Lock it Down:
PL/SQL Security Features for Code Assertion, VPD, and Encryption

Peter Koletzke
Technical Director &
Principal Instructor
ORACLE ACE Director

quovera

utoug utah oracle users group

Good to Remember

We will bankrupt ourselves
in the vain search for
absolute security.

—Dwight David Eisenhower,
(1890-1969)

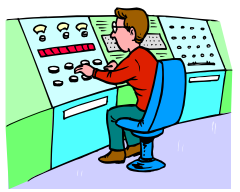
Scope

- In scope – 10g &11g PL/SQL techniques:
  - Data access
  - Functional access
  - Encryption
- Not in scope - security techniques:
  - OS, web or application server, file systems
  - Denial of service (DOS)
  - Cross-site Scripting (XSS)
  - Parameter manipulation, session hijacking, buffer overflow

A bit about 12c later, if time

Agenda

- Introduction
- Locking down application functions
- Locking down data
- New 12c features
- Further study



Security Planning

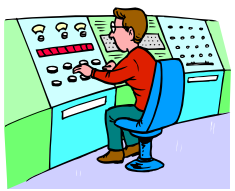
- Need to develop test plans early on, design into system
  - Who can do what, who can see what
  - Application security fits into an overall plan
    - Risk/exposure/safeguards
    - Test roles, functions, data
    - Coding standards
- How not to handle security: roll your own
  - Better: use prebuilt techniques
    - Vendors have certified and tested them

General Safeguards

- Application roles and privileges
  - Users in roles
  - Roles granted to application functions
- Application uses a non-privileged “user” account
  - Not application table owner account
  - User account cannot do DDL
- Restrict access to DML (including SELECT) to application tables
- Think about how users could foil safeguards
  - For example, SQL injection

Agenda

- Introduction
- Locking down application functions
- Locking down data
- New 12c features
- Further study



What is SQL Injection?

- An attempt to add unintended clauses to a SQL statement
  - For example, a search field on a web form
  - The value entered is used for the query:
    - Adds search criteria to a WHERE clause OR
    - Passed as a parameter to a PL/SQL function

Simple Example

- Search form field for “Last Name”
  - Example query to process the field

```
SELECT email, first_name
FROM employees
WHERE last_name ='||F_LastName||';
```

- User is supposed to enter a name like “Hunold”

```
SELECT email, first_name
FROM employees
WHERE last_name = 'Hunold';
```


“Bad” User Does This

- Fills in Last Name field as:

```
'||last_name; DELETE FROM employees
WHERE 1=1||'
```

- SQL becomes

```
SELECT email, first_name
FROM employees
WHERE last_name = ''|| last_name;
DELETE FROM employees WHERE 1=1||'';
```

PL/SQL Function Also Vulnerable

```
PROCEDURE find_emp(
   p_last_name  IN VARCHAR2,
   p_email      OUT VARCHAR2,
   p_first_name OUT VARCHAR2)
IS
   v_stmt VARCHAR2(2000);
BEGIN
   v_stmt := 'SELECT email, first_name '||
             'FROM   employees '||
             'WHERE  last_name = '''|| p_last_name||''';
   --
   EXECUTE IMMEDIATE v_stmt INTO p_email, p_first_name;
END find_emp;
```

- Same value passed into p_last_name will have the same results (Ouch!)

SQL Injection Cure 1: Bind Variables

8i+

- **The** most important technique

```
SELECT email, first_name
FROM employees
WHERE last_name = :p;
```

- Reason: statement is parsed before the bind variable value is substituted
  - The value is always treated as a value, not part of the statement
- Same in PL/SQL

PL/SQL Bind Variable

```
PROCEDURE find_emp(
   p_last_name  IN VARCHAR2,
   p_email      OUT VARCHAR2,
   p_first_name OUT VARCHAR2)
IS
   v_stmt VARCHAR2(2000);
BEGIN
   v_stmt := 'SELECT email, first_name '||
             'FROM   employees '||
             'WHERE  last_name = :p';
   --
   EXECUTE IMMEDIATE v_stmt INTO p_email, p_last_name
   USING p_last_name;
END find_emp;
```

- Value passed in as a value, not as a SQL clause

Building SQL Dynamically

```
PROCEDURE find_emp(
   p_col_1       IN VARCHAR2,
   p_col_1_value IN VARCHAR2,
   p_email       OUT VARCHAR2)
IS
   v_stmt VARCHAR2(2000);
BEGIN
   v_stmt := 'SELECT email '||
             'FROM   employees '||
             'WHERE  '||p_col_1||' = :p';
   --
   EXECUTE IMMEDIATE v_stmt INTO p_email
   USING p_col_1_value;
END find_emp;
```

- The column name needs to be flexible
- SQL injection danger if you concatenate SQL

SQL Injection Cure 2: Code Assertion

10g+

- Confirm that the component you concatenate is a SQL object
- Use the **DBMS_ASSERT** package
  - **ENQUOTE_LITERAL()**
    - To safely add quotes around a value
  - **SIMPLE_SQL_NAME()**
    - To ensure the name is a real database object
  - **E_ENCODE_ERROR**
    - Exception raised when there is a violation of the assertion

DBMS_ASSERT Examples

- **ENQUOTE_LITERAL()**
  - Adds quotes and checks for spurious quotes in the value
  - **VALUE_ERROR exception**
- **SIMPLE_SQL_NAME()**
  - Checks that the name could be used for a SQL object
  - **SYS.DBMS_ASSERT.INVALID_SQL_NAME** exception
- Always lead with SYS schema prefix
  - Ensures the proper user's package is run

Adding Quotes

```
DECLARE
   v_value   VARCHAR2(200);
BEGIN
   v_value := 'HARRY';
   DBMS_OUTPUT.PUT_LINE(v_value||
     ' = '||SYS.DBMS_ASSERT.ENQUOTE_LITERAL(v_value));
   v_value := 'HARRY''';
   DBMS_OUTPUT.PUT_LINE(v_value||
     ' = '||SYS.DBMS_ASSERT.ENQUOTE_LITERAL(v_value));
EXCEPTION
   WHEN VALUE_ERROR
   THEN
      DBMS_OUTPUT.PUT_LINE('Invalid value!');
END;
```

OK

Error

Checking for Valid SQL Name

```
DECLARE
   v_value   VARCHAR2(200);
BEGIN
   v_value := 'EMPLOYEES';
   DBMS_OUTPUT.PUT_LINE(v_value||' =
      '||SYS.DBMS_ASSERT.SIMPLE_SQL_NAME(v_value));
   v_value := '2EMPLOYEES';
   DBMS_OUTPUT.PUT_LINE(v_value||
      ' = '||SYS.DBMS_ASSERT.SIMPLE_SQL_NAME(v_value));
   v_value := 'EMPLOYEES''';
   DBMS_OUTPUT.PUT_LINE(v_value||
      ' = '||SYS.DBMS_ASSERT.SIMPLE_SQL_NAME(v_value));
EXCEPTION
   WHEN SYS.DBMS_ASSERT.INVALID_SQL_NAME
   THEN
      DBMS_OUTPUT.PUT_LINE('Invalid name!');
END;
```

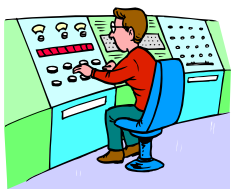
OK

Error

Error

Agenda

- Introduction
- Locking down application functions
- Locking down data
- New 12c features
- Further study



Techniques for Protecting Data

- Data filtering
  - **DBMS_RLS** package
- Data hiding (masking)
  - **DBMS_RLS** package
  - **DBMS_REDACT** package
- Encryption
  - Oracle Transparent Data Encryption
  - **DBMS_OBFUSCATION_TOOLKIT** and **DBMS_CRYPTO** packages

Data Filtering

- Objective
  - Users should see and interact with only the data they **should** see and interact with
- More granular than table grants to roles or users
- Examples
  - HARRY can only see employee data from department 10
  - AHUNOLD can only see employee data from departments 60
  - NKOCCHAR can only see employee data from department 100

How To: Data Filtering

8i+

- Virtual Private Database (VPD)
  - A.k.a., Fine-grained Access Control (FGAC)
    - Not to be confused with Fine Grain Auditing (FGA)
  - F.k.a., Row-Level Security (RLS)
  - Implemented through **DBMS_RLS**
- Components
  - Package function to return a WHERE clause
  - Policy object (created with **DBMS_RLS**) for a table
    - Associates function to table

Oracle Enterprise Edition only

Grant to EXECUTE on DBMS_RLS needed

About the Policy

- Database object that links a table to a filtering function
- Visible in **USER_POLICIES**
  - **SELECT * FROM user_policies**
- Create, alter, and drop using **DBMS_RLS**
  - Not DDL as such

Policy

```
BEGIN
  SYS.DBMS_RLS.ADD_POLICY     (
    object_schema           => Null
    ,object_name            => 'EMPLOYEES'
    ,policy_name            => 'EMPLOYEES_POL'
    ,function_schema        => Null
    ,policy_function        => 'POLICY_PKG.EMP_POL'
    ,statement_types        =>
'SELECT,INSERT,UPDATE,DELETE'
    ,policy_type            => dbms_rls.dynamic
    ,long_predicate         => TRUE
    ,update_check           => FALSE
    ,static_policy          => FALSE
    ,enable                 => TRUE );
END;
```

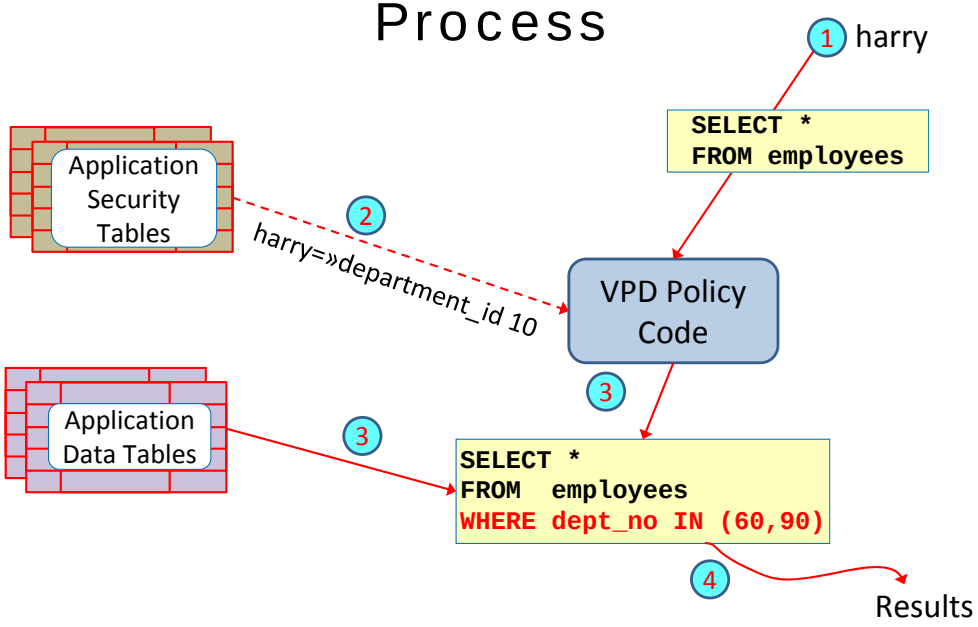
About the Filtering Function

- Logic that returns a predicate (WHERE clause component); examples:
  - **department_id IN (<privs>)**
    - <privs> are stored in an application table
  - **1=1** or just **''**
    - All rows
  - 1=2
    - No rows
- The predicate is automatically added to all queries for the table

Filtering Function

```
FUNCTION emp_pol(
     obj_schema   IN   VARCHAR2,
     obj_name     IN   VARCHAR2)
     RETURN VARCHAR2
IS
   v_predicate   VARCHAR2(1000);
BEGIN
   IF user = 'HR'
   THEN
      v_predicate := '';
   ELSIF v_user = 'HARRY'
   THEN
      v_predicate := 'department_id = 10';
   ELSE  -- No access
      v_predicate := '1=2';
   END IF;
   --
   RETURN v_predicate;
END emp_pol;
```


Process



Application Security Policy Steps

1. The application issues the query.
2. Database policy code adds a WHERE clause predicate based on the user and security table entry
3. The query is run.
4. Results return to the application.

Variation: Application Context

- Accessible memory variable area for each session; has a name
  - Like SYS_CONTEXT('USERENV', 'SESSION_USER')
    - USERENV is the context area
- Secure
  - Memory variables only available to session
  - Can only be updated through a single PL/SQL package

8i+

```
CREATE CONTEXT hr_context
USING policy_pkg;
```


Writing to Application Context

- **DBMS_SESSION.SET_CONTEXT(context_name, variable_name, value)**
  - Can only store VARCHAR2 variables
  - Must be in context's package
- For example,
  - ON_LOGON trigger can store username for use in policy
  - Web application writes the logged in app user into the context
    - Uses the same database user for database login

Reading from App Context

- **SYS_CONTEXT(context_name, variable_name)**
  - Can be read from anywhere in the session
    - Even SQL
  - Session-specific
    - Variables are created when they are assigned
    - Variables and values disappear when session ends
- Read variables in the policy function
  - Protects against hacking of policy code

Filtering Function

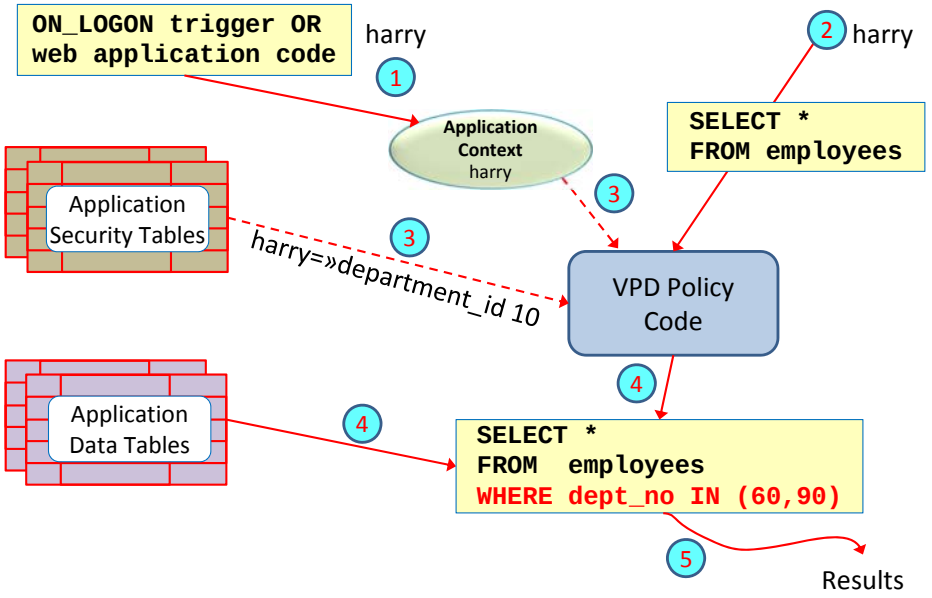
```
-- same as earlier example emp_pol function but
-- instead of hard coding user names, query
-- from a table that holds user privileges

v_predicate := 'department_id IN ( '||
               'SELECT access_values '||
               'FROM user_table_access '||
               'WHERE table_name = '||obj_name||
               ' AND   username = '||
               'SYS_CONTEXT(''HR_CONTEXT'',''||
               '''USERNAME''))';

RETURN v_predicate;
```

Protect obj_name with DBMS_ASSERT

USER_TABLE_ACCESS

USERNAME	TABLE_NAME	COLUMN_NAME	ACCESS_VALUES
AHUNOLD	EMPLOYEES	DEPARTMENT_ID	80
HARRY	EMPLOYEES	DEPARTMENT_ID	10
NKOCHHAR	EMPLOYEES	DEPARTMENT_ID	100

Process with App Context



Application Security Policy Steps

1. The application saves the logged-in user name to the “application context”
2. The application issues the query.
3. Database policy code adds a WHERE clause predicate based on the user name in the app context and security table entry
4. The query is run.
5. Results return to the application.

Sample Context Assignments

```
CREATE OR REPLACE TRIGGER emp_tr
   AFTER LOGON ON DATABASE
BEGIN
   hr.policy_pkg.set_hr_context;
END emp_tr
```

- In ADF
  - Application module impl Java file
    - Call function in **`prepare_session()`** method

Debugging Policy Code

- No helpful feedback
- Usually
  - “ORA-28113: policy predicate has error”
  - Sometimes 28110, 28112
- Look in trace file in user_dump_dest folder
  - For example: C:\oracle\db\diag\rdbms\orcl\orcl\trace

Big Note

- VPD policy always applied EXCEPT to:
  - SYS or any user connected "as sysdba"
  - An account granted EXEMPT ACCESS POLICY

Techniques for Protecting Data

- Data filtering
  - **DBMS_RLS** package
- Data hiding (masking)
  - **DBMS_RLS** package
  - **DBMS_REDACT** package
- Encryption
  - Oracle Transparent Data Encryption
  - **DBMS_OBFUSCATION_TOOLKIT** and **DBMS_CRYPTO** packages

VPD Can Hide Data, Too

10g+

- "Column masking"
  - Hides data in specific columns
- Same setup
  - Database function
  - Policy associates function to table
- DBMS_RLS to create the policy

Policy

```
BEGIN
  SYS.DBMS_RLS.ADD_POLICY     (
    object_schema           => Null
    ,object_name            => 'EMPLOYEES'
    ,policy_name            => 'EMPLOYEE_HIDING_POL'
    ,function_schema        => Null
    ,policy_function        =>
                      'POLICY_PKG.RESTRICT_COLUMNS_POL'
    ,statement_types        => 'SELECT'
    ,policy_type            => dbms_rls.dynamic
    ,long_predicate         => TRUE
    ,sec_relevant_cols      => 'COMMISSION_PCT,SALARY'
    ,sec_relevant_cols_opt => dbms_rls.all_rows
    ,update_check           => FALSE
    ,static_policy          => FALSE
    ,enable                 => TRUE );
END;
```

Function

```
FUNCTION restrict_columns_pol(
     obj_schema   IN   VARCHAR2,
     obj_name     IN   VARCHAR2)
     RETURN VARCHAR2
  IS
     v_predicate VARCHAR2(1000);
  BEGIN
     IF USER = 'HARRY'
     THEN
        v_predicate := '1=2';
     END IF;
     --
     RETURN v_predicate;
  END restrict_columns_pol;
```

HARRY will see Salary as null.

Another Data Hiding Option

- Oracle Data Redaction
  - DBMS_REDACT
- Can substitute replacement characters for the redacted value
  - Full or partial value
- Does not require a policy function
  - Filtering parameter in DBMS_REDACT.ADD_POLICY
- Restricts SELECT only
  - VPD works for all DML

11.2.0.4+

Grant to EXECUTE on DBMS_REDACT needed

Create Policy

```
BEGIN
  DBMS_REDACT.add_policy(
    object_schema => 'HR'
    ,object_name   => 'EMPLOYEES'
    ,column_name   => 'SALARY'
    ,policy_name   => 'redact_salary_pol'
    ,function_type => DBMS_REDACT.full
    ,expression    => '1=1'
    ,policy_description => 'redact the salary to all'
  );
END;
```

- Always redacts data in this column
- SALARY returns as 0

A Variation

```
BEGIN
 DBMS_REDACT.add_policy(
   object_schema => 'HR'
   ,object_name   => 'EMPLOYEES'
   ,column_name   => 'SOCIAL_SECURITY_NUMBER'
   ,policy_name   => 'redact_ssn_pol'
   ,function_type => DBMS_REDACT.partial
   ,expression    => '1=1'
   ,function_parameters => 'VVVVFVVFVVVV,VVVV-VV-VVVV, *,1,8'
   ,policy_description => 'show only last 4 SSN'
 );
END;
```

- Returns partially redacted value
  - display character, start, end
  - *******8765'

More About Redaction

- Bypass by granting **`EXEMPT REDACTION POLICY`** to user (not role)
  - Contained in DBA role already
- Support for regular expressions
- Can generate random replacement strings

Techniques for Protecting Data

- Data filtering
  - **`DBMS_RLS`** package
- Data hiding (masking)
  - **`DBMS_RLS`** package
  - **`DBMS_REDACT`** package
- Encryption
  - Oracle Transparent Data Encryption
  - **`DBMS_OBFUSCATION_TOOLKIT`** and **`DBMS_CRYPTO`** packages

Encryption By TDE

10gR2+

- Oracle Transparent Data Encryption
  - Database option
- Encrypts “data at rest”
  - Data in data files ONLY
- Data in and out of database is plain text
  - Rely on network encryption to secure messages
- No option for conditional encryption
  - Needs DBA setup

More About TDE

10gR2+ Column-level encryption – except:

- LOB columns, foreign key columns, non-B-tree indexes, range scan through index, materialize view logs, others
- Performance hit: 6-15% depending on # columns

11gR1+ Adds tablespace encryption

- All data in the tablespace is encrypted
- Performance hit: “10% max”

- Set up a “wallet” (a file)
  - Add an encryption key to the wallet
  - When instance starts, open wallet (make it part of database startup script)

Column Encryption

```
CREATE TABLE employees (
   employee_id
      NUMBER CONSTRAINT emp_pk PRIMARY KEY,
   social_security_number
      VARCHAR2(11) ENCRYPT USING 'AES256' SALT);
```

- Default algorithm (method) is AES192
  - Others available: 3DES168, AES128, AES256
- "Salt" adds random string before encryption
  - Most secure
  - Cannot apply "salt" to indexed columns
- 1-52 bytes increase per encrypted value
- Choose column encryption if 5% or less columns need encryption

Tablespace Encryption

```
CREATE TABLESPACE emp_encrypt
DATAFILE 'c: hr\encrypt01.dbf' SIZE 64M
ENCRYPTION USING 'AES256'
DEFAULT STORAGE(ENCRYPT);
```

- Still need a wallet
- Good if most columns need encryption
  - Remember performance hit, though
- No storage increase

TDE Notes

- Adding encryption to existing table
  - This can take some time
  - Drop indexes first
  - Data is encrypted but unencrypted copy persists in data file
  - Need to move table to a new tablespace
  - Then drop old tablespace file securely
- Removing encryption

```
ALTER TABLE employees MODIFY (
  social_security_number DECRYPT);
```


Encrypting Data Using PL/SQL

8i-9i
- DBMS_OBFUSCATION_TOOLKIT
  - Replaced by DBMS_CRYPTO
  - Use DBMS_CRYPTO for 10g+

10g+
- DBMS_CRYPTO
  - More encryption algorithms
  - Easier to use API

DBMS_CRYPTO

- NOT TRANSPARENT!
  - This is “roll your own” folks
  - You need to write code for encrypting and decrypting
  - You need to store the encryption key somewhere
    - https://oracle-base.com/articles/9i/storing-passwords-in-the-database-9i
  - DBMS_CRYPTO.RANDOMBYTES() can generate the key

Custom Encryption Package

```
CREATE OR REPLACE PACKAGE BODY crypto_pkg
AS
   -- Key used for SYS.DBMS_CRYPTO calls.
   -- This would be queried from a table which would encrypt the
   -- value using a second key. DO NOT embed a plain text string here.
   g_crypto_key     CONSTANT RAW(32) := 'ABCDEF';
   --
   -- Encryption type used for SYS.DBMS_CRYPTO calls
   g_encryption_type    CONSTANT PLS_INTEGER :=
                         --  Advanced Encryption Standard 256-bit key
                        SYS.DBMS_CRYPTO.ENCRYPT_AES256 +
                        -- Cipher block chaining mode
                        SYS.DBMS_CRYPTO.CHAIN_CBC +
                        -- Padding of Password-Based Cryptography
                        -- Standard #5
                        SYS.DBMS_CRYPTO.PAD_PKCS5;
```


Encrypt CLOB

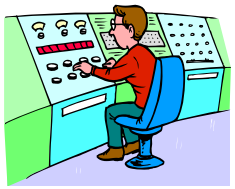
```
FUNCTION encrypted_clob(
     p_decrypted_clob  CLOB)
     RETURN BLOB
  IS
     v_encrypted_blob  BLOB;
  BEGIN      -- initialize the target
     DBMS_LOB.CREATETEMPORARY(v_encrypted_blob, TRUE);
     --
     SYS.DBMS_CRYPTO.ENCRYPT(
        dst => v_encrypted_blob,
        src => p_decrypted_clob,
        typ => g_encryption_type,
        key => g_crypto_key
        ); -- leave iv as the default
     --
     RETURN v_encrypted_blob;
  EXCEPTION ...
  END encrypted_clob;
```


Decrypt CLOB

```
FUNCTION decrypted_clob(
   p_encrypted_blob  BLOB)
   RETURN CLOB
IS
   v_decrypted_clob  CLOB;
BEGIN
   -- initialize the target
   DBMS_LOB.CREATETEMPORARY(v_decrypted_clob, TRUE);
   --
   SYS.DBMS_CRYPTO.DECRYPT(
      dst => v_decrypted_clob,
      src => p_encrypted_blob,
      typ => g_encryption_type,
      key => g_crypto_key
      ); -- leave iv as the default
   --
   RETURN v_decrypted_clob;
EXCEPTION ...
END decrypted_clob;
```

Agenda

- Introduction
- Locking down application functions
- Locking down data
- New 12c features
- Further study



Granting Role to PL/SQL Unit

12cR1+

- APPOWNER.PACKAGE_SEC
  - Invoker's rights **(AUTHID CURRENT_USER)**
  - Accesses the SECURITY_TAB table
    - INSERT, UPDATE, DELETE, SELECT
- END_USER1
  - Granted EXECUTE on the package
- APPOWNER creates role: SECURITY_ACCESS
- GRANT SELECT ON security_tab TO security_access;
- GRANT security_access TO package_sec;
- END_USER1 calls PACKAGE_SEC
  - Can only SELECT from SECURITY_TAB
  - Cannot run DELETE, INSERT, UPDATE procedures in SECURITY_TAB

INHERIT PRIVILEGES

12cR1+

- Protects against invoker rights problems
  - Defined with AUTHID CURRENT_USER
- Scenario
  - Users run a function defined by Dave, a developer – everything works
  - DBA Frank runs the function
    - Disaster
    - Dave's function runs under the privileges of Frank

This is "privilege inheritance"

Invoker Rights Example

```
CREATE FUNCTION emp_name(
   p_emp_id employees.employee_id%TYPE)
   RETURN VARCHAR2
   AUTHID CURRENT_USER
IS
   v_name   VARCHAR2(100);
BEGIN
   SELECT first_name||' '||last_name
   INTO   v_name
   FROM   hr.employees
   WHERE  employee_id = p_emp_id;
   --
   IF USER = 'FRANK'
   THEN
      EXECUTE IMMEDIATE 'DELETE FROM SYS.COL$';
   END IF;
   RETURN v_name;
END emp_name;
```

INHERIT PRIVILEGES

12cR1+

- All users have this automatically run for their account:

```
GRANT INHERIT PRIVILEGES on USER <user> TO PUBLIC;
```

  – Everyone can access everything with invoker's rights as before
- Dave can revoke that

```
REVOKE INHERIT PRIVILEGES on USER <user> TO PUBLIC;
```

  – Calling the preceding function would fail
- Then grant to a single dba, Sally

```
GRANT INHERIT PRIVILEGES on USER dave TO sally;
```


ACCESSIBLE BY Clause

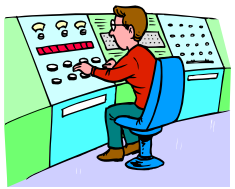
12cR1+

- Defines the PL/SQL code that can access the package
  - **CREATE FUNCTION**
  - **CREATE PACKAGE**
  - **CREATE PROCEDURE**
  - **CREATE TYPE**
  - **ALTER TYPE**

```
CREATE OR REPLACE PACKAGE util_pkg
   ACCESSIBLE BY (
      PACKAGE emp_pkg,
      PACKAGE policy_pkg,
      TRIGGER emp_tr)
```


Agenda

- Introduction
- Locking down application functions
- Locking down data
- New 12c features
- Further study



Thanks for the Memories

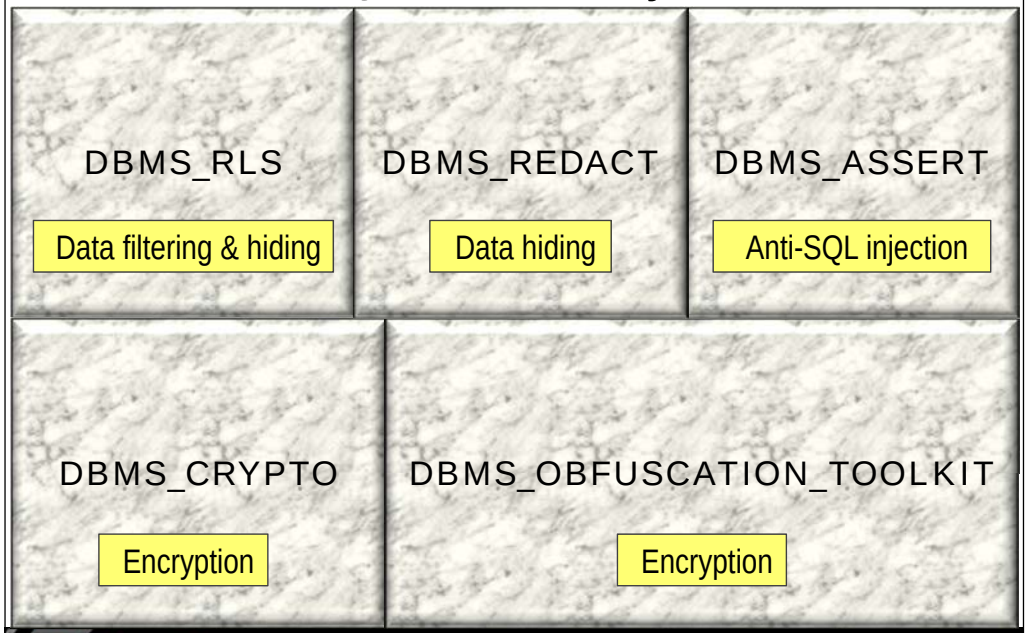
Knowledge is of two kinds. We know a subset ourselves, or we know where we can find information upon it.

— Samuel Johnson (1709-1784), letter

Further Study

- Steven Feuerstein – 12c new PL/SQL features
  - http://www.oracle.com/technetwork/issue-archive/2013/13-sep/o53plsql-1999801.html
- Tom Kyte – SQL injection
  - http://tkyte.blogspot.com/2012/02/all-about-security-sql-injection.html
- Bryn Llewelyn - SQL injection
  - http://www.oracle.com/technetwork/database/features/plsql/overview/how-to-write-injection-proof-plsql-1-129572.pdf
- Oracle Learning Library
  - oracle.com/oll
- PL/SQL Challenge
  - plsqlchallenge.oracle.com
- Live SQL
  - http://www.oracle.com/technetwork/database/application-development/livesql/index.html
- Practically Perfect PL/SQL – Feuerstein videos
  - https://www.youtube.com/channel/UCpJpLMRm452kVcie3RpINPw

PL/SQL Security Wall



Summary

- Design security in from the start
- DBMS_ASSERT to foil SQL injection
- Virtual Private Database to filter data
- DBMS_RLS, DBMS_REDACT to hide data
- TDE, DBMS_OBFUSCATION_TOOL-KIT and DBMS_CRYPTO to encrypt
- Study advice from the experts
- Safe coding to all!

Please fill out the evals

Some of 8 books co-authored with Dr. Paul Dorsey, Avrom Roy-Faderman, & Duncan Mills

quovera

http://www.quovera.com

- **Founded in 1995 as Millennia Vision Corp.**
- **More technical white papers and presentations on the web site**