pentest
security assurance

Have your objects been tampered with?

By Pete Finnigan

The Problem

How would you know if an attacker had been into your production database and altered any of your database objects or altered some of your PL/SQL code to place a trojan or even added new objects such as database triggers to capture data? An unknown hacker could have done any of the above things and stole some data or done some damage before you noticed. Even worse it could be one of your employees and he or she didn't even have to break through your firewall to do it!

Checking if an object has changed or been added is quite simple to do. You could just select objects from the data dictionary where the date is today's date each day and note what has changed and investigate why it has changed. It would be simpler to store the date times of all the objects each day and then produce a report of those that had changed.

Of course storing this information within the same database defeats the object if the attacker knows that you are keeping records of changes as he could then very easily change the records themselves. Of course our Wiley hacker if he knew what he was doing could edit the data dictionary in the database he attacks and change the dates as well to cover his tracks.

Therefore we would like to store the records elsewhere ideally in a separate database. Lets look at the data dictionary and see exactly what timestamps change when a procedure is created and when the PL/SQL code is reloaded and also when it's recompiled. Here is some test code to demonstrate

```
SQL> create procedure pete_test as
  2  begin
  3  null;
  4  end;
  5  /

Procedure created.

SQL> select to_char(ctime,'DD-MON-YYYY HH24:MI:SS'),
  2  to_char(mtime,'DD-MON-YYYY HH24:MI:SS'),
  3  to_char(stime,'DD-MON-YYYY HH24:MI:SS')
  4  from obj$
  5  where name='PETE_TEST';

TO_CHAR(CTIME,'DD-MO TO_CHAR(MTIME,'DD-MO TO_CHAR(STIME,'DD-
MO
-------------------- -------------------- --------------------
-
29-NOV-2001 17:37:40 29-NOV-2001 17:37:40 29-NOV-2001
17:37:40
```

```
SQL> create or replace procedure pete_test as
  2  begin
  3  null;
  4  end;
  5  /

Procedure created.

SQL> select to_char(ctime,'DD-MON-YYYY HH24:MI:SS'),
  2  to_char(mtime,'DD-MON-YYYY HH24:MI:SS'),
  3  to_char(stime,'DD-MON-YYYY HH24:MI:SS')
  4  from obj$
  5* where name='PETE_TEST'
SQL> /

TO_CHAR(CTIME,'DD-MO TO_CHAR(MTIME,'DD-MO TO_CHAR(STIME,'DD-
MO
-------------------- -------------------- --------------------
-
29-NOV-2001 17:37:40 29-NOV-2001 17:39:25 29-NOV-2001
17:39:25

SQL> alter procedure pete_test compile;

Procedure altered.

SQL> select to_char(ctime,'DD-MON-YYYY HH24:MI:SS'),
  2  to_char(mtime,'DD-MON-YYYY HH24:MI:SS'),
  3  to_char(stime,'DD-MON-YYYY HH24:MI:SS')
  4  from obj$
  5  where name='PETE_TEST';

TO_CHAR(CTIME,'DD-MO TO_CHAR(MTIME,'DD-MO TO_CHAR(STIME,'DD-
MO
-------------------- -------------------- --------------------
-
29-NOV-2001 17:37:40 29-NOV-2001 17:41:27 29-NOV-2001
17:39:25

SQL>
```

As you can see there are three time fields *ctime*, *mtime* and *stime* in the table `OBJ$`. When a `PL/SQL` object is created all three date / time fields are set. When the `PL/SQL` code is reloaded then *mtime* and *stime* fields change to the current time. Finally when the object is re-compiled in the third case *mtime* changes.

What to monitor

When an object is added we will see a new record of course, when the source is reloaded (i.e. changed by someone) *stime* is amended and *mtime* is changed because the object is also compiled. Finally when a compile takes place *mtime* is changed. Lets summarise:

ctime	Object create time
stime	Object re-create time
mtime	Object compile time

To find where objects have been added then we can simply look for new records in `OBJ$`. We can also look for records that have been deleted which would signify that the object has been deleted. Looking at the compile time can be useful but could be ignored as objects could become invalid for any number of reasons. In this example we will ignore compile time but it could easily be included. To check if someone has altered objects we can monitor *stime*.

An Example

The example I show here is implemented in the same database as we are monitoring. This as I have stated previously is not ideal. The reader can move this code into a sepa rate database and use database links to enable it to run. I also just monitor database procedures in this example again for simplicity. Also the database user that does this work needs to be able to access `SYS.OBJ$` and `SYS.USER$`.

First create a table to store the results in:

```
SQL> create table monitor_obj
  2  (
  3     name    varchar2(30),
  4     owner#  number,
  5     type#   number,
  6     create_date date,
  7     reload_date date,
  8     recompile_date date,
  9     run_date date
 10  );

Table created.

SQL>
```

For a table to be used in production a suitable storage clause should be added and a suitable tablespace used for the table. The user used to create the table should be considered. As this is a just a simple example I have created the table without storage or a specific tablespace and I have created it as the database user `SYS`.

Permission to access the table would then need to be granted explicitly to the user who will access it for adding records and also for reading. In this case we will simply grant access to `PUBLIC` to simplify the example as follows:

```
SQL> grant select,insert,delete,update on monitor_obj to
public;

Grant succeeded.

SQL> create public synonym monitor_obj for sys.monitor_obj;

Synonym created.

SQL>
```

Now that we have a table, we need to populate it with records. This would be run as often as the *dba* or administrator would like. It can be run as a DBMS_JOB, job or run in *cron* or through Oracle Enterprise Manager. We need to collect a snapshot of the system and store it so that we can compare the system with its past the next time it runs and report all of the differences. In this example we will only keep one set of records (the last run) and delete the previous ones before we run the next one. Of course for this to be useful you would keep a set of records that suited your purpose. Here is a simple PL/SQL package procedure to add and remove records from the monitor table.

```
        --
        -- package procedure to create and delete object change
        -- records from the monitor database.
        --
        spool out.lis

        set feed on
        set head on
        set pages 25
        set lines 132

        create or replace package monitor_main as
                procedure del_monitor(rem_date in date:=null);
                procedure add_monitor;
        end monitor_main;
        /

        create or replace package body monitor_main as
                --
                procedure del_monitor(rem_date in date:=null)
                is
                begin
                        if rem_date is null then
                                null;
                        else
                                delete from monitor_obj
                                where   to_date(run_date,'DD-MON-YYYY')
                                        =to_date(rem_date,'DD-MON-
YYYY');
                        end if;
                exception
                        when others then
                                dbms_output.put_line('Error in
monitor_main.del_monitor '||sqlcode);
                                raise_application_error(-
20100,'monitor error');
                end del_monitor;
                --
                procedure add_monitor
                is
                begin
                        insert into monitor_obj
                        select  name,
                                        owner#,
                                        type#,
                                        ctime,
                                        stime,
                                        mtime,
                                        sysdate
```

```
                        from    sys.obj$
                        where   type#=7;
                exception
                        when others then
                                dbms_output.put_line('Error in
monitor_main.add_monitor '||sqlcode);
                                raise_application_error(-
20100,'monitor error');
                end add_monitor;
                --
        end monitor_main;
        /
```

This simple package allows us to create audit records for database PROCEDURE and to remove them. By using this package procedure we can use the two functions to delete a record and add a new one. This is illustrated in SQL*Plus as follows. The delete function could be modified to delete the earliest record but at the moment it's deleting the record it's told to.

```
SQL> select count(*),run_date
  2  from monitor_obj
  3  group by run_date;

 COUNT(*) RUN_DATE
--------- ---------
       13 03-DEC-01
       13 04-DEC-01

2 rows selected.

SQL> exec monitor_main.del_monitor('03-DEC-2001');

PL/SQL procedure successfully completed.

SQL> exec monitor_main.add_monitor;

PL/SQL procedure successfully completed.

SQL> select count(*),run_date
  2  from monitor_obj
  3  group by run_date;

 COUNT(*) RUN_DATE
--------- ---------
       13 04-DEC-01
       13 05-DEC-01

2 rows selected.

SQL>
```

The above example shows that we had two records, one for the 3rd of December and one for the fourth. We then remove the record for the 4th and add a new record for the current day.

The last thing to do is find differences between the previous record and the latest and then report those differences. We will illustrate this with an example session

here. I will first create a set of records in the monitor table and then re-load the PL/SQL for the procedure PETE_TEST and recompile the PROCEDURE PSTUB. I will also add a procedure NEW_PROC. Lets see this first:

```
SQL> exec monitor_main.add_monitor;

PL/SQL procedure successfully completed.

SQL> create or replace procedure pete_test as
  2  begin
  3     dbms_output.put_line('TESTING');
  4  end;
  5  /

Procedure created.

SQL> alter procedure pstub compile;

Procedure altered.

SQL> create procedure new_proc as
  2  begin
  3     null;
  4  end;
  5  /

Procedure created.

SQL> alter system set fixed_date='04-DEC-2001';

System altered.

SQL> exec monitor_main.add_monitor;

PL/SQL procedure successfully completed.

SQL> select count(*),run_date
  2  from monitor_obj
  3  group by run_date;

 COUNT(*) RUN_DATE
--------- ---------
       13 03-DEC-01
       14 04-DEC-01

SQL> commit;

Commit complete.

SQL>
```

Setting up this test case also includes using the variable fixed_date to change the date at which the system thinks its running. This is just to simulate us running this on a different day. See fixed_date for a simple article highlighting the issues with this system parameter.

All that is left to do now is to write a simple report that can be run to find the changes made to the system. Here is the SQL and a sample run to find the changes we made above:

```
        --
        -- check for changes in PL/SQL Procedures
        --

        undefine start_date
        undefine end_date

        spool ch_mon.lis

        doc
                Find procedures that have been added to the database
        #

        select  a.name,c.name
        from    monitor_obj a,
                user$ c
        where   c.user#=a.owner#
        and not exists (select 'x'
                from monitor_obj b
                where to_char(b.run_date,'DD-MON-
YYYY')='&&start_date'
                and a.owner#=b.owner#
                and a.name=b.name)
        and to_char(a.run_date,'DD-MON-YYYY')='&&end_date'
        /

        doc
                Find procedures that have been re-loaded into the
database
        #

        select  a.name,c.name
        from    monitor_obj a,
                monitor_obj b,
                user$ c
        where   c.user#=a.owner#
        and     to_char(b.run_date,'DD-MON-YYYY')='&&start_date'
        and     a.owner#=b.owner#
        and     a.name=b.name
        and     to_char(a.reload_date,'DD-MON-YYYY HH24:MI:SS')
                        <>to_char(b.reload_date,'DD-MON-YYYY
HH24:MI:SS')
        and to_char(a.run_date,'DD-MON-YYYY')='&&end_date'
        /

        doc
                Find procedures that have been recompiled
        #

        select  a.name,c.name
        from    monitor_obj a,
                monitor_obj b,
                user$ c
        where   c.user#=a.owner#
        and     to_char(b.run_date,'DD-MON-YYYY')='&&start_date'
        and     a.owner#=b.owner#
        and     a.name=b.name
        and     to_char(a.recompile_date,'DD-MON-YYYY HH24:MI:SS')
```

```
                        <>to_char(b.recompile_date,'DD-MON-YYYY
HH24:MI:SS')
        and to_char(a.run_date,'DD-MON-YYYY')='&&end_date'
        /

        spool off
```

And here is a sample run to find the changes we made above:

```
        DOC>        Find procedures that have been added to the
database
        DOC>#
        Enter value for start_date: 03-DEC-2001
        old   7:        where to_char(b.run_date,'DD-MON-
YYYY')='&&start_date'
        new   7:        where to_char(b.run_date,'DD-MON-YYYY')='03-
DEC-2001'
        Enter value for end_date: 04-DEC-2001
        old  10: and to_char(a.run_date,'DD-MON-YYYY')='&&end_date'
        new  10: and to_char(a.run_date,'DD-MON-YYYY')='04-DEC-2001'

        NAME                           NAME
        ------------------------------ ------------------------------
        NEW_PROC                       SYS

        DOC>        Find procedures that have been re-loaded into the
database
        DOC>#

        NAME                           NAME
        ------------------------------ ------------------------------
        PETE_TEST                      SYS

        DOC>        Find procedures that have been recompiled
        DOC>#

        NAME                           NAME
        ------------------------------ ------------------------------
        PETE_TEST                      SYS
        PSTUB                          SYS
```

Conclusions

Although this technique is quite simple as is the SQL and PL/SQL shown you can see how easily this could be expanded to cover all database objects and be automated and also to place the results in another database and maybe report on multiple databases at the same time.

Pentest is developing a simple product to expand on these ideas and to add some additional functionality. This will be available from Products very soon.

About Pentest

Established in 2001, Pentest Limited is a leading international provider of IT security, specialising in Web Application Security and Penetration Testing services. Pentest provides independent, practical advice to a wide range of clients across the UK, Europe, USA and Asia. For more information, or for further details about Pentest's services, please visit www.pentest.co.uk or call +44 (0) 161 233 0100.