


PURDUE UNIVERSITY

Computer Security CS 426 Lecture 19

Database Security VPD



Elisa Bertino
Purdue University
IN, USA
bertino@cs.purdue.edu

Center for Education and Research
in Information Systems and Security

1

PURDUE UNIVERSITY

Oracle VPD

- Virtual Private Database (VPD)
 - Fine-grained access control: associate security policies with database objects
 - Application Context: define and access application or session attributes and use them in access control, for example for implementing temporal access control
- By combining these two features, VPD enables administrators to define and enforce row-level access control policies based on session attributes

PURDUE UNIVERSITY

Why VPD

- Scalability
 - Table Customers contains 1,000 customer records. Suppose we want customers to access their own records only. Using views, we need to create 1,000 views. Using VPD, it can be done with a single policy function.
- Simplicity
 - Say, we have a table T and many views are based on T. Suppose we want to restrict access to some information in T. Without VPD, all view definitions have to be changed. Using VPD, it can be done by attaching a policy function to T; as the policy is enforced in T, the policy is also enforced for all the views that are based on T.
- Security
 - Server-enforced security (as opposed to application-enforced).

PURDUE UNIVERSITY

Oracle VPD

- How does it work?

When a user accesses a table (or view or synonym) which is protected by a VPD policy (function):

 1. The Oracle server invokes the policy function.
 2. The policy function returns a predicate, based on session attributes or database contents.
 3. The server dynamically rewrites the submitted query by appending the returned predicate to the WHERE clause.
 4. The modified SQL query is executed.

Oracle VPD - Example

- Suppose Alice has (is the owner of) the following table.
`my_table(owner varchar2(30), data varchar2(30));`
- Suppose that we want to implement the following policy:
 - Users can access only data that refer to themselves. However Admins should be able to access any data without restrictions.

Oracle VPD - Example

1. Create a policy function

```

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

userenv is the pre-defined application context
 object_name is the name of table or view to which the policy will apply
 object_schema is the schema owning the table or view

SYS_CONTEXT

- In Oracle/PLSQL, the `sys_context` function is used to retrieve information about the Oracle environment.
- The syntax for the `sys_context` function is:
`sys_context(namespace, parameter, [length])`
- `namespace` is an Oracle namespace that has already been created. If the namespace is 'USERENV', attributes describing the current Oracle session can be returned.
- `parameter` is a valid attribute that has been set using the `DBMS_SESSION.set_context` procedure.
- `length` is optional. It is the length of the return value in bytes. If this parameter is omitted or if an invalid entry is provided, the `sys_context` function will default to 256 bytes

USERENV namespace valid parameters

Parameter	Explanation	Return Length
AUDITED_CURSORID	Returns the cursor ID of the SQL that triggered the audit	NA
AUTHENTICATOR_DATA	Authentication data	256
AUTHENTICATOR_TYPE	Describes how the user was authenticated. Can be one of the following values: Database, OS, Network, or Proxy	30
BG_JOB_ID	If the session was established by an Oracle background process, this parameter will return the Job ID. Otherwise, it will return NULL.	30
CLIENT_IDENTIFIER	Returns the client identifier (global context)	64
CLIENT_INFO	User session information	64
CURRENT_SCHEMA	Returns the default schema used in the current schema	30
CURRENT_SCHEMAID	Returns the identifier of the default schema used in the current schema	30
CURRENT_SQL	Returns the SQL that triggered the audit event	64
CURRENT_USER	Name of the current user	30
CURRENT_USERID	Usuid of the current user	30
DB_DOMAIN	Domain of the database from the DB_DOMAINS initialization parameter	256
DB_NAME	Name of the database from the DB_NAME initialization parameter	30
ENTRYID	Available auditing entry identifier	30
EXTERNAL_NAME	External of the database user	256
GLOBAL_CONTEXT_MEMORY	The number used in the System Global Area by the globally accessed context	NA
HOST	Name of the host machine from which the client has connected	64
INSTANCE	The identifier number of the current instance	30

USERENV namespace valid parameters



OSDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.	30
LANG	The ISO abbreviation for the language.	62
LANGUAGE	The language, territory, and character of the session. In the following format: language_territory_character set	52
NETWORK_PROTOCOL	Network protocol used.	256
NLS_CALENDAR	The calendar of the current session.	62
NLS_CURRENCY	The currency of the current session.	62
NLS_DATE_FORMAT	The date format for the current session.	62
NLS_DATE_LANGUAGE	The language used for dates.	62
NLS_SORT	BINARY or the linguistic sort basis.	62
NLS_TERRITORY	The territory of the current session.	62
OS_USER	The OS username for the user logged in.	30
PROXY_USER	The name of the user who opened the current session on behalf of SESSION_USER.	30
PROXY_USERID	The identifier of the user who opened the current session on behalf of SESSION_USER.	30
SESSION_USER	The database user name of the user logged in.	30
SESSION_USERID	The database identifier of the user logged in.	30
SESSIONID	The identifier of the auditing session.	30
TERMINAL	The OS identifier of the current session.	10



Oracle VPD - Example

2. Attach the policy function to my_table

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

- The VPD security model uses the Oracle *dbms_rls* package (RLS stands for row-level security)
- `update_check`: Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting `update_check` to TRUE causes the server to also check the policy against the value after insert or update.

DBMS_RLS.ADD_POLICY syntax

```
DBMS_RLS.ADD_POLICY (
    object_schema IN VARCHAR2 NULL,
    object_name IN VARCHAR2,
    policy_name IN VARCHAR2,
    function_schema IN VARCHAR2 NULL,
    policy_function IN VARCHAR2,
    statement_types IN VARCHAR2 NULL,
    update_check IN BOOLEAN FALSE,
    enable IN BOOLEAN TRUE,
    static_policy IN BOOLEAN FALSE,
    policy_type IN BINARY_INTEGER NULL,
    long_predicate IN BOOLEAN FALSE,
    sec_relevant_cols IN VARCHAR2,
    sec_relevant_cols_opt IN BINARY_INTEGER NULL);
```

Oracle VPD - Example

3. Bob accesses my_table

```
select * from my_table;
```

```
=> select * from my_table where owner = 'bob';
: only shows the rows such that owner is 'bob'
```

```
insert into my_table values('Some data', 'bob'); OK!
```

```
insert into my_table values('Other data', 'alice'); NOT OK!
= because of the check option.
```

Policy Commands

- ADD_POLICY – creates a new policy
- DROP_POLICY – drops a policy


```
DBMS_RLS.DROP_POLICY (
  object schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2,
  policy_name    IN VARCHAR2);
```
- ENABLE_POLICY – enables or disables a fine-grained access control policy


```
DBMS_RLS.ENABLE_POLICY (
  object schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2,
  policy_name    IN VARCHAR2,
  enable         IN BOOLEAN   );
```

Enable - TRUE to enable the policy, FALSE to disable the policy

Column-level VPD

- Instead of attaching a policy to a whole table or a view, attach a policy only to security-relevant columns
 - Default behavior: restricts the number of rows returned by a query.
 - Masking behavior: returns all rows, but returns NULL values for the columns that contain sensitive information.
- Restrictions
 - Applies only to 'select' statements
 - The predicate must be a simple Boolean expression.

Column-level VPD: Example

- Suppose Alice has (is the owner of) the following table.
Employees (e_id number(2), name varchar2(10), salary number(3));

e_id	Name	Salary
1	Alice	80
2	Bob	60
3	Carl	99

- Policy: Users can access e_id's and names without any restriction. But users can access only their own salary information.

Column-level VPD: Example

1. Create a policy function

```
Create function sec_function (object_schema varchar2,
object_name varchar2)
Return varchar2
As
  user VARCHAR2(100);
Begin
  user := SYS_CONTEXT('userenv', 'SESSION_USER');
  return 'name = ' || user;
End;
```

Column-level VPD: Example

2. Attach the policy function to Employees (default behavior)

```
execute dbms_rls.add_policy (object_schema => 'Alice',
                             object_name => 'employees',
                             policy_name => 'my_policy',
                             function_schema => 'Alice',
                             policy_function => 'sec_function',
                             sec_relevant_cols=>'salary');
```

Column-level VPD: Example

3. Bob accesses table Employees (with the default behavior). REMEMBER: default behavior restricts the number of rows returned by a query

a) select e_id, name from Employee;

e_id	Name
1	Alice
2	Bob
3	Carl

b) select e_id, name, salary from Employee;

e_id	Name	Salary
2	Bob	60

Column-level VPD: Example

- 2'. Attach the policy function to Employees (masking behavior)

```
execute dbms_rls.add_policy (object_schema => 'Alice',
                             object_name => 'employees',
                             policy_name => 'my_policy',
                             function_schema => 'Alice',
                             policy_function => 'sec_function',
                             sec_relevant_cols=>'salary',
                             sec_relevant_cols_opt=>dbms_rls.ALL_ROWS);
```

Column-level VPD: Example

3. Bob accesses table Employees (with masking behavior). REMEMBER: Masking behavior returns all rows, but returns NULL values for the columns that contain sensitive information.

select e_id, name from Employee;

e_id	Name
1	Alice
2	Bob
3	Carl

Select e_id, name, salary from Employee;

e_id	Name	Salary
1	Alice	
2	Bob	60
3	Carl	

Application Context

- Application contexts act as secure caches of data that may be used by a fine-grained access control policy.
 - Upon logging into the database, Oracle sets up an application context in the user's session.
 - You can define, set and access application attributes that you can use as a secure data cache.
- There is a pre-defined application context, "userenv".

Application Context

- One can create a customized application context and attributes.
 - Say, each employee can access a portion of the Customers table, based on the job-position.
 - For example, a clerk can access only the records of the customers who lives in a region assigned to him. But a manager can access any record.
 - Suppose that the job-positions of employees are stored in a LDAP server (or in the Employee table).
 - Such information can be accessed and cached in an application context when an employee logs in.
- To set an attribute value in an application context,
 - DBMS_SESSION.SET_CONTEXT('namespace', 'attributename', value);
- To get an attribute value from an application context,
 - SYS_CONTEXT('namespace', 'attributename');

Create Application Context

1. Create a PL/SQL package that sets the context

```

Create package Set_emp_env IS
  procedure Set_job_position IS
    jp varchar(100);
  begin
    select job_pos into jp from Employee
    where name = SYS_CONTEXT('USERENV', 'SESSION_USER');
    DBMS_SESSION.SET_CONTEXT('emp_env', 'job', jp);
  end;
End;
```

2. Create a context and associate it with the package

```
Create Context emp_env Using Emp_env_context;
```

- Any attribute in the "emp_env" context can only be set by procedures in the "Emp_env_context" package.

Using Application Context

3. Set the context before users retrieve data (at the login)

```

Create or Replace Trigger Emp_trig
  After Logon On Database
  Begin
    Emp_env_context.Set_job_position
  End
```

- Use an event trigger on login to pull session information into the context.

4. Use the context in a VPD function

```

if (SYS_CONTEXT('emp_env', 'job') = 'manager')
  return '';
else ...
```

Multiple Policies

- It is possible to associate multiple policies with a database object.
 - The policies are enforced with AND syntax.
 - For example, suppose table T is associated with {P1, P2, P3}.
 - When T is accessed by query Q = select A from T where C.
 - Q' = select A from T where $C \wedge (c1 \wedge c2 \wedge c3)$.
- Different from Stonebraker's approach
 - The policies are enforced with OR syntax.
 - Q' = select A from T where $C \wedge (c1 \vee c2 \vee c3)$.

VPD Related Privileges

- Who can create VPD policies? That is, what privileges are needed to create a VPD policy on a database object?
 - EXECUTE on the DBMS_RLS package to attach a policy to an object
 - the package includes add_policy, drop_policy, enable_policy, and so on.
 - CREATE PROCEDURE to create a policy function
 - Not absolutely necessary as you can use somebody else's policy functions.
 - Does not need to have any privilege on the policy functions.
 - Does not require any object privilege on the target objects unless you are defining the policy function (explained later).
- Who can create application contexts?
 - CREATE ANY CONTEXT (there is no CREATE CONTEXT)
 - CREATE PROCEDURE
 - EXECUTE on the DBMS_SESSION package
 - Privileges on the objects that the setup functions access.
- Two classes of users are exempt from VPD policies.
 - SYS user is exempt by default.
 - Users with the EXEMPT ACCESS POLICY system privilege.

AC Based-on DB Content

- It is possible to define VPD policy functions without using the application context. Instead, we can directly query the database content from the policy functions.
- Alice: Employees(e_id number(2), name varchar2(10), salary number(2));
- Bob: Values(p_id number(2), value number(2));
- Users can access the record of any employee whose salary is less than the maximum value in Values.

AC Based-on DB Content

1. Create a policy function


```
create or replace function Policy_func (object_schema varchar2, object_name
varchar2)
return varchar2
as
cond varchar2(100);
mxv number;
begin
select max(value) into mxv from Bob.Values;
cond := 'salary < ' || mxv;
return (cond);
end Policy_func ;
```
2. Attach the function to Employee


```
execute dbms_rls.add_policy('alice', 'employees', 'policy', 'alice', 'Policy_func',
'select');
```

Issue (1): Invoker's right?

- As previously mentioned, one can attach a policy function to a database object whether or not (s)he has any privilege on the function or the objects the function refers to.
- Then when the policy function is invoked, the function runs on behalf of whom? The definer? The invoker?
 - If the function is defined as definer's right, the answer is straightforward. The definer.
 - What if the function is defined as invoker's right? Surprisingly, the invoker's right function still runs on behalf of the definer.
 - However, if the policy function invokes an invoker's right procedure, the procedure runs on behalf of the invoker.
 - Not consistent at all.

Issue (2): Recursion

- "Note: Although you can define a policy against a table, you *cannot* select that table from within the policy that was defined against the table." (from Ch. 13 of Oracle Security Guide)
 - That is, a policy function of an object should not access the object.
 - Suppose that a policy function PF that protects a table T accesses T.
 - When T is accessed, PF is invoked. PF tries to access T, and another PF is invoked. This results in endless function invocations.
- This cyclic invocation can occur in a longer chain.
 - For example, define a policy function for T, that accesses another table T₁. If T₁ is protected by another policy function that refers to T, then we have a cycle.
 - It is hard to check. (A policy function can even invoke a C program.)
 - Note that this problem can be avoided by using application context.

Discussion

- VPD provides a very powerful access control.
- It is difficult, if not impossible, to verify whether or not a particular user has access to a particular data item in a particular table in a particular state.
 - Such verification requires checking all policy functions.
 - As policy functions are too "flexible", it is computationally impossible to analyze them.