Securing PL/SQL Applications with DBMS_ASSERT

David Litchfield [davidl@ngssoftware.com]
25th October 2005

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

Introduction

Over the past few years Oracle has fixed a large number of PL/SQL injection vulnerabilities in their database server product. The vulnerability arises due to procedures and functions accepting user input and performing no validation on it before passing off to be executed in an SQL query. By carefully crafting their input an attacker can inject nefarious SQL and gain complete control over the database server under the right circumstances; the right circumstances are provided by being able to inject into a PL/SQL procedure or function owned by a high privileged user such as SYS, SYSMAN, MDSYS, CTXSYS or WKSYS and which has not been defined with the AUTHID CURRENT_USER keyword.

To help combat this class of attack Oracle has introduced the DBMS_ASSERT PL/SQL package. Whilst integrated into Oracle 10g Version 2 from day one, the DBMS_ASSERT was introduced into 10g Version 1 as part of the October 2005 Critical Patch Update. As a security researcher, it is excellent to see Oracle finally making the right positive moves in the direction of greater security.

The DBMS_ASSERT package exports a number of functions that can be used to validate user input. There are seven functions available namely NOOP, SCHEMA_NAME, SQL_OBJECT_NAME, QUALIFIED_SQL_NAME, SIMPLE_SQL_NAME, ENQUOTE_NAME and ENQUOTE_LITERAL. Wherever your PL/SQL applications embed values, input or data in SQL statements which are then executed, these functions should be used to ensure that whatever's going into the query cannot be used inject arbitrary SQL. Even if the source of the data or value is considered as trusted I'd recommend erring on the side of caution and going ahead and validating.

The DBMS_ASSERT functions

Each of these functions takes as their first argument a string (VARCHAR2). There is a second version of the NOOP function which takes a CLOB as an argument. The ENQUOTE_NAME function takes a second parameter, a BOOLEAN, which determines whether the first argument should be converted to upper case or not; the default is TRUE.

The **NOOP** function.

This function, as the name describes, performs no operation. Whatever string is passed in as the function's argument is simply returned.

```
SQL> SELECT SYS.DBMS_ASSERT.NOOP('NGSSQUIRREL') FROM DUAL
SYS.DBMS_ASSERT.NOOP('NGSSQUIRREL')
--------------------------------------------------------------------
NGSSQUIRREL

SQL> SELECT SYS.DBMS_ASSERT.NOOP('NGSSQ  UIRREL') FROM DUAL
SYS.DBMS_ASSERT.NOOP('NGSSQ  UIRREL')
--------------------------------------------------------------------
NGSSQ  UIRREL

SQL> SELECT SYS.DBMS_ASSERT.NOOP('NGSSQ'' /**/ -- UIRREL') FROM DUAL
SYS.DBMS_ASSERT.NOOP('NGSSQUIRREL')
--------------------------------------------------------------------
NGSSQ' /**/ -- UIRREL
```

I'd recommend not using this function, even if you consider the data to have come from a trusted source. Too often during code reviews we come across secondary or tertiary attacks that rely on assumptions and trust.

The ***SCHEMA_NAME*** function.

This function is used to check whether schema exists or not. The function works by running the query "SELECT COUNT(*) FROM ALL_USERS WHERE USERNAME = SCHEMA" where "SCHEMA" is the name of the schema to check. If the schema exists the schema is returned but an error is raised if the schema does not exist.

```
SQL> SELECT SYS.DBMS_ASSERT.SCHEMA_NAME('NO_SUCH_SCHEMA') FROM DUAL;
SELECT SYS.DBMS_ASSERT.SCHEMA_NAME('NO_SUCH_SCHEMA') FROM DUAL
               *
Error at line: 1
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 267

SQL> SELECT SYS.DBMS_ASSERT.SCHEMA_NAME('MDSYS') FROM DUAL;
SYS.DBMS_ASSERT.SCHEMA_NAME('MDSYS')
------------------------------------
MDSYS
```

The ***SQL_OBJECT_NAME*** function.

This function checks to see whether the supplied string is a valid extant object in the database. It does this by passing the first argument to the DBMS_UTILITY.NAME_RESOLVE procedure which performs the check.

Object names can be unquoted

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('ALL_OBJECTS') FROM DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('ALL_OBJECTS')
----------------------------------------------
ALL_OBJECTS
```

Object names can be quoted

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('"ALL_OBJECTS"') FROM DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('"ALL_OBJECTS"')
------------------------------------------------
"ALL_OBJECTS"
```

Object names can be specified with the owner

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('SYS.ALL_OBJECTS') FROM
DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('SYS.ALL_OBJECTS')
--------------------------------------------------
SYS.ALL_OBJECTS
```

Object names can be specified with the owner and quoted

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('"SYS"."ALL_OBJECTS"') FROM
DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('"SYS"."ALL_OBJECTS"')
------------------------------------------------------
"SYS"."ALL_OBJECTS"
```

Invalid objects will cause an error

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQuirreL') FROM DUAL;
SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQuirreL') FROM DUAL;
       *
Error at line: 1
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 307
```

There is one caveat, here. If you insert an at sign (@) making it appear as if there is a database link no object validation is performed even if the database link does not exist:

```
SQL> SELECT SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQuirreL@Real_dbLink')
FROM DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQUIRREL@REAL_DBLINK')
------------------------------------------------------------
NGSSQuirreL@Real_dbLink

SQL> SELECT
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQuirreL@no_such_dbLink') FROM
DUAL;
SYS.DBMS_ASSERT.SQL_OBJECT_NAME('NGSSQUIRREL@NO_SUCH_DBLINK')
-------------------------------------------------------------
NGSSQuirreL@no_such_dbLink
```

A potential security risk could creep in due to this. To examine this let's consider a contrived situation. For a moment assume an attacker has the CREATE PUBLIC DATABASE LINK permission. Also assume there is a PL/SQL procedure that takes as an argument the name of a table in a schema specified by the procedure which is then queried. If data from this table is trusted and consequently not sanitized before it itself it embedded in a query then an attacker could redirect the procedure to acquire data from their own database server.

```
..
..
STMT1:= 'SELECT VAL FROM "SYS"."' ||
DBMS_ASSERT.SQL_OBJECT_NAME(USER_SUPPLIED_TABLE) || '" WHERE X=1';
EXECUTE IMMEDIATE STMT2 INTO BUFFER;
-- Assumption: as only SYS can insert into the table which exists
-- in the sys schema we can trust the data
STMT2 :='SELECT FOO FROM "SYS"."BAR" WHERE PQR = ''' ||
DBMS_ASSERT.NOOP(BUFFER);
..
..
```

The fact that the table's location is restricted to the SYS schema is irrelevant if a table on a database link is passed for USER_SUPPLIED_TABLE. It'll be the remote SYS schema that the data is drawn from and as this remote database is owned by and under the control of the attacker they can fully influence what goes into BUFFER.

The ***SIMPLE_SQL_NAME*** function.

This function checks that characters in an SQL element comprise only of A-Z, a-z, 0-9, $, # and _. If and only if the SQL element is quoted with double quotes then other characters are allowed.

```
SQL> SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('ABCD789$#_zxcvbnm') FROM
DUAL;
DBMS_ASSERT.SIMPLE_SQL_NAME('ABCD789$#_ZXCVBNM')
```

```
------------------------------------------------
ABCD789$#_zxcvbnm

/ * @ signs are not allowed therefore database links aren't allowed */
SQL> SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('foo@bar') FROM DUAL;
SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('foo@bar') FROM DUAL;
               *
Error at line: 1
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 174

/* non-quoted example with non-alphanumeric */
SQL> SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('Z%@:**') FROM DUAL;
SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('Z%@:**') FROM DUAL;
               *
Error at line: 1
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 174


/* quoted example with non-alphanumeric*/
SQL> SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('"Z%@:**"') FROM DUAL;
SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('"Z%@:**"')
------------------------------------------
"Z%@:**"
```

The ***QUALIFIED_SQL_NAME*** function.

This function is very similar to the SIMPLE_SQL_NAME function but allows for database links.

```
SQL> SELECT SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('foo@bar') FROM DUAL;
SYS.DBMS_ASSERT.SIMPLE_SQL_NAME('FOO@BAR')
-----------------------------------------
foo@bar
```

The ***ENQUOTE_NAME*** function.

This function enquotes the argument if it has not already been enquoted, that is if the user supplied string has not been enclosed in double quotes already, the function will do it.

```
SQL> SELECT SYS.DBMS_ASSERT.ENQUOTE_NAME('ngssquirrel') FROM DUAL;
SYS.DBMS_ASSERT.ENQUOTE_NAME('NGSSQUIRREL')
------------------------------------------
"NGSSQUIRREL"
```

The ***ENQUOTE_LITERAL*** function.

This function encloses the user supplied string in single quotes if it has not already been done.

```
SQL> SELECT SYS.DBMS_ASSERT.ENQUOTE_LITERAL('ngssquirrel') FROM DUAL;
SYS.DBMS_ASSERT.ENQUOTE_LITERAL('NGSSQUIRREL')
---------------------------------------------
'ngssquirrel'
```

Further Reading

Oracle PL/SQL Injection
http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-litchfield.pdf
David Litchfield

The Database Hacker's Handbook
http://www.amazon.com/exec/obidos/tg/detail/-/0764578014/103-1229345-0451064?v=glance
NGSSoftware

SQL Injection and Oracle
http://online.securityfocus.com/infocus/1644
Pete Finnigan

An Introduction to SQL Injection attacks for Oracle Developers
http://www.integrigy.com/info/IntegrigyIntrotoSQLInjectionAttacks.pdf
Stephen Kost