TWIN CITIES ORACLE Users Group

# Oracle Security

Spring 2018

Daniel A. Morgan
email: vp@tcoug.org
mobile: +1 206-669-2949

Wednesday: April 18, 2018

# SQL Rewrite Vulnerabilities

Dan Morgan

TWIN CITIES ORACLE USERS GROUP

# What Is A Rewrite Vulnerability?

- Rewrite occurs when the database optimizer transparently replaces executed SQL and PL/SQL with a completely different statement
- The replacement statement may improve performance
- The replacement statement may be the worst Cartesian Join you can imagine
- The replacement statement may breach your carefully crafted security
- There are three places in Oracle where rewrite occurs in most databases
  - Optimizer Rewrites
    - Enabled rewrites such as **`STAR_TRANSFORMATION_ENABLED`**
    - By default the Oracle database will rewrite every DML statement is processes
    - The only way you can stop this rewrite is with SQL baselines or with full hinting
  - Optimizer rewrites will never change the nature of statement and thus cannot, in and of themselves, constitute a security risk

# Full Hinting (an example by Jonathan Lewis)

Consider, for example:

```
SELECT /*+ index(t1 t1_abc) index(t2 t2_abc) */ COUNT(*)
FROM t1, t2
WHERE t1.col1 = t2.col1;
```

For weeks, this may give you the plan:

```
   NESTED LOOP
     table access by rowid t1
       index range scan t1_abc
     table access by rowid t2
       index range scan t2_abc
```

Then, because of changes in statistics, or init.ora parameters, or nullity of a column, or a few other situations that may have slipped my mind at the moment, this might change to:

```
   HASH JOIN
      table access by rowid t2
        index range scan t2_abc
      table access by rowid t1
        index range scan t1_abc
```

Your hints are still obeyed, the plan has changed. On the other hand, if you had specified:

```
SELECT /*+ no_parallel(t1) no_parallel(t2) no_parallel_index(t1) no_parallel_index(t2)
ordered use_nl(t2) index(t1 t1_abc) index(t2 t2_abc) */ COUNT(*)
FROM t1, t2
WHERE t1.col1 = t2.col1;
```

Then I think you could be fairly confident that there was no way that Oracle could obey the hints whilst changing the access path.

# Materialized View Rewrites

- Materialized View Rewrites must be authorized through DDL and instruct a query to consider using a Materialized View in place of a table
- Here are some examples of explicit MV rewrite authorizations

```
CREATE MATERIALIZED VIEW mv_rewrite
TABLESPACE uwdata
REFRESH ON DEMAND
ENABLE QUERY REWRITE
AS SELECT s.srvr_id, i.installstatus, COUNT(*)
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id
GROUP BY s.srvr_id, i.installstatus;

ALTER SYSTEM SET query_rewrite_enabled = TRUE;
ALTER SYSTEM SET query_rewrite_enabled = FORCE;
ALTER SESSION SET query_rewrite_integrity = ENFORCED;
ALTER SESSION SET query_rewrite_integrity = STALE_TOLERATED;
ALTER SESSION SET query_rewrite_integrity = TRUSTED;
```

- Materialized View rewrites will never change the nature of statement and thus cannot, in and of themselves, constitute a security risk

# What Is A Rewrite Vulnerability?

- But there are 3 rewrite capabilities that are far more powerful and thus far more dangers ... you need to be aware of them
  - DBMS_ADVANCED_REWRITE
  - DBMS_SQL_TRANSLATOR
  - DBMS_SQLDIAG

# DBMS_ADVANCED_REWRITE

- This package contains interfaces that can be used to create, drop, and maintain functional equivalence declarations for query rewrites
- According to the Oracle docs: "To gain access to these procedures, you must connect as SYSDBA and explicitly grant execute access to the desire database administrators"

```
SQL> SELECT grantee
  2  FROM dba_tab_privs
  3  WHERE table_name = 'DBMS_ADVANCED_REWRITE'
  4  ORDER BY 1;

no rows selected
```

- But should someone gain execute privilege on the package, for example through any one of a number of means they can do this

```
dbms_advanced_rewrite.declare_rewrite_equivalence(
name             VARCHAR2,
source_stmt      CLOB,
destination_stmt CLOB,
validate         BOOLEAN  := TRUE,
rewrite_mode     VARCHAR2 := 'TEXT_MATCH');
```

and have the optimizer swap the authentic statement for one they crafted

# DBMS_SQL_TRANSLATOR

- The Oracle docs state: " When translating a SQL statement or error, the translator package procedure will be invoked with the same current user and current schema as those in which the SQL statement being parsed. The owner of the translator package must be granted the TRANSLATE SQL user privilege on the current user. Additionally, the current user must be granted the EXECUTE privilege on the translator package."
- The declared business case for this package is that it can be used to intercept TransactSQL calls to an Oracle database and allow the database owner to translate those that would fail into Oracle SQL or PL/SQL

```
dbms_sql_translator.register_sql_translation(
profile_name    IN VARCHAR2,
sql_text        IN CLOB,
translated_text IN CLOB    DEFAULT NULL,
enable          IN BOOLEAN DEFAULT TRUE);
PRAGMA SUPPLEMENTAL_LOG_DATA(register_sql_translation, AUTO_WITH_COMMIT);
```

```
exec dbms_sql_translator.register_sql_translation(
profile_name =>'UW_TSQLTRANS',
sql_text =>'SELECT srvr_id INTO uwclass.tsql_target FROM uwclass.servers',
translated_text =>'INSERT INTO uwclass.tsql_target SELECT srvr_id FROM uwclass.servers');
```

# DBMS_SQLDIAG

- DBMS_SQLDIAG is part of the Oracle Diagnostic Pack and contains the procedure CREATE_SQL_PATCH
- A SQL patch, as used by this procedure, is a set of user specified hints for specific statements identified by the SQL text
- When considering this as a vulnerability consider the following
  - By default EXECUTE is granted to PUBLIC
  - Hints can be used to override configuration settings such as PARALLEL DEGREE and have the effect of substantially degrading performance and oversubscribing resources

```
dbms_sqldiag.create_sql_patch(
sql_text    IN CLOB,
hint_text   IN CLOB,
name        IN VARCHAR2 := NULL,
decription  IN VARCHAR2 := NULL,
category    IN VARCHAR2 := NULL,
validate    IN BOOLEAN  := TRUE)
RETURN VARCHAR2;
```

```
SQL> DECLARE
  2    stxt CLOB := 'SELECT /* CREATE_PATCH2 */ COUNT(*), MAX(siid)
FROM uwclass.serv_inst WHERE srvr_id = :srvrid';
  3    htxt CLOB := 'BIND_AWARE';
  4    retVal VARCHAR2(60);
  5  BEGIN
  6     retVal := sys.dbms_sqldiag.create_sql_patch(stxt, htxt);
  7  END;
  8  /

PL/SQL procedure successfully completed.
```

# An IT Terrorist Attack (7:7)

- How many of Oracle's vulnerability enhancing defaults such as grants of EXECUTE to PUBLIC have you disabled?
- No better time to start than tomorrow