


CSC12001

Security Issues in Information Systems

C03 – Access Control - VPD (Virtual Private Databases)


Dr. Phạm Thị Bạch Huệ
MSc. Lương Vĩ Minh

Information System Department – Faculty of Information Technology
University of Science, VNU-HCM



@ 2020 – University of Science – Vietnam National University HCM

1



Outline

1. Row – level Security with VPD
2. Principle of VPD
3. Enforcing a security policy using VPD
4. Using application context
5. Discussion

2

2

Goal

- VPD is a tool of Oracle supporting for **content-based access control**.
- VPD's row-level security allows security managers to restrict access to records based on a security policy implemented in PL/SQL.

3

3

Principle of VPD

- A *security policy* describes the rules governing access to the data rows.
- **Step 1:** Creating a PL/SQL function that returns a **string** (called the **predicate**) that representing the security policy.
- **Step 2:** The function is then registered against the **tables, views, or synonyms** you want to protect by using the DBMS_RLS PL/SQL package.
- **Step 3:** When a query is issued against the protected object, Oracle automatically and transparently appends the **string** returned from the function to the original SQL statement, thereby filtering the data records.

4

4



- **EMP (USERNAME, ENAME, JOB, SAL, DEPTNO)**
- Security policy: **Exclude department 10 records from the query on Scott.EMP.**
- PL/SQL function:

```
sec_mgr:
CREATE OR REPLACE FUNCTION no_dept10 (
  p_schema IN VARCHAR2,
  p_object IN VARCHAR2)
RETURN VARCHAR2
AS
BEGIN
  RETURN 'deptno != 10';
END;
```

5

5



- To protect the SCOTT.EMP table, simply associate the preceding PL/SQL function to the table using the DBMS_RLS.ADD_POLICY procedure:

```
sec_mgr:
BEGIN
  DBMS_RLS.add_policy
    (object_schema => 'SCOTT',
     object_name => 'EMP', -- the name of the object to which the policy will be applied
     policy_name => 'VPD_testing', -- a name for the policy
     policy_function => 'no_dept10'); -- the name of a PL/SQL function
END;
```

6

6

DBMS_RLS.ADD_POLICY procedure

- The PL/SQL functions are registered to tables, views, or synonyms by invoking the DBMS_RLS.ADD_POLICY procedure.
- The DBMS_RLS package is not granted to everyone; administrators will require direct execute privileges on the package.
- The ADD_POLICY procedure requires, at minimum:
 1. The name of the object to which the policy will be applied,
 2. A name for the policy,
 3. The name of a PL/SQL function that will implement the security policy.

7

7

Example

User submits an original query:

```
SELECT DISTINCT DEPTNO FROM EMP
```

1. DB checks VPD policies for a matching table and statement: 1. Table = Emp, 2. DML=Select
2. Calls function when activate the policy: return 'deptno != 10'
3. Function returns a string, which will be appended to original query and executed:

```
SELECT DISTINCT DEPTNO
FROM EMP
WHERE 'DEPTNO != 10'
```

- Department 10 is no longer seen because the RLS policy transparently filters out those records:

```
DEPTNO
-----
20
30
```

8

8



- To test this policy, log on as a user with access to the SCOTT.EMP table and issue your DML.
- The following shows all the department numbers available in the table.

```
scott> SELECT DISTINCT deptno FROM emp;
```

```

      DEPTNO
-----
          20
          30

```

9

9



- The security policy implemented by the function can change without requiring any **re-registration** with the DBMS_RLS package.

- Security policy: **No records should be returned for the user SYSTEM:**

```

sec_mgr> CREATE OR REPLACE FUNCTION no_dept10 (
  p_schema IN VARCHAR2,
  p_object IN VARCHAR2)
  RETURN VARCHAR2
AS
BEGIN
  RETURN 'USERNAME != 'SYSTEM'';
END;
```

10

10

- Test by counting records as scott

```
scott> SELECT COUNT(*) Total_Records FROM emp;
TOTAL_RECORDS
-----
14
```

- Test by counting records as SYSTEM

```
system> SELECT COUNT(*) Total_Records FROM scott.emp;
TOTAL_RECORDS
-----
0
```

11

11

Example

Policy: A user can select/ insert/ update only his/her record, DBA can see all the rows of EMP table.

1. Create the policy function:

```
Create function sec_function(p_schema varchar2, p_obj
    varchar2)
Return varchar2
As
    user VARCHAR2(100);
Begin
    if ( SYS_CONTEXT('userenv', 'ISDBA') ) then
        return ' ';
    else
        user := SYS_CONTEXT('userenv', 'SESSION_USER');
        return 'username = ' || user;
    end if;
End;
```

// userenv = the pre-defined application context

12

12

2. Binding the security function to the EMP table:

```
execute dbms_rls.add_policy (object_schema => 'scott',
                             object_name => 'EMP',
                             policy_name => 'my_policy',
                             function_schema => 'sec_mgr',
                             policy_function => 'sec_function',
                             statement_types => 'select, update, insert',
                             update_check => TRUE );
```

13

13

3. Testing:

```
Scott:  select * from emp;
=> select * from emp where username = 'scott';
: result contains only the row of scott
```

```
DBA: select * from emp;
=> select * from emp;
: result contains all the rows
```

```
Blake: insert into emp values('Blake', , , 2000, 10); OK!
```

```
Blake: insert into emp values('Peter', , , 2000, 20); ERROR!
```

14

14

EMP1 (E_ID, ENAME, SALARY)

E_ID	ENAME	SALARY
1	A	80
2	B	60
3	C	99

- Policy: Users can see only their own salaries.

15

15

1. Creating the policy function:

```

Create function sec_function(p_schema varchar2,
p_obj varchar2)
Return varchar2
As
    user VARCHAR2(100);
Begin
    user := SYS_CONTEXT('userenv', 'SESSION_USER');
    return 'ename = ' || user;
end if;
End;

```

16

16

Example

2. Binding the policy function to Emp1

```
execute dbms_ols.add_policy (object_schema => 'scott',
    object_name => 'emp1',
    policy_name => 'my_policy',
    function_schema => 'sec_mgr',
    policy_function => 'sec_function',
    sec_relevant_cols => 'salary');
```

17

17

Example

3. B retrieves the data:

```
select e_id, ename from Emp1;
```

e_id	ename
1	A
2	B
3	C

```
select e_id, ename, salary from Emp1;
```

e_id	ename	Salary
2	B	60

18

18

2'. Another way of binding the security function to EMP1:

```
execute dbms_ols.add_policy (object_schema => 'SCOTT',
object_name => 'emp1',
policy_name => 'my_policy',
function_schema => 'SEC_MGR',
policy_function => 'sec_function',
sec_relevant_cols => 'salary',
sec_relevant_cols_opt => dbms_ols.ALL_ROWS);
```

19

19

3'. B accesses Emp1 again:

```
select e_id, ename from Emp1;
```

e_id	ename
1	A
2	B
3	C

```
select e_id, ename, salary from Emp1;
```

e_id	ename	Salary
1	A	
2	B	60
3	C	

20

20

Key Points

- VPD allows the database to apply separate policies based on the DML type. For example, the database can easily support a policy to allow all records for SELECT statements; an INSERT, UPDATE policy to restrict records to a user's department on insert and update operations; and a DELETE policy that restricts DELETE operations to only the user's record.
- Whenever a user directly or indirectly accesses a protected table, view, or synonym, the RLS engine is transparently invoked, the PL/SQL function registered will execute, and the SQL statement will be modified and executed.
- Multiple policies also can be applied to the same object: the database logically ANDs the policies together.
That is, if there is one policy that returns 'ename = USER' and another policy (on the same object for the same DML) that returns 'sal > 2000', the database will automatically add both policies, effectively generating where ename = USER *and* sal > 2000.

21

21

Key points

- The PL/SQL functions are registered to tables, views, or synonyms by invoking the DBMS_RLS.ADD_POLICY procedure.
- The DBMS_RLS package is not granted to everyone; administrators will require direct execute privileges on the package.
- The ADD_POLICY procedure requires, at minimum, the name of the object to which the policy will be applied, a name for the policy, and the name of a PL/SQL function that will implement the security policy.
- The policies can be applied to all DML statements such as: SELECT, INSERT, UPDATE, DELETE, and INDEX statements. The index affects CREATE INDEX and ALTER INDEX DDL commands.
- The ADD_POLICY procedure accepts a STATEMENT_TYPES parameter that allows the administrator to specify which DML operations the policy is to apply.
- https://docs.oracle.com/cd/B28359_01/appdev.111/b28419/d_rls.htm#i998159

22

22

Application Context

- **PEOPLE (USERNAME, JOB, SALARY, DEPTNO)**

```
sec_mgr: CREATE TABLE lookup_dept
```

```
AS SELECT username, deptno FROM scott.people;
```

- **A user is allowed to see all records; to insert and update records only within their department; and to delete only their individual record.**

23

23

- sec_mgr creates namespace for application context

```
sec_mgr: CREATE CONTEXT people_ctx USING sec_mgr.people_ctx_mgr;
```

- The namespace manager program will set the context based on the user's department number as stored in the LOOKUP_DEPT table:

sec_mgr creates namespace manager program for modifying context.

```
sec_mgr:
```

```
CREATE OR REPLACE PACKAGE people_ctx_mgr
```

```
AS
```

```
PROCEDURE set_deptno;
```

```
PROCEDURE clear_deptno;
```

```
END;
```

24

24

- sec_mgr:

```
CREATE OR REPLACE PACKAGE BODY people_ctx_mgr
AS
  PROCEDURE set_deptno
  AS
    l_deptno NUMBER;
  BEGIN
    SELECT deptno INTO l_deptno FROM lookup_dept
    WHERE username = SYS_CONTEXT ('userenv', 'session_user');
    DBMS_SESSION.set_context (namespace => 'people_ctx',
                              ATTRIBUTE => 'deptno',
                              VALUE => l_deptno);
  END set_deptno;
```

25

25

```
PROCEDURE clear_deptno
AS
  BEGIN
    DBMS_SESSION.clear_context (namespace => 'people_ctx', ATTRIBUTE =>
    'deptno');
  END clear_deptno;
END people_ctx_mgr;
```

26

26



```

sec_mgr: CREATE OR REPLACE TRIGGER set_user_deptno
AFTER LOGON ON DATABASE
BEGIN
    sec_mgr.people_ctx_mgr.set_deptno;
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        -- If user is not in table,
        -- a no_data_found is raised
        -- If exception is not handled, then users not in table
        -- will be unable to log on
        NULL;
END;

```

27

27



Test the context by logging in as the SCOTT user:

```

sec_mgr:CONN scott/tiger
Connected.
Scott: COL deptno format a8
Scott: SELECT SYS_CONTEXT ('people_ctx',
                        'deptno') deptno
                        FROM DUAL;

DEPTNO
-----
20

```

28

28

Creating the policy function

```
sec_mgr: CREATE OR REPLACE FUNCTION dept_only (
  p_schema IN VARCHAR2 DEFAULT NULL,
  p_object IN VARCHAR2 DEFAULT NULL)
  RETURN VARCHAR2
AS
BEGIN
  RETURN 'deptno = sys_context(''people_ctx'', 'deptno')';
END;
```

29

29

• Applying the Insert/Update Policy

```
sec_mgr: BEGIN
  DBMS_RLS.add_policy
    (object_schema      => 'SCOTT',
     object_name        => 'PEOPLE',
     policy_name        => 'PEOPLE_IU',
     function_schema    => 'SEC_MGR',
     policy_function     => 'Dept_Only',
     statement_types     => 'INSERT,UPDATE',
     update_check       => TRUE);
END;
```

30

30



Scott: SELECT username, deptno FROM people WHERE username < 'C';

USERNAME	DEPTNO
-----	-----
ALLEN	30
BLAKE	30
ADAMS	20

Scott: UPDATE people SET username = 'GRIZZLY' WHERE username = 'ADAMS';

1 row updated.

Scott: UPDATE people SET username = 'BOZO' WHERE username = 'BLAKE';

no rows updated.

31

31



Scott: INSERT INTO people (username, job, salary, deptno)
VALUES ('KNOX', 'Clerk', '3000', 20);

1 row created.

Scott: INSERT INTO people username, job, salary, deptno
VALUES ('ELLISON', 'CEO', '90000', 30);

INSERT INTO people
*

ERROR at line 1:

ORA-28115: policy with check option violation

32

32


```
sec_mgr@KNOX10g> CREATE OR REPLACE FUNCTION user_only (
    p_schema IN VARCHAR2 DEFAULT NULL,
    p_object IN VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2
AS
BEGIN
    RETURN 'username = sys_context(''userenv'', 'session_user')';
END;
```

Function created.

33

33

- To apply the delete policy, specify DELETE statements in the ADD_POLICY procedure and provide the USER_ONLY function for the POLICY_FUNCTION:

```
sec_mgr: BEGIN
    DBMS_RLS.add_policy
        (object_schema    => 'SCOTT',
         object_name       => 'PEOPLE',
         policy_name       => 'People_Del',
         function_schema   => 'SEC_MGR',
         policy_function    => 'user_only',
         statement_types    => 'DELETE');
END;
```

34

34

```
Scott: DELETE FROM people;
```

```
1 row deleted.
```

```
Scott: SELECT * FROM people
       WHERE username = 'SCOTT';
```

```
no rows selected
```

35

35

Application context

- An application context is a set of name-value pairs, held in memory, which can be defined, set, and retrieved by users and applications.
- Related values can be grouped together. The group is collectively defined and accessed by its name or namespace.
- Within the namespace, the individual attributes and their associated values are stored in memory and retrieved by calling a PL/SQL function call.
- By storing the values in either shared or private memory, depending on the context, the access to the values will be very fast.
- Typically, application contexts hold several attributes, such as an application or **user name, organization, role, and title**. Your security policies may reference these attributes in controlling user access.
- Storing the values in memory saves the time and resources that would be required to repetitively query data tables to retrieve this information.
- There is no requirement to use security with an application context or to use an application context with a security implementation.

36

36

Privileges related to VPD

- Privileges needed for creating a policy function:
 - EXECUTE on DBMS_RLS package for adding the policy function to an object.
 - add_policy, drop_policy, enable_policy, ...
 - CREATE PROCEDURE for creating the policy.
- Privileges needed for creating the application context:
 - CREATE ANY CONTEXT
 - CREATE PROCEDURE
 - EXECUTE on DBMS_SESSION package (DBMS_SESSION.Set_context())
 - Access to the object.
- EXEMPT ACCESS POLICY privilege allows the grantee to be exempted from *all* RLS functions
- DBAs and the data owner cannot bypass the RLS policy.
- Users can bypass VPD
 - SYS
 - User with EXEMPT ACCESS POLICY privilege.
- VPD can be used for controlling the following privileges SELECT, INSERT, UPDATE, INDEX, DELETE.

37

37

Q&A

Dr. Phạm Thị Bạch Huệ - ptbhue@fit.hcmus.edu.vn

MSc. Lương Vĩ Minh – lvminh@fit.hcmus.edu.vn

38

38