Advanced Exploitation of Oracle PL/SQL Flaws

David Litchfield
(davidl@ngssoftware.com)

Objectives

- Discuss current “threat landscape”
- Introduce a new class of vulnerability
- Introduce a new method of attack
- Show practical demonstrations
- Look at some defences

Agenda

- PL/SQL Risks
  - SQL Injection
  - “Dangling” Cursor Snarfing
  - Cursor Injection
- Demonstrations
  - Grant DBA Privileges
  - Indirect Privilege Escalation

What is PL/SQL?

- Procedural Language / Structured Query Language
- Oracle’s extension to standard SQL

  Programmable like T-SQL in the Microsoft world.
- Used to create
  - Stored Procedures
  - Functions
  - Packages (collections of procedures and functions)
  - Triggers
  - Objects
- Extends functionality with External Procedures and Java

Privileges – Definer vs. Invoker rights

- PL/SQL executes with the privileges of the definer
    - A procedure owned by SYS executes with SYS privileges
- AUTHID CURRENT_USER keyword
    - PL/SQL created using the AUTHID CURRENT_USER keyword executes with the privileges of the invoker
    - A procedure owned by SYS but called by SCOTT executes with the privileges of SCOTT
- Analogous to Suid programs in the *nix world.

Running SQL from PL/SQL

- EXECUTE IMMEDIATE ‘…’
- OPEN
- DBMS_SQL
  - Key to Cursor Snarfing and Cursor Injection

DBMS_SQL

```
DECLARE
MY_CURSOR NUMBER;
MY_RESULT NUMBER;
BEGIN
MY_CURSOR:=DBMS_SQL.OPEN_CURSOR();
DBMS_SQL.PARSE(MY_CURSOR,
'SELECT 1 FROM DUAL',0);
MY_RESULT:=DBMS_SQL.EXECUTE(MY_CURSOR);
END;
/
```

DBMS_SQL Cursors

- *Cursors are numbers… start from 1 to 300*
- *Unique to a specific session*
- *Like a handle – remains open ‘til closed*
- *If an exception occurs and the cursor is not closed in “cleanup” routines then the cursor is left “dangling”.*

Cursor Snarfing

- *If an attacker can cause an exception in higher privileged code where there are no cleanup routines then the attacker can re-use that cursor and gain access – sometimes limited, sometimes complete.*

- *Simple example – csnarf.txt*

- *We’ll come back to snarfing in a moment…*

Contrived Example vulnerable procedure

```
CREATE OR REPLACE PROCEDURE GET_OWNER (P_OBJNM VARCHAR) IS
TYPE C_TYPE IS REF CURSOR;
CV C_TYPE;
BUFFER VARCHAR2(200);
BEGIN
    DBMS_OUTPUT.ENABLE(1000);
    OPEN CV FOR 'SELECT OWNER FROM ALL_OBJECTS WHERE
   OBJECT_NAME = ''' || P_OBJNM ||'''';

    LOOP
        FETCH CV INTO BUFFER;
        DBMS_OUTPUT.PUT_LINE(BUFFER);
        EXIT WHEN CV%NOTFOUND;
    END LOOP;
    CLOSE CV;
END;
```

Exploiting GET_OWNER() with only CREATE SESSION

- *UNION SELECT*
- *Inject extant function*
- *Inject a cursor*

Example: get_owner.txt

Real world example

MDSYS.SDO_DROP_BEFORE_USER contains the following SQL:

```
EXECUTE IMMEDIATE
'begin ' ||
'mdsys.rdf_apis_internal.' ||
'notify_drop_user(''' || dictionary_obj_name || '''); ' ||
'end;';
```

Exploiting SDO_DROP_USER_BEFORE

1) Find a table anyone can insert into (e.g. OL$ owned by SYSTEM

2) Will inject into the SDO_DROP_USER_BEFORE to create another trigger on the OL$ table

3) This new trigger will give us DBA privileges

4) Insert into OL$ to fire the trigger

5) Demo – trigger.txt

Possible defences

Revoke execute on DBMS_SQL from PUBLIC… not a good idea; too many dependencies.

Trigger to prevent DML…

Questions and Answers

Any questions?

Thank You

http://www.ngsconsulting.com/