an object attribute, objects and assignments corresponding to a RDMBS row, and column values. Consider, for example the following routine in the context of figure 3(b):

```
Delete_Row_from_EmployeeTable(row)
{    For each object obj in row {
       DeleteO (obj) /*includes deletion of assignments of obj*/
     }
     DeleteOAinOA(row, EmployeeTable) /*includes deletion
           of assignments row to EmployeeTable*/
}
```

Similar to Insert, a Delete routine could automatically be created for each table of an RDBMS schema, or a generic Delete routine could exist that uses a template specific to each table.

Administrative routines not only allow for consistence between RDBMS rows and corresponding NGAC object attributes, objects, and assignments, but also provide a means for testing a user's authority to Insert and Delete RDBMS rows.

For Insert:
The algorithm for translating an Insert statement to an NGAC authorization response assumes the existence of an NGAC administrative Insert routine. The algorithm is as follows:

(1) Invoke the routine corresponding to the table specified in the Insert statement using the identity of the user that issued the Insert statement with the specified row and column values, thereby creating an object attribute that corresponds to the row as well as objects that represent and correspond to column values that are assigned to the row and are appropriately assigned to object attributes that correspond to columns.
(2) If the routine successfully executes, GRANT the SQL Insert statement. Otherwise, DENY access.

For Delete:
The algorithm for translating a Delete statement to an NGAC authorization response assumes the existence of an NGAC administrative Delete routine, particularized for the referenced table. The algorithm is as follows:

(1) Identify the set of rows in the SQL database that meet the criteria included in the Delete SQL statement.
(2) For each row identified in (1), sequentially invoke, using the identity of the user that issued the statement, the Delete routine of the table specified in the Delete statement by caching the parameters of the object attribute corresponding to the identified row and the objects contained in the object attribute.
(3) If any invocation of the routine fails to successfully execute, DENY the SQL Delete statement, and roll back changes due to previous invocations by applying the cache as NGAC administrative Insert routine parameters. Otherwise, GRANT.

6. Related Work

NDAC is not the only system for enforcing fine-grain access control policies over database queries in support of applications. Two others are Oracle's Real Application Security (RAS) [8] and Axiomatics' Data Access Filter (ADAF) [9]. Both are designed to intercept and modify SQL statements for the purpose of applying rule-based controls in database access scenarios.

RAS Allows application developers to define a data security policy, application roles, and application users. At the application layer, security policies are defined in terms of Access Control Lists on dynamically created (using a "where" clause) Data Realms (set of rows) and static "Views" on columns using the RAS API. In effect, control is provided down to the record/field level.

ADAF includes a proxy that intercepts SQL statements, which in turn are sent to an ADAF engine. The engine employs two policy enforcing capabilities. First, a "where" clause is computed and added to the SQL statement, thereby filtering out rows for which the user is not authorized. This filtering operates on XACML 3.0 [10] policies in terms of object attributes created to correspond to the tables and columns of the database schema. Second, ADAF uses Masking to further redact individual cells of the filtered rows, thereby providing filtering down to the record/field level.

NDAC has a number of similarities and differences with RAS and ADAF. The RAS protection scheme is application centric and DBMS specific, while ADAF and NDAC allow the same policies to protect multiple databases from queries sent from multiple applications. ADAF and RAS policies for controlling access to rows are fully dependent on the database schema definition, while NDAC is not. NDAC can define object attributes that contain schema related object-attributes (e.g., Public and Sensitive of figure

3(b)) and express policies in terms of those object attributes. This is an important distinction because enterprise policies are fluid and change over time while schemas are ridged and typically remain fixed.

In contrast to ADAF and RAS, NDAC does not need to maintain and coordinate policies of two access control schemes to achieve fine-grain access control. Although NDAC is shown external to the DBMS, its policies are expressed in terms of relations like RAS, allowing NDAC to be implemented as a database-kernel loadable module.

Since ADAF is based on XACML, it is not amenable to policy review, while RAS and NDAC can query the rule configuration (relations) to determine the tables, rows, and columns accessible to a given user in advance (without computing a decision). Moreover, NDAC can graphically visualize the overall set of rules. See [7] for NGAC algorithms and techniques for efficient policy review and visualization.

7. CONCLUSION

This paper describes a system that leverages a ANSI/INCITS Next Generation Access Control (NGAC) standard called Next-generation Database Access Control (NDAC) for accessing data in tables, rows, and columns in existing RDBMS products. NDAC imposes access control at the data level and eliminates the need for implementing and managing access control in applications, and/or through the use of proprietary RDBMS mechanisms. As a consequence, the same policies can protect multiple databases from queries sent from multiple applications. Furthermore, NDAC does not only provide control down to the field level, but to the level of varying fields of select rows. Although other technologies achieve a similar granularity of protection through the combined use of multiple protection mechanisms, NDAC is unique in its use of just one policy store. Operationally, users issue wide sweeping queries and NDAC allows access the optimal amount of data permissible for the user.

The NDAC process for expressing access control policies begins with an existing RDBMS schema that includes columns and tables that are automatically converted into NGAC corresponding object attributes and assignments. Since rows are also object containers, existing rows can automatically be converted as well. NGAC data elements and relations also include User Attributes, a generic set of operations, and three types of relations for specifying an access policy. Once the RDBMS schema has been converted, NGAC relations are configured in formulating policy in terms of the created object attributes and assignments using NGAC's administrative API. The resulting data elements and relations are stored as NGAC Access Control Data. In addition to the conversion and the additional data elements and relations, NDAC includes an Access Manager for trapping SQL queries from applications and a Translator for converting SQL queries along with a user identity to NGAC inputs and NGAC authorization responses to those inputs to either an access Deny or permitted SQL queries that are sent to the RDBMS for policy preserving access.

The U.S. Government has filed a patent application of certain aspects of the subject matter disclosed in this paper.

Disclaimer: *Products may be identified in this document, but identification does not imply recommendation or endorsement by NIST, nor that the products identified are necessarily the best available for the purpose.*

8. REFERENCES

[1] Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA), INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.

[2] American National Standards Institute, Information technology – Next Generation Access Control – Generic Operations and Data Structures (GOADS), INCITS 526-2016, American National Standard for Information Technology, January 2016.

[3] NIST Policy Machine Versions 1.5 and 1.6 - Harmonia [Website], https://github.com/PM-Master [accessed 9/26/16].

[4] D. Ferraiolo, S. Gavrila, and W. Jansen, "Policy Machine: features, architecture, and specification," National Institute of Standards and Technology, Internal Report 7987 Rev. 1, 2015.

[5] D. Ferraiolo, S. Gavrila, and W. Jansen, "On the Unification of Access Control and Data Services," in *Proceedings of the 2014 IEEE 15th International Conference of Information Reuse and Integration*, IEEE, 2014, pp. 450 – 457. http://dx.doi.org/10.1109/IRI.2014.7051924

[6] D.F. Ferraiolo, V. Atluria, and S.I. Gavrila, "The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 412-424, April 2011. http://dx.doi.org/10.1016/j.sysarc.2010.04.005

[7] P. Mell, J. Shook, S. Gavrila, Restricting Insider Access through Efficient Implementation of Multi-Policy Access Control Systems. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*. Vienna, Austria, October 24-26, 2016.

[8] http:/www.oracle.com/technetwork/database/security/real-application-security/real-application-security-1964775.html.

[9] https:/www.axiomatics.com/resources/102-data-sheets/431-axiomatics-data-access-md-filter-data-sheet.html.

[10] The Xtensible Access Control Markup Language (XACML), Version 3.0, OASIS Standard, January 22, 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf