Secure Coding. 
Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

This paper is from the SANS Software Security site. Reposting is not permitted without express written permission.

Interested in learning more?
Check out the list of upcoming events offering "()"
Protecting Access to Data and Privilege with Oracle Database Vault

A SANS Product Review

Written by Pete Finnigan

January 2015

Sponsored by Oracle
Introduction

One of the biggest problems facing companies is how to avoid the potentially disastrous commercial consequences—and the inevitable media embarrassment—of having customer data stolen and paraded publicly. This particularly includes personal details that could be used to steal someone’s identity, or worse, data such as credit card numbers, PINs and other information.

Oracle has been synonymous with the word security from its inception; among its first customers were three-letter agencies that have always been keen to ensure that data and secrets remain so. That focus remains, and Oracle wants everyone to benefit from extra security features in the database, as expressed in Oracle Database Vault, which is aimed at creating a security cordon around the data and preventing privilege abuse.

In this paper, we take you on a hands-on walk through Oracle Database Vault with Oracle Database 12c by looking at how some of its features can be used to protect real data in real world organizations. This paper is written for database administrators, operations staff, and security or IT managers who are tasked with protecting data.

Oracle Database Vault 12c protects data at the source from privilege abuse by identifying privileged users and then implementing least privilege and separation of duties principles at the data level. Database Vault cannot solve all data security problems, but it can solve some of the essential protections required for keeping data safe from privileged user and privilege escalation attacks. It does so easily and, in some cases, straight out-of-the-box. Policies, as well as actions such as hardening, audit and setting up realms (to protect against indirect privilege grants and abuse) are straightforward, while precreated policies are sensible and also easy to administer. These and other features are covered in the rest of this report.

**Key Features of Database Vault 12c**

Some of the main features of Oracle Database Vault 12c include:

- **Mandatory realms.** A “realm” can be used to seal off access to application objects and lock down privileges granted to roles. It can also be used as an additional gate check prior to allowing access by privileged as well as traditional users, including the object owner.

- **Runtime analysis of roles and privileges.** This allows examination of privileges used at runtime to help identify the least number of privileges the application needs to run, thereby reducing the attack surface, making the application more secure.

- **Easy enablement of protections.** This removes a barrier to implementation on production systems.

- **Precreated policies for enterprise applications.** This enables easy, out-of-the-box protection for applications such as Oracle E-Business Suite, Oracle PeopleSoft or SAP.

- **Support for the new Oracle Multitenant option of Oracle Database 12c.** Database Vault can be targeted at each pluggable database and does not need to be used across the whole container.

- **Hardening of the database configuration.** This is available for Database Vault. By default, some basic database hardening is done to limit configuration settings and privileges.
Defining the Problem

Securing data in databases is multifaceted effort. Oracle Database, the web servers, their applications and the design work needed to securely use data are all complex entities. The security of the data can be compartmentalized in roughly the fashion shown in Figure 1.

![Figure 1. Areas of Security Within a Database](image)

Applying security patches isn’t trivial and can be costly: Patching can involve regression testing of applications and systems. Although important, patching is often delayed or not completed due to these complexities.

Hardening of systems involves locking default accounts, applying controls to system resources and, of course, reducing privilege. These steps should be part of any system deployment, especially those involving production systems. Finally, security for “the rest” should be included in the design from the beginning of development, but too often, organizations focus on SLAs and meeting functional requirements at the expense of few or no security protections for data in the database.

As an example, we used a poorly designed database application as a workhorse for our evaluation of Database Vault 12c. In this application, the schema account lacked any separation of duties for web-based end users or back-office staff. As another example of failed separation of duties in our application, we had internal business data stored with external marketing data; this presented a data-level failure to separate duties appropriately. Database Vault can help with such issues by providing a cordon around the data itself, but in the long run, a design overhaul may be needed to keep Database Vault from becoming mere duct tape over a bad design. Good design characteristics generally include:

- Blocking access from all but designated IP addresses
- Giving end users the bare minimum of privileges necessary, thereby limiting the possibility for abuse of the data via the application
- Using logon triggers or network security to limit access to application tables, even by privileged users
Database Vault also addresses rules to limit privilege—another issue where database administrators are often handed the “keys to the kingdom” even though their duties require them to access only specific areas. Database Vault addresses these two problems by limiting sweeping access for groups of users and data, whether out-of-the-box or with minimal further effort. Additionally, the Privilege Analysis feature enables organizations to analyze current privilege settings, with the aim of reducing these to the least privilege needed. This is not only applicable to database administrators with specific duties, but also to the applications themselves. Many applications run with DBA-like privileges, creating a potential stepping-stone to other parts of the database for hackers who attack the application schema.

Main Threats to Data

Many threats to data security result from violations of two best practices: least privilege and separation of duties.

Too Much Privilege

Each user of the database should have one account with enough privileges to complete authorized tasks, following the “least privilege” model. Violations of this principle can occur throughout an application, from data access for end users to ownership of data and functionality, to practices of direct support staff such as database administrators.

Failure to Separate Duties

Each actor having direct access to data should have a user account and privileges designed for his or her job role, but in real life this is often not the case. To access databases such as our example, developers often use commonly available accounts where passwords are known to them, instead of their own individual accounts and passwords. These borrowed accounts often have excessive privileges granted for nonstandard purposes such as tuning and, worse, employ passwords that are known to a group of people. Instead, accounts should be unique to each business need, and passwords should not be shared. (The DBA role and the omnipotent-by-default SYS and SYSTEM accounts are particularly dangerous and often unsupervised.)
Database Vault Components

Database Vault protections include several main components, which we will see in action later in this walkthrough. Table 1 offers a short description of each component.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factors</td>
<td>Context used in Database Vault rules; e.g., the client’s IP address or the program used to access the database.</td>
</tr>
<tr>
<td>Rules</td>
<td>Determine whether something should be allowed. An example could be translated as: “Is the database connection using the schema account, the SQL*Plus utility or [something else]?” Rules can use built-in or user-defined factors.</td>
</tr>
<tr>
<td>Rule Sets</td>
<td>Groups of rules that determine whether an action can proceed. A rule set is a container for multiple rules, and provides a pass/fail evaluation each time it executes.</td>
</tr>
<tr>
<td>Command Rules</td>
<td>Control commands such as CONNECT or CREATE TABLE. A command rule uses a rule set to determine whether the command can be used.</td>
</tr>
<tr>
<td>Realms</td>
<td>Protection zones around data that restrict access by privileged users while maintaining the ability to perform authorized administrative tasks. A realm can be created around a complete schema, a single table or a package; users can then be authorized to the realm. Realms can use a rule set to determine access. A “regular” realm allows users with object privileges to access the realm without explicit authorization; mandatory realms block access based on system privileges as well as access based on object privileges, even from the owner of the data.</td>
</tr>
</tbody>
</table>
Examples of Insecure Code

The testbed used in this walkthrough centers on a web-based application called “Orablog” that used a pluggable database to store the application schema for our test data. The application and web servers were connected to the database as a shared user, so external end users were essentially unauthenticated. More detail on setup can be found in Appendix A.

Following are some simple examples of how our application and data were insecure by default. Let’s start with an obvious one: Our database administrator could use the CVJMUJOVTFS4:45&.	BVTFSXJUIUIFQPXFSGVM0SBDMFSPMFPGi%#” database, create or alter user passwords, including the one for the application schema BDDPVOU	VTFSOBNFi03#-0(w
8FTFUJUVQTPBENJOJTUSBUPSTDPVMEWJFXFWFO decrypted credit card details stored in the database.

Our problem was worse because the default permissions on the credit card table and DSZQUPQBDLBHFXFSFBWBJMBCMFUPBMMVTFST	UIFi16#-*$wHSPVQ
5IJTNFBOTBVTFSXJUI just CREATE SESSION privileges could also read and decrypt the credit card details. This is shown in Figure 2, where a user can see information in a decrypted state.

As you can see, there is a lot wrong with the basic security of this database, but it gets worse. The database supports both public data (the website) and also business data, including credit card details that are supposed to be processed only by back-office staff.
Implementing Database Vault

Oracle Database Vault is installed by default with Oracle Database 12c. In this system, enabling Database Vault is as simple as running a few commands in SQL*Plus.

If you already use Oracle Enterprise Manager (OEM) in your organization, a second element adds to the lower cost of ownership for Database Vault: The web-based management interface for Database Vault in OEM that previously shipped as a separate web application and web server is now fully integrated into Cloud Control, Release 3.

In Oracle Database 12c and earlier, you can configure and use Database Vault from the SQL command line via the PL/SQL API; however, it is sometimes convenient to use the web interface, as we did for the part of our review shown later under Example 3. In most cases, we used the PL/SQL interface, as it is a typical user interface for seasoned Oracle hands. It is a convenient and fast way to create and test command rules, conditional (true/false) rules, factors and realms, and then apply them to multiple databases.

Oracle Database 12c includes the option to install the database in legacy mode (one database in one database container) or in multitenant mode (multiple databases in one container). Database Vault must be enabled in the root container (the holder for all pluggable databases) before it can be enabled in each/any pluggable database.

Oracle Label Security

As our application was installed in the pluggable database PDBORCL, we needed to first establish whether Database Vault and Oracle Label Security—a separate product that enables security labels to be added to rows of data, which in turn enables access to be controlled at the data level—are enabled. First, we ran a canned PL/SQL procedure in Figure 3 to configure and enable label security.

```
SQL> connect sys/oracle10/@192.168.1.87:1521/pdborcl.localdomain as sysdba
Connected.
SQL> exec lbacsys.configure_ols;
PL/SQL procedure successfully completed.
SQL> exec lbacsys.ols_enforcement.enable_ols;
PL/SQL procedure successfully completed.
SQL>
```

*Figure 3. Enabling Label Security*

Next, we had to create two accounts, one to act as the Database Vault owner and another to manage database accounts, as Database Vault 12c’s protection blocks heretofore-powerful roles such as DBA and all-powerful users such as SYS and SYSTEM from managing database accounts. Instead, we wanted to have the users DBV_OWNER and DBV_ACCTMGR handles, respectively, the ownership and management roles.
Our next step was to configure Database Vault in the pluggable container, as shown in Figure 4.

```sql
begin
    dvsys.configure_dv(       
        owner_uname => 'dbv_owner',
        acctmgr_uname => 'dbv_acctmgr');
end;
/

PL/SQL procedure successfully completed.

SQL>
```

*Figure 4. Configuring Database Vault in a Pluggable Container*

We then had to recompile all objects in the pluggable database, using the `utlrp.sql` script. Once this was done, we connected as the user DBV_OWNER to enable Database Vault, as shown in Figure 5.

```sql
SQL> connect dbv_owner/dbv_owner@//192.168.1.87:1521/pdborcl.localdomain
Connected.
SQL> exec dbms_macadm.enable_dv;

PL/SQL procedure successfully completed.

SQL>
```

*Figure 5. Enabling Database Vault*

Enabling Database Vault in our testbed’s database was effortless. The fact that Database Vault is now preinstalled—requiring only a few simple steps to enable it—is a big improvement from Oracle Database 11g, where a separate software installation had been required.

---

1 Oracle provides this script for use when recompiling invalid database objects; it is typically used following a patch or upgrade.
Enhancing Security Effortlessly

We already gained some security simply by enabling Database Vault 12c, taking advantage of the basic database hardening features and Database Vault’s built-in policies and realms. Most notably, we could now no longer create a new user in this database or change a user’s password with the DBA role or with standard administrative accounts such as SYS.

If we had started with a properly configured Orablog application (so that access to the CREDIT CARDS table had not been granted to all database users, but instead was available on a need-to-know basis rather than via sweeping privileges such as SELECT ANY TABLE), Oracle Database Vault would have also protected our key data without further configuration. However, our test application reflects most real-world applications, where privilege design and access to data have often not been properly controlled.

Hardening Databases

By default, when Oracle Database Vault is configured and enabled, a set of database hardening steps takes place in the background that serves two purposes: First, the installation routine enables common parameters recommended by Oracle in its own database hardening guides and by other entities such as the Defense Information Systems Agency (DISA), the National Institute of Standards and Technology (NIST) or the Center for Internet Security (CIS). Second, the routine also sets basic parameters such as auditing for SYSDBA privileges.

These steps also prevent users from manipulating other users, programs, jobs or classes, while revoking some privileges from roles such as DBA, EXECUTE_CATALOG_ROLE and SCHEDULER_ADMIN. Privileges such as ALTER USER and CREATE USER are blocked from all users (including SYS) by default. Once this hardening takes effect, security administrators have to use the user management account DBV_ACCTMGR (created during the enabling of Database Vault) for such tasks instead of the SYS account, thereby ensuring some degree of separation of duties.

There are, of course, complexities with hardening. Some areas to watch out for include:

- If Database Vault is enabled first in the root container, the revoked privileges need to be added back manually to pluggable databases if Database Vault is not specifically enabled in them.
- If the Real Application Clusters option is involved and each cluster node has a separate physical home, the hardening may need to be done manually on cluster nodes.
- If controls are employed across a company to prevent users from changing their own passwords, changing users’ passwords could increase a provisioning team’s workload.
Analyzing Privilege

Our testbed included a content management system (CMS) to serve up the company’s website and, as such, presented a simpler version of most commonly used Oracle-backed applications. Our testbed was insecure by default and allowed external end users—that is, from the website—to connect as the schema owner ORABLOG without authenticating. These users, by virtue of connecting as the schema owner, owned all of the data and therefore needed no specific privileges to access its own data. As we see later in this section, ORABLOG also has some direct system privileges and Oracle roles.

It makes sense in a real production system to analyze the use of privileges before creating realms and rules for Database Vault. This analysis most logically should inform the Database Vault deployment rather than the other way around. Analyzing privileges before Database Vault is enabled also provides an understanding of current privileges, including those that need to be updated or removed.

Analyzing Privilege in Oracle Database

The basic steps to using Database Vault’s Privilege Analysis feature are:

1. Create a policy specifying the required settings for analyzing privileges.
2. Enable the policy in the database.
3. Test the policy by doing some work in the database that will use privileges and roles.
4. Disable the policy in the database to stop gathering data.
5. Create a report that stores results to the Oracle Database data dictionary.
6. Review the results either using the Privilege Analysis screens in Oracle Enterprise Manager Cloud Control or via SQL*Plus and dictionary views.
7. Delete the report and results by deleting the policy.

We examined what privileges the schema owner ORABLOG needs to complete tasks.
Analyzing Runtime Privileges

We could then connect as the user DBV_OWNER—a Database Vault administrator with the CAPTURE_ADMIN role needed for analysis—and begin analyzing privileges. We wished to analyze only the ORABLOG account, so we created a policy to capture privileges for this user only and then enabled it, as shown in Figure 6.

```sql
SQL> begin
2  sys.dbms_privilege_capture.create_capture(
3         name => 'ORABLOG Privilege Analysis',
4         description => 'Analysis of the ORABLOG schema privilege use',
5         type => dbms_privilege_capture.G_CONTEXT,
6         condition => 'SYS_CONTEXT("USERENV","SESSION_USER")="ORABLOG"');
7  end;
8 /
PL/SQL procedure successfully completed.

SQL> exec dbms_privilege_capture.enable_capture('ORABLOG Privilege Analysis');
PL/SQL procedure successfully completed.
```

Figure 6. Enabling Privilege Capture Policy

After enabling the policy, we simulated some work in our application by visiting all of the website screens and running various reports that access the back-office data, in particular, the credit card details. (In real life you may wish to leave the policy enabled for some time to obtain meaningful results.) In our case we simply accessed all facilities, and then disabled the policy and generated the report’s data, as shown in Figure 7.

```sql
SQL> connect dbv_owner/dbv_owner@//192.168.1.87:1521/pdborcl.localdomain
Connected.

SQL> exec dbms_privilege_capture.disable_capture('ORABLOG Privilege Analysis');
PL/SQL procedure successfully completed.

SQL> exec dbms_privilege_capture.generate_result('ORABLOG Privilege Analysis');
PL/SQL procedure successfully completed.
```

Figure 7. Generating Policy Data Report
Enhancing Security Effortlessly (CONTINUED)

We could then see the results of the analysis. The two simplest reports available to us reviewed the privileges actually used and, more importantly, listed those privileges not used: DBA_USED_PRIVS and DBAUNUSED_PRIVS, respectively. Figure 8 shows the report of unused database privileges for ORABLOG.

![Figure 8. Unused Privileges of User ORABLOG](image)

This report did not include all of the PUBLIC group’s privileges, which are granted to all users; instead, such a check should focus on roles and direct system privileges first, then tackle PUBLIC’s privileges afterward.

### Analyzing Use Modes

Analyzing privileges to monitor runtime use only is not enough. For a thorough investigation, all “use modes” of a user or schema should be examined. Too often, privileges granted to a schema are used only during installation or for compilation of the functional code. Code using so-called “definer’s rights” needs those privileges only during compilation.

Simply recompiling our code would not test the intended use case of installation with privilege analysis, and we would have to drop and recreate the PL/SQL procedure to test its use of those privileges. In the case of tables, it is usually inconvenient to remove a production table and recreate it, especially if it holds very large amounts of data. Also, consider that recreating a procedure means all of the existing GRANT statements for the procedure are also dropped and must be recreated as well. Analysts must therefore carefully plan their testing with Privilege Analysis.

---

2 Oracle has two modes of privilege resolution for PL/SQL code: “Definer’s rights” code executes as the owner of the code, and the rights are analyzed when that code is compiled. “Invoker’s rights” code executes as the invoker of the PL/SQL, and the rights are analyzed at runtime rather than when compiled. Definer’s rights may need significantly fewer privilege grants to invokers of the code, but are more dangerous, as the rights are “baked into” the code and an invoker could easily exploit them.
Establishing Least Privilege

To set appropriate privilege, we also needed to analyze what privileges were not in use, in order to remove them. Figure 9 shows privileges we could potentially remove to approach “least privilege.”

![Least Privileges that Should Be Removed](image)

It is imperative to analyze all modes of operation to determine the privileges available for every user of the database, so their privileges may be reduced accordingly and/or the results used to design Database Vault rules as needed. Comparing the results shown in Figure 8 and Figure 9 reveals four privileges that are used during installation but not during execution. In the design of our application, we would do well to:

- Remove those privileges needed only for installation.
- Remove the reliance on the roles RESOURCE and CONNECT and design our own roles, reduced to the privileges necessary for the application schema.
- Remove the UNLIMITED TABLESPACE privilege and instead grant quotas on the ORABLOG_TABLESPACE privilege.
- Review all direct access accounts used—such as admins or support staff—and analyze their runtime privileges, or whether they are used at all.

With minimal effort on our part, Oracle Database Vault successfully hardened the database’s configuration and analyzed the runtime privileges of our application’s schema owner, giving us a starting point for remediation efforts.
In this section we are going to use Database Vault to work around some of the bad-design issues in our test application, making it more secure.

**Example 1: Limiting Command Access**

With Database Vault, we can limit what a user can do using command rules and rule sets to control the following:

- Tools or programs used
- Times that users can work
- Privileges a user can have
- User connections

We used a combination of factors, rules, rule sets and command rules to make sure the ORABLOG database account cannot be used with SQL*Plus from the application server (Server 1 in our testbed diagram [Figure A-1] on page 22). This would provide protection in case an attacker is able to access that server’s Unix command line and attack the database through that avenue. Figure 10 shows a logical view of the relationships among these elements.

![Figure 10. Relationships of the Orablog Application's Factors, Rules, Rule Sets and Command Rules](image-url)
Our first step was to create a new factor that allowed us to capture the client program used from the web server. We then created a rule set with two rules: one for the user ORABLOG, banning it from using SQL*Plus for database access, and a second allowing any other user to do so. We used the new client program factor in the first rule, shown in line 5 of Figure 11, then linked the rule set to the command rule governing the CONNECT command.

```
SQL> get cr_rules.sql
1  -- create two rules
2  BEGIN
3  DBMS_MACADM.CREATE_RULE(
4      rule_name => 'Prevent ORABLOG Schemas Access to SQL*Plus',
5      rule_expr => 'UPPER (DVF.F4C_F_I) != ''SQLPLUS@EEL59ORABLOG.LOCALDOMAIN(TNS V1-V3)'' AND DVF.F$SESSION_USER IN (''ORABLOG''))
6  END;
7  /
8  BEGIN
9  DBMS_MACADM.CREATE_RULE(
10     rule_name => 'Allow Non-ORABLOG Access to SQL*Plus',
11     rule_expr => 'DVF.F$SESSION_USER NOT IN (''ORABLOG''))
12  END;
SQL> #cr_rules.sql
```

**Figure 11. Rules Restricting Database Access**

The ORABLOG account was locked out of SQL*Plus on this web server, as shown in Figure 12, but it could still be used from other machines in the organization, which was our goal.

```
[oracle@eel59orablog ~]$ sqlplus orablog//192.168.1.87:1521/pdborcl.localdomain
SQL*Plus: Release 12.1.0.1.0 Production on Tue Jan 28 12:22:15 2014
Copyright (c) 1982, 2013, Oracle. All rights reserved.
ERROR:
ORA-47306: 20401: SQL*Plus not allowed for ORABLOG
```

**Figure 12. Error Message After Attempted Access**
Example 2: Securing Access

We extended Example 1 and further secured the ORABLOG account by editing one of the rules to ensure that this account could connect to the database from only two devices: the application server (in this case, IP address 192.168.1.89) or from a specified administrator’s PC (192.168.1.50). Figure 13 shows the new rule.

```
BEGIN
  DBMS_MACADM.CREATE_RULE(
    rule_name => 'Prevent ORABLOG Schemas Access to SQL*Plus',
    rule_expr => 'UPPER (DVF.FSCLIENT_IP) LIKE '+
          '''SQLPLUS@ELS90ORABLOG.LOCALDOMAIN (TNS V1-V3)'' AND UPPER (DVF.FSCLIENT_IP) in ('''192.168.1.89''','''192.168.1.50'') AND DVF.FSESSION_USER IN ('''ORABLOG''));
  END;
/
```

Figure 13. Restricting Access Based on IP Address

This added to the rule the internal factor DVF.F$CLIENT_IP, which tested any connection by the schema owner ORABLOG to verify that it was coming from either the application server and was not using SQL*Plus, or was coming from our admin PC. When we tried to connect as ORABLOG from that PC, we could do so, but we received an error message if we tried to use SQL*Plus from the application server with that account.

We could also change the rules in Database Vault to force the program to whitelist our application server and admin PC instead of blacklisting SQL*Plus, as the latter’s identity (from the perspective of Oracle Database) is machine-dependent, making it more of a challenge to block in a real-world environment.

Example 3: Blocking Access to Credit Card Details

Remember that our test data also contained some credit card details; compliance regimes such as PCI DSS require such data to be protected by encryption and other functionality and data to be accessed only by back-end applications or back-end staff. In practice, support staff or the web application’s supplier are sometimes allowed to connect to a database as its schema owner for maintenance or troubleshooting, putting them in a position where they could access or decrypt credit card details. The mandatory realm feature of Database Vault 12c can instead be used to protect back-end data from support user abuse.
As we saw in the discussion of Database Vault components, a realm is a protective zone around data or functionality (or a complete schema). We wanted to block access to the CREDIT_CARD table and the cryptography used to decrypt credit card numbers with a new realm. By making it a mandatory realm, we would also protect against access by the ORABLOG account, the owner of the data. In its place, we would allow a back-end database user named CCADMIN to be the only user able to select data from the CREDIT_CARD table and use the cryptography module.

We used the web interface in Oracle Enterprise Manager Cloud Control to create a new mandatory realm on the objects ORABLOG.CREDIT_CARD and ORABLOG.ORABLOG_CRYPTO. We also configured Database Vault to block every user from the realm, including those with direct rights; even the schema owner ORABLOG would be blocked. Finally, we authorized the user CCADMIN to the realm so that back-end processing could take place. Figure 14 shows the realm’s final configuration summary in Cloud Control.

Figure 14. Creating a Realm to Protect Credit Card Details in ORABLOG Schema
After the realm was activated, the website still functioned correctly, and CCADMIN could still access data.

The most important test was to see whether the ORABLOG user, when connected from an allowed admin PC (as defined in Example 2), was blocked from accessing the CREDIT_CARD table. As ORABLOG, we connected to the database and attempted to read and decrypt credit card details, but failed, as seen in Figure 15.

```
SQL> select name_on_card,orablog.orablog_crypto.decrypt(pan)
2    from orablog.credit_card;
from orablog.credit_card
*     
ERROR at line 2:
ORA-01031: insufficient privileges
```

Figure 15. Connection Failure Caused by Access Restrictions

In summary, we found it simple to use Database Vault’s features to improve the security of a database. Limiting access to commands, restricting database access to designated hosts and protecting data itself with a Database Vault realm were done without requiring any developer involvement.
As we go about making changes to database access and privileges, it is critical that these changes are detected and reported. Here again, separation of duties comes into play. It is generally more convenient to use an external solution to monitor Database Vault and the core database, as well as other products such as Oracle Audit Vault and Database Firewall.

The Database Vault installation is built on top of a number of database roles; access to reports is limited to the roles DV_OWNER, DV_ADMIN or DV_SECANALYST. Any user with one of these roles can access Database Vault’s reporting tools via Oracle Enterprise Manager Cloud Control.

A number of built-in reports exist, including:

- Top five attempted violations of realm or command rule policies
- Top five attempted violations by users or client host
- Time-based trend reports

Two classes of reports are available: Database Vault reports and general security reports. Database Vault also generates audit records by default for more than 200 settings, and auditing can be enabled when creating one’s own command rules and realms.

To get a feel for the reporting capability, we accessed the report interface for the realm and saw that the SELECT statement by ORABLOG in Figure 15 is now a violation, as seen in Figure 16.
The report above was available as a dashboard alert, although Database Vault is not a monitoring solution and its primary function is not to act as an alerting tool. Nevertheless, a security administrator could use the admin screens to easily view Database Vault violations, as we have seen.
Although Oracle’s embedding of strong security in the database reaches back more than 15 years to the release of Oracle 8i and its Virtual Private Database (VPD) feature, Oracle Database Vault 12c reaches a new level by enforcing operational security inside the database. The most powerful new feature in Database Vault 12c—simply because we can enable it with a single checkbox—is mandatory realms, which allow one to protect against abuse of access privileges, even when the owner of the object is involved. The second most powerful feature is the new privilege analysis for finding privileged roles and applications at runtime.

This is a great product that provides security out of the box by preventing abuse by database administrators and other powerful users, as well as criminals who may steal their privileged credentials. Deploying and using Database Vault is quite simple, as we have seen, but it provides powerful tools for limiting privilege, preventing administrative abuse and simplifying complex tasks such as limiting a user to a single web server connection or limiting the tools employed by that user. Finally, we can easily protect a badly designed database and application with Database Vault until it can be properly reworked to a secure design.
Our testbed is simple, but also realistic, representing the same architecture commonly found by the author when auditing systems. The environment included a web-based application called “Orablog” that used an Oracle Database 12c R1 pluggable database—suited for the multitenant deployment mode of Oracle Database—to store the application schema for our mocked-up business data; the application schema supported back-office operations as well as the front-of-business website. Because the application and web servers connected to the database as a shared user, external end users were essentially unauthenticated—which is an invitation to attackers, as we saw in our three examples.

In this model, end users would access the Orablog application via a Web browser. Internal users—including database administrators, application developers, power users and staff who process critical business data—might use tools such as SQL Developer or SQL*Plus, or other applications that access the database.

We used four servers in our testbed, all with Oracle Linux as the OS:

- A database server with Oracle Database 12c and its Multitenant option, hosting the data for both the Orablog application and Database Vault itself in a pluggable database
- A server hosting the Orablog application; this consisted of an Apache webserver and a custom, PHP-based application server using the Oracle Call Interface (OCI) API set
- An installation of Oracle Enterprise Manager Cloud Control, Release 3, which is the GUI for administering Database Vault; this consists of its own web server and Oracle application server
- A secondary database server running Oracle Database 11g Enterprise Edition, with the repository for Cloud Control

In Figure A-1, we see the four hosts running the various components of our testbed; the most important pieces for our discussion are Server 1—hosting the Orablog application itself—and Server 2, hosting the database with the application data.
Many different “actors” access the Oracle databases, applications, web servers and application servers in various modes and with various job descriptions. Each group of actors represents, at some level, a risk to the data and, by extension, our “Orablog Ltd.”
Pete Finnigan is an expert in helping customers big and small secure data held in their Oracle databases. Pete has worked tirelessly for more than 12 years developing methods, free tools and publishing his research and knowledge on Oracle security. Pete was the author of the very popular and respected SANS guide, *Oracle Security Step by Step*, which was used as the basis for the Center for Internet Security’s Oracle benchmark. He runs his own Oracle security consulting company, PeteFinnigan.com Ltd., specializing in all areas of securing data in Oracle databases, including security auditing, audit trail designs and detailed training courses, as well as offering related products. Pete is a popular speaker on the subject of Oracle security around the world at both Oracle user conferences and pure security conferences, including the UK Oracle User Group and Black Hat.

*SANS would like to thank this paper’s sponsor:*
Upcoming SANS App Sec Training