IBM Software Group

Overview of Label-Based Access Control (LBAC) in DB2 for Linux, Unix and Windows (LUW)

Walid Rjaibi
wrjaibi@ca.ibm.com
October 12, 2006

DB2 Information Management Software

e business on demand software

Agenda

- Introduction
- Basic Concepts
- Protected Tables
- Row Level Labeling Example
- LBAC and Constraints
- LBAC and Query Optimization
- LBAC and Data Partitioning
- Security Model Extensions
- Summary

Introduction

- Label-Based Access Control (LBAC) is an implementation of Mandatory Access Control (MAC)
  - In the context of database systems, LBAC is commonly referred to as the ability to control access to data rows based on security labels.
- The DB2 LBAC solution is a *flexible* implementation of MAC at both:
  - The Row level
  - The Column level
- Row level and column level protection are orthogonal and can be used either separately or combined
- Table level labeling can be easily simulated using column level labeling
- LBAC complements the traditional DB2 Discretionary Access Control (DAC)
  - When a protected table is accessed, DB2 enforces two (2) levels of access control:
    1. DAC at the table level, i.e., does the user hold the required privilege to perform the requested operation on the table?
    2. LBAC, which can be at the row level, column level or both

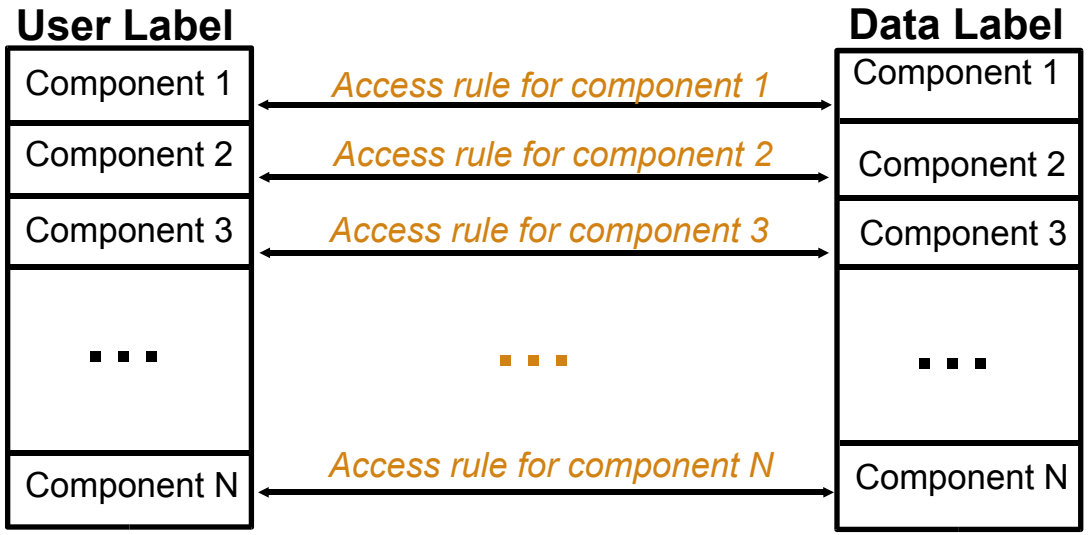
Basic Concepts

- Security Label Component
  - Represents any criteria upon which you would like to control access
  - Three (3) types are supported:
    - Array: An ordered set, e.g., level = [‘HS', 'S', 'C', ‘NS']
    - Set: A set, e.g., Projects = {'A', 'B', 'C', 'D'}
    - Tree: A hierarchy, e.g., departments = {G1, G2, G3}, where G1 is "root", and G2, G3 are children of G1
- Security Policy
  - Defines a security label type and access rules.
  - The access rules are predefined within DB2 and collectively referred to as DB2LBACRULES
    - Read access rules: They apply when data is retrieved (data is retrieved on SELECT, UPDATE and DELETE SQL statements)
    - Write access rules: They apply on INSERT, UPDATE and DELETE SQL statements
  - Example

```
CREATE SECURITY POLICY mySecPolicy
COMPONENTS level, Projects WITH DB2LBACRULES
```

Basic Concepts *(Continued)*

- When an object protected by a security policy is accessed, DB2 applies the appropriate access rules, based on the access type and the component type



Basic Concepts *(Continued)*

- Read access rules
  1. DB2LBACREADARRAY: The array component of the user's security label must be greater than or equal to the array component of the object's security label
  2. DB2LBACREADSET: The set component of the user's security label must include the set component of the object's security label
  3. DB2LBACREADTREE: The tree component of the user's security label must include at least one of the elements in the tree component of the object's security label (or the ancestor of one such element)
- Write access rules
  4. DB2LBACWRITEARRAY: The array component of the user's security label must be equal to the array component of the object's security label
  5. DB2LBACWRITESET: The set component of the user's security label must include the set component of the object's security label
  6. DB2LBACWRITETREE: The tree component of the user's security label must include at least one of the elements in the tree component of the object's security label (or the ancestor of one such element)

Basic Concepts *(Continued)*

- Security Label
  - DB2 distinguishes between three types of security labels
    - User security label: A security label that is granted to a database user
    - Row security label: A security label associated with a data row of a database table
    - Column security label: A security label associated with a column of a database table
  - A user can be granted a security label from a given security policy for
    - Read access only
    - Write access only
    - Both read and write access
  - If the Security Administrator (SECADM) would like user Bob to read data with level 'S' and compartments {'A', 'B', 'C'}, but only write data with compartments {'C'}, then
    - Grant Bob a security label L1, where level = 'S' and compartments = {'A', 'B', 'C'} for read access
    - Grant Bob a security label L2, where level = 'S' and compartments = 'C' for write access

Basic Concepts *(Continued)*

- Exemption
  - A special privilege that allows a user to bypass a particular rule in a security policy
  - The SECADM can grant a user an exemption on any combination of the following rules:
    - DB2LBACREADARRAY
    - DB2LBACREADSET
    - DB2LBACREADTREE
    - DB2LBACWRITEARRAY WRITEDOWN
    - DB2LBACWRITEARRAY WRITEUP
    - DB2LBACWRITESET
    - DB2LBACWRITETREE
  - If the SECADM would like to allow user Bob to write data at lower security levels than his level, then
    - Grant Bob an exemption on DB2LBACWRITEARRAY WRITEDOWN
  - WARNING
    - Exemptions allow users to bypass the LBAC rules!

Protected Tables

- A protected table is a table on which LBAC is enforced
- DB2 offers two flavours of protected tables:
  - Protected table with row level granularity
  - Protected table with column level granularity
- Unlike with non protected tables, a Database Administrator (DBADM) has no inherent ability to access data within a protected table!
  - No security label/exemptions, no rows!
- The content of a protected table with row level granularity appears different depending on the identity of the user accessing such table
  - Think of the Truman Show movie with Jim Carrey :-)
- Example

The following creates a protected table with row level granularity:

```
CREATE TABLE T1 (A DB2SECURITYLABEL,
                 B INTEGER,
                 C CHAR(5))
SECURITY POLICY mySecPolicy
```

Protected Tables *(Continued)*

- Protected Column
  - A column secured with a security label
  - Access to a protected column will be denied if the user's security label and exemption credentials do not allow such access
  - A protected table with column level granularity is a table with one or more protected columns
  - Example

The following creates a protected table with column level granularity:

```
CREATE TABLE T1 (A CHAR(8) SECURED WITH mySecLabel,
                 B INTEGER,
                 C CHAR(5))
SECURITY POLICY mySecPolicy
```

Protected Tables *(Continued)*

- Providing the Row Security Label Value on INSERT and UPDATE
  - The row security label column is implicitly a NOT NULL WITH DEFAULT column
  - The default value is the session authorization ID's security label for write access
  - Three (3) different ways of providing the row security label value:
    1. DEFAULT keyword option
    2. SECLABEL_BY_NAME built-in function
    3. SECLABEL built-in function
  - Example

Suppose we have the following protected table with row level granularity:

```
CREATE TABLE T1 (A DB2SECURITYLABEL,B INTEGER,C CHAR(5))
SECURITY POLICY mySecPolicy
```

The row security label can be provided as:

```
INSERT INTO T1 VALUES (DEFAULT, 1, 'Test')
```

Protected Tables *(Continued)*

OR

```
INSERT INTO T1
VALUES
(SECLABEL_BY_NAME('mySecPolicy','mySecLabel'),
 1,
'Test')
```

OR

```
INSERT INTO T1
VALUES
(SECLABEL('mySecPolicy','S:A'),
1,
'Test')
```

Protected Tables *(Continued)*

- Retrieving the Row Security Label Value on SELECT
  - The row security label column is a "regular" column with respect to accessing it using SELECT statements
  - For faster query processing, DB2 stores the row security label value in an encoded format that makes comparing two security labels very efficient
    - A plain SELECT of the row security label column will return this encoded format
  - For an external (string) representation of the row security label value, the SECLABEL_TO_CHAR built-in function could be used
  - Example

```
SELECT SECLABEL_TO_CHAR('mySecPolicy', A), B, C
FROM T1
```

Protected Tables *(Continued)*

- What is table level labeling?
  - Table level labeling associates a security label with the table as whole
  - When all the data in a table is known to always have the same security label, table level labeling avoids incurring:
    - The row level security performance overhead
    - The row security label storage overhead (e.g., millions of rows * N bytes!)
  - The DB2 LBAC solution is the only database MAC implementation that allows an organization to take advantage of table level labeling
    - You can use column level labeling to *simulate* table level labeling
  - Example

The following simulates table level labeling:

```
CREATE TABLE T1 (A CHAR(8) SECURED WITH mySecLabel,
                 B INTEGER SECURED WITH mySecLabel,
                 C CHAR(5) SECURED WITH mySecLabel)
SECURITY POLICY mySecPolicy
```

Protected Tables *(Continued)*

- What else can you do with table level labeling?
  - Table level labeling can also be useful even when your application does not manage classified information
  - You can leverage table level labeling to ensure that your Database Administrator (DBADM) cannot gain access to your favourite tables
  - Currently, this is the only way you can do so in DB2 UDB for LUW

Row Level Labeling Example

```
CREATE SECURITY LABEL COMPONENT LEVEL
ARRAY ['HIGHLY SENSITIVE', 'SENSITIVE', 'CONFIDENTIAL', 'PUBLIC']

CREATE SECURITY POLICY DATAACCESS
COMPONENTS LEVEL WITH DB2LBACRULES
RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL

CREATE SECURITY LABEL DATAACCESS.MANAGERSECLABEL
COMPONENT LEVEL 'SENSITIVE'

GRANT SECURITY LABEL DATAACCESS.MANAGERSECLABEL
TO USER JOE

CREATE TABLE EMPLOYEE (ROWLABEL DB2SECURITYLABEL,
                       EMPNO INTEGER,
                       LASTNAME VARCHAR(20),
                       DEPTNO INTEGER)
SECURITY POLICY DATAACCESS
```

Row Level Labeling Example *(Continued)*

User Joe issues some **INSERT** statements:

INSERT INTO EMPLOYEE **VALUES** (**DEFAULT**, 1, 'SMITH', 11)

ROWLABEL	EMPNO	LASTNAME	DEPTNO
SENSITIVE	1	SMITH	11

INSERT INTO EMPLOYEE **VALUES** (**SECLABEL**('DATAACCESS','CONFIDENTIAL'),
2, 'MILLER', 11)

The statement fails because user Joe is not allowed to write down.

INSERT INTO EMPLOYEE **VALUES** (**SECLABEL**('DATAACCESS','HIGHLY SENSITIVE'),
3, 'WILLIAMS', 11)

The statement fails because user Joe is not allowed to write up.

Row Level Labeling Example *(Continued)*

The SECADM grants user Joe an exemption to write down

GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN
FOR DATAACCESS **TO USER** JOE

User Joe re-issue the following SQL statement:

INSERT INTO EMPLOYEE **VALUES** (**SECLABEL**('DATAACCESS','CONFIDENTIAL'),
2, 'MILLER', 11)

ROWLABEL	EMPNO	LASTNAME	DEPTNO
SENSITIVE	1	SMITH	11
CONFIDENTIAL	2	MILLER	11

Will this **UPDATE** statement from user Nancy (who holds the same security label as user Joe) succeed or fail?

UPDATE EMPLOYEE **SET** DEPTNO = 66 **WHERE** DEPTNO = 11

Row Level Labeling Example *(Continued)*

UPDATE EMPLOYEE **SET** DEPTNO = 66 **WHERE** DEPTNO = 11 AND EMPNO = 1

ROWLABEL	EMPNO	LASTNAME	DEPTNO
SENSITIVE	1	SMITH	66
CONFIDENTIAL	2	MILLER	11

The SECADM creates and grants a new security label to user Alice

CREATE SECURITY LABEL DATAACCESS.EMPLOYEESECLABEL
COMPONENT LEVEL 'CONFIDENTIAL'

GRANT SECURITY LABEL DATAACCESS.EMPLOYEESECLABEL
TO USER ALICE

Row Level Labeling Example *(Continued)*

User Alice issues a **SELECT** statement

SELECT * **FROM** EMPLOYEE

ROWLABEL	EMPNO	LASTNAME	DEPTNO
CONFIDENTIAL	2	MILLER	11

User Alice issues a **DELETE** statement

DELETE FROM EMPLOYEE

User Alice issues a **SELECT** statement

SELECT * **FROM** EMPLOYEE

zero rows are returned. Is the table really empty?

Row Level Labeling Example *(Continued)*

User Joe issues a **SELECT** statement

SELECT * **FROM** EMPLOYEE

ROWLABEL	EMPNO	LASTNAME	DEPTNO
SENSITIVE	1	SMITH	66

LBAC and Constraints

- LBAC & Referential Integrity (RI) Constraints
  - Parent table DEPARTMENT is protected with row level granularity
  - Child table EMPLOYEE is protected with row level granularity
  - Scenario 1: User Bob attempts to delete a row from parent table DEPARTMENT for which a child row exists but is NOT visible to Bob according to the LBAC rules
    - ☒ If the delete is allowed to go through, an orphan child is created and database integrity compromised!
    - ☒ If the delete is rejected, the LBAC read access rules are not observed!
  - Scenario 2: User Alice attempts to insert a row into child table EMPLOYEE for which a parent row exists but is NOT visible to Alice according to the LBAC rules
    - ☒ If the insert is rejected, the RI constraint is not observed!
    - ☒ If the insert is allowed, the LBAC read access rules are not observed!
  - RI constraints and LBAC can be thought of as two conflicting requirements to satisfy within the database!

LBAC and Constraints *(Continued)*

- The following rules explain how LBAC is enforced in the presence of referential integrity constraints:
  - Rule 1: The LBAC read access rules are NOT applied for internally generated scans of child tables (avoids having orphan children)
  - Rule 2: The LBAC read access rules are NOT applied for internally generated scans of parent tables
  - Rule 3: The LBAC write rules are applied when a CASCADE operation is performed on child tables (for example, If a user deletes a parent, but cannot delete any of the children because of an LBAC write rule violation, then the delete is rolled-back and an error raised)

LBAC and Constraints *(Continued)*

- Check Constraints
  - Suppose that user Bob attempts to define a check constraint on protected table T1 but not all rows in T1 are visible to Bob according to the LBAC read access rules
    - ☒ If the check constraint is allowed, database integrity has been compromised!
  - Rule : The LBAC read access rules are NOT applied when validating a check constraint
- Primary and Unique Key Constraints
  - Rule : The LBAC read access rules are NOT applied when validating a primary or unique key constraint
- In general, the LBAC read access rules are not applied for any internal scans generated by the database for its own housekeeping

LBAC and Constraints *(Continued)*

- Inference
  - Given a unique constraint (primary key or unique index) on a protected table with row level granularity, users may infer the value of primary key columns at higher security levels if an insert is rejected due to a primary key constraint violation
  - Recommendation: If inference is a concern, you may want to include the DB2SECURITYLABEL column in your primary key or unique index definitions

LBAC and Query Optimization

- Index Only Plans
  - Traditionally, the query optimizer can choose between three options about how to access a table: A TSCAN (table scan), an ISCAN-FETCH (table access through an index), or an ISCAN (index only access)
  - If an index only plan is chosen for accessing a protected table with row level granularity, and the row security label column is not part of the index key, the LBAC access rules are compromised!
  - The DB2 LBAC implementation will never choose an index only plan if the LBAC rules can be compromised
- Predicate Evaluation Sequence
  - Suppose that a query has a predicate that involves a UDF that takes the whole data row as input parameter and that the UDF source code makes a copy of the data row outside the database (or send it as an e-mail to some destination)
  - Now, suppose that some data row R cannot be returned to the user because this would violate the LBAC access rules
  - If the predicate involving the UDF is evaluated prior to the LBAC rules, then data row R will be consumed by the UDF and LBAC is compromised!
  - The DB2 LBAC implementation ensures that the LBAC rules are always evaluated before any unsafe predicate (i.e., a UDF)

LBAC and Data Partitioning

- DB2 Viper introduces a new feature called partitioned tables which allows table data to be spread across different partitions according to a user-defined partitioning scheme
  - LBAC and partitioned tables can be combined to, for example, increase security through separation of data from different security levels
    - Highly sensitive data on partition A
    - Sensitive data on partition B
    - Confidential data on partition C
    - Public data on partition D
- LBAC can also be combined with other DB2 data partitioning features such as:
  - Multi-Dimensional Clustering (MDC)
  - Data Partitioning Facility (DPF)
    - With DPF comes both performance (parallel query execution) and increased security through physical separation of data from different security levels (e.g., highly sensitive data goes on the most trusted node in the DPF configuration, etc.)

Security Model Extensions

- Security Administrator (SECADM) Authority
  - The authority required to
    - ☒Create and drop security label components
    - ☒Create and drop security policies
    - ☒Create and drop security labels
    - ☒Grant and revoke security labels
    - ☒Grant and revoke exemptions
    - ☒Grant and revoke the SETSESSIONUSER privilege
  - SYSADM is the only authority that can grant SECADM
  - SECADM has no inherent ability to access data within a protected table!
  - SECADM cannot grant any privileges to him/her self!

Security Model Extensions *(Continued)*

- SETSESSIONUSER privilege
  - The privilege required to change the session authorization ID to a different authorization ID
  - The old model used to allow a user with DBADM authority to change the session authorization ID to any ID of their choice
    - In an LBAC world, this would allow the DBADM to (indirectly) gain access to data within a protected table
    - This ability is now controlled through the SETSESSIONUSER privilege

Security Model Extensions *(Continued)*

- CREATE DATABASE
  - Traditionally, when a database is created, DB2 grants SELECT access to the catalogs to PUBLIC
  - Catalogs contain, among other things, statistics about data distribution in a table, which are needed by the query optimizer to determine the best way to execute a query
    - Users may indirectly gain knowledge about some data values by accessing the catalogs!
  - The new RESTRICT option on the CREATE DATABASE command can be used to create a database where access to the catalogs is not granted to PUBLIC
    - Access to the catalogs will then need to be granted on a "need-to-know" basis

Summary

- What are the advantages of the DB2 for LUW LBAC capability?
  - Column Level Labeling
    - Works independently of row level labeling
    - Allows to control access to a column based on security labels
  - Table Level Labeling
    - Can be easily set up as a special case of column level labeling
    - Provides better performance (and storage savings) than row level labeling in application scenarios where all the data in a table is known to have the same security label
    - Can be leveraged by applications that do not manage classified information to prevent the DBADM from gaining access to the application's tables
  - Flexibility of Security Policy Definition
    - Security Administrators can define the security policy that suits them best
    - Security Administrators can choose from three predefined security label component types (array, set, and tree) to define a security label type
    - A security label can have any number of components of any types as defined by the security policy (i.e., not the hard-wired "level" and "compartments" as with other MAC solutions)

Summary *(Continued)*

- DB2 LBAC selects and applies the relevant LBAC rules based on the security policy (i.e., not the hard-wired read and write access rules as with other MAC solutions)
- DB2 LBAC can be easily used to address the requirements of application scenarios where the security label consists of more or less than the traditional level and compartments components (foreign governments apps.)

► Flexibility of Security Label to User Assignment

- Database users can be granted a security label for both read and write access, for read access only, or for write access only
- Granting a database user a security label for read access only allows them to read data, but not modify such data (attractive for assigning to database users who need access to sensitive data only to generate reports)

► Fine Granularity

- Security Administrators can protect different tables with different security policies within the same database

• Security Label Encoding

- Users provide and retrieve data row labels in the character representation they are familiar with, but DB2 LBAC stores data row labels in an internal binary representation for efficient enforcement of the LBAC access rules

Summary *(Continued)*

- The following are some points to consider when using LBAC
  - Create your database, ideally with the new RESTRICT option
  - Determine who should be your security administrator (SECADM) and assign that authority to them
  - Determine how you would like to label your data, e.g., how many security label components make up your security label, and what are the types of those components (ARRAY, SET, TREE)?
  - Determine who should be granted a security label, and who would need to be granted one or more exemptions
  - Have your SECADM set the security policy, and grant security labels and exemptions to users as required
  - Determine whether you need row level labeling, column level labeling or table level labeling
  - Create your tables with the level of granularity you need
- Where can I learn more about DB2 LBAC?
  - See the LBAC chapter in the DB2 Administration Guide
  - See the DB2 SQL Reference
  - E-mail me at wrjaibi@ca.ibm.com if you cannot find what you are looking for ☺

Thank you!