ORACLE

Securing Oracle Database 12*c* A Technical Primer

ORACLE DATABASE 12*c*

Michelle Malcher
Paul Needham
Scott Rotondo

Foreword by Thomas Kyte

Oracle Press

Security Oracle Database 12*c*: A Technical Primer

ISBN 978-0-07-182617-4
MHID 0-07-182617-3

Sponsoring Editor
Paul Carlstroem

Editorial Supervisor
Patty Mon

Acquisitions Coordinator
Amanda Russell

Copy Editor
Margaret Berson

Proofreader
Paul Tyler

Production Supervisor
Jean Bodeaux

Art Director, Cover
Jeff Weeks

About the Authors

Michelle Malcher IOUG Board of Directors President and DBA Team Lead, DRW Holdings

Michelle is an Oracle ACE Director and the DBA Team Lead at DRW Holdings in Chicago, with several years' experience in database development, design, and administration. She has expertise in security, performance tuning, data modeling, and database architecture of very large database environments. When she is not securing her own company's databases, she writes articles and gives presentations on security and compliance topics as well as other database administrative areas such as RAC, ASM, and recovery. She is also a contributing author for multiple books including the *IOUG Best Practices Tip Booklet*. She has been very involved in the IOUG, and is currently serving as president on the Board of Directors.

Paul Needham Senior Director of Product Management for Database Security, Oracle

Paul is responsible for the development of Oracle Database security features and products spanning Oracle Advanced Security, Oracle Database Vault, Oracle Audit Vault and Database Firewall, and Oracle Label Security. Joining Oracle Consulting in 1991, Paul worked closely with customers to help identify their security needs and challenges, and build innovative solutions. In 1998, he joined the Oracle Database Security product management team, and has since then introduced many new database security features and products. Prior to Oracle, Paul was responsible for various government projects at BDM International, a multinational information technology company, and was an intern at the National Security Agency studying database security. Paul graduated from Purdue University with a Bachelor of Science degree in computer science.

Scott Rotondo Consulting Member of Technical Staff for Database Security, Oracle

Scott is a software architect for the Oracle Database Security development group. With 25 years of experience in the computer industry, Scott has held senior technical and management positions in operating system and database development, primarily focused on security features. From 2008 through 2010, Scott served as president of the Trusted Computing Group, an industry consortium and standards body dedicated to enhancing system security using mass-produced, standardized security hardware.

Foreword

In the first half of the 1990s, few if any databases were connected to the Internet, certainly not the Internet as we know it today. The primary consumers of our security technologies back then were government and defense organizations concerned about data classification and multilevel security. Compliance regulations were few and far between. There was no such thing as the Payment Card Industry Data Security Standards (PCI-DSS), and HIPAA was just beginning to take shape. If a DBA wanted to steal a large amount of information (back then, a few megabytes was a "large" amount), they would need to carry out a disk drive larger than a few construction bricks and almost as heavy.

Outside of Oracle's multilevel security solutions, the set of database security features used in our larger customer base consisted of discretionary access controls and database roles. Threats to data from insiders, organized crime, hackers, and SQL injection were not even a consideration. The primary security requirement was enforcing the principle of need-to-know using privileges, roles, and views.

When I joined Oracle in 1993, the user base of any database was necessarily limited—it was the day of client-server computing and, given the machines that databases ran on, databases were rather small; the number of users who had access to that database was small. For most customers database security was barely a consideration. SQL injection was considered acceptable in some applications. In fact, I remember demonstrating the use of SQL injection in Oracle Forms to show how easy it was to modify a query. Of course, due to the client-server nature of most applications, this was not a big deal as the application user was connected to the database using their own personal credentials and not the One-Big-Application User Model used by applications today.

Today, the world is a much different place. A DBA can walk out of work with terabytes of information easily—in a form factor smaller than a mobile phone and weighing even less. Privileged accounts and SQL injection are by some accounts the number one method of accessing sensitive information. Privacy and compliance requirements permeate nearly every industry worldwide. The Sarbanes-Oxley Act, the EU Data Protection Directive, China's Guide for Personal Information Protection, and Japan's APPI are just a few of the many regulations that customers must deal with on a regular basis. Today, the three-tier software architecture is the standard and the user base that has some level of access to application data is huge, typically measuring in the thousands, if not more. Securing your data and infrastructure can no longer be an afterthought, unless you want to appear on the front page of the news for having your users' personal information stolen. Security is something that must be considered as important as high availability and scalability. When designing new applications, security must be considered from the very beginning and through every stage of the lifecycle development process.

That is where *Securing Oracle Database 12*c*: A Technical Primer* comes in. The three authors have dozens of years of experience between them and more importantly—dozens of years of *Oracle Database security* experience.

They begin with the basics—how to control access for your authorized users. This includes the concepts of enforcing "least privilege," for example, making it harder for your trusted DBA or someone with unauthorized access to physically or figuratively walk out with terabytes of sensitive information. Each user in your database, each application schema, should have the smallest set of grants and privileges necessary to accomplish the job—and here we see how to accomplish that and learn why it is paramount.

After having discussed access control fully, they discuss how to secure data in the event of theft; how to secure data even if all access control is subverted. They introduce the concept of data encryption and discuss how to implement it at multiple levels.

Next, they return to access controls, getting into more sophisticated approaches such as column- and row-level access controls. This takes us beyond simple table and system privileges. They then move to a related security feature: auditing. Here we discover the auditing capabilities of the database and how to use them proactively, rather than as an after-the-fact diagnostic tool.

SQL injection is next. SQL injection is perhaps the most ubiquitous attack approach to databases. SQL injection exploits flaws in developed application code—not in the database itself. Given that there is a lot more application code out there floating around than database code, and given that much of that application code was written without thinking about SQL injection issues, there are many vulnerable applications. A quick search on the Internet for "SQL injection" will return millions of hits, many to news articles describing the latest company attacked, breaching sensitive customer data as a result. This book describes how database and security professionals can add a layer of defense between the application and database in order to reduce the ability of an outsider to launch a successful SQL injection attack.

Lastly, the authors look at implementing compliance. In 1993, if a company had 10 or even 100 Oracle databases running, that was considered "a lot." Twenty years later, having thousands, or even tens of thousands, of Oracle databases running is commonplace. The need to ensure that all of these databases have the current patch and have been rolled out using a secure configuration is paramount. Doing that assurance check can be cumbersome, if not impossible, unless you have the right approach and tools, of course; that is what this section is all about. You'll see how to verify and validate your system configuration compliance so that your database environment is set up in a secure, validated fashion.

If you are interested in security—specifically, securing an Oracle database—this is the book for you. While the title references Oracle Database 12*c*, most all of the content is applicable to Oracle Database 9*i* and above.

Thomas Kyte
http://asktom.oracle.com/

Acknowledgments

The authors wish to thank the following individuals who provided invaluable assistance during the preparation and review of this manuscript: Troy Kitch, Melody Liu, Vikram Pesati, and James Spooner.

Introduction

The problem of securing important information has unfortunately become a familiar one to organizations everywhere. A constant stream of news reports tells of successful attacks that gain access to sensitive data and the legal, economic, and reputational damage that results. Even though the vast majority of sensitive data is stored in relational databases, very little of the information security effort in most organizations is devoted to making those databases secure.

While there are many technologies and products available to improve the security of a database in various ways, what is needed is a brief but comprehensive overview that describes the major threats and appropriate techniques to address them. Attackers can be expected to exploit any available weakness including incorrect configuration of security controls in the database, unpatched operating system vulnerabilities, or compromised user accounts. More indirect methods such as SQL injection or intercepting data on the network are also possible. Truly securing a database system requires consideration of any opening an attacker might use.

Each chapter in this book covers a single threat area, but they are all related. There is no single solution that prevents all methods of attack, and each security mechanism reinforces the others. Defense-in-depth is the only way to effectively combat both threats that are known today and those that will be discovered tomorrow.

We begin with security features available within the database itself.

- **Chapter 1: Controlling Data Access and Restricting Privileged Users** describes the fundamental notions of authenticating users and controlling the data that they can access. It covers best practices for determining the access that each user requires and limiting the powers of highly privileged users.
- **Chapter 2: Preventing Direct Access to Data** explains the use of encryption to prevent attacks that attempt to gain access to data directly, bypassing the access controls described in the previous chapter.
- **Chapter 3: Advanced Access Control** covers more sophisticated access control mechanisms that allow for more precise control. These mechanisms include Virtual Private Database, Oracle Label Security, and Real Application Security.
- **Chapter 4: Auditing Database Activity** describes the techniques for maintaining an effective audit trail, which is a vital defense-in-depth technique to detect misuse by privileged users and unexpected violations of the security policies implemented in the previous chapters.

We then broaden the discussion to include external components that improve the security of the database and the data it stores.

- **Chapter 5: Controlling SQL Input** explains the use of a specialized database firewall to monitor the SQL statements going to the database. This helps to protect the database against SQL injection attacks launched by Web users.

- **Chapter 6: Masking Sensitive Data** covers the use of data masking to remove sensitive information from data that is used for test or development purposes. It also describes the use of Data Redaction to dynamically mask the results of queries on production databases.
- **Chapter 7: Validating Configuration Compliance** describes the need to evaluate the database configuration against accepted standards and the tools available for performing the evaluation to ensure continued compliance.

Throughout the book, we highlight new features found in Oracle Database 12*c*. However, the majority of the solutions described in this book are applicable to earlier Oracle Database releases as well.

CHAPTER 1

Controlling Data Access and Restricting Privileged Users

The most fundamental step in securing a database system is determining who should be able to access which data. This chapter describes the management of user accounts and the mechanisms for determining the access that each user has. It continues with a discussion of the types of privileged access that a user may have and available tools for removing any additional access they do not need.

User Management

All access to the database is through users, whether these are administrative users, application accounts, or regular users. As the users have direct connection to the database, it is important that they are properly authenticated and have appropriate roles, and that their accounts cannot easily be compromised. It is also important to ensure that there are proper resource constraints on their usage, or else the rest of the database may be indirectly affected.

The `CREATE USER` statement is used to create a database user and its associated schema. In the following example, the user is identified by a password, and the account follows the policy specified by `org_profile`.

```
CREATE USER jsmith IDENTIFIED BY NoOne!Knows PROFILE org_profile
       DEFAULT TABLESPACE data_ts TEMPORARY TABLESPACE temp_ts;
```

A *profile* specifies a named set of resource limits and password parameters that restricts excessive consumption of system resources and enforces constraints on the passwords. The password-specific parameters provide password management including account locking, password aging, password history, and password complexity verification. The password verification function is perhaps the most important control to ensure that users pick complex passwords, making it difficult for intruders to guess them. The `FAILED_LOGIN_ATTEMPTS` parameter limits brute-force password-guessing attacks by locking the account after a specified number of incorrect logins.

```
CREATE PROFILE org_profile LIMIT
  FAILED_LOGIN_ATTEMPTS 6         -- attempts allowed before locking
  PASSWORD_LIFE_TIME 180          -- max life-time for the password
  PASSWORD_VERIFY_FUNCTION ora12c_verify_function; -- Password complexity
check
```

The dictionary views `DBA_USERS` and `DBA_PROFILES` describe the users and profiles, respectively. The privilege to create users must be limited to the DBA or the security administrator. Each user should have an assigned tablespace; otherwise, any objects they create would go into the SYSTEM tablespace, thus creating contention between the data dictionary objects and the user objects.

Oracle Multitenant Database Users

Oracle Multitenant, an Oracle Database 12*c* option, includes both common and local users. A *common* user is created in the container database and has the same user name and password in all of the pluggable databases that are part of the container database. The common user can have privileges that are granted at the container level, and other privileges that are granted in each pluggable database. The privileges can be different in each of the pluggable databases, but the user doesn't need to be created in each pluggable database.

To create a common user for the container database and all of the pluggable databases, log in to the container database as `SYSTEM` and create a user with `CONTAINER=ALL`. Note that all common user names begin with the prefix `C##`.

```
SQLPLUS> CONNECT SYSTEM@root
Enter password: **********
Connected.
SQLPLUS> CREATE USER C##DB_ADMIN
IDENTIFIED BY IronMan4
CONTAINER = ALL;
```

A *local* user, on the other hand, is created in the pluggable database, and does not have access to the container. This is good for the administrator who manages a pluggable database but does not manage the overall system. To create a local user, connect to the pluggable database as `SYSTEM`, create the user, and grant the needed roles and privileges as before, but specify `CONTAINER=CURRENT` instead of `CONTAINER=ALL`.

```
SQLPLUS> CONNECT SYSTEM@pdb1
Enter password: *********
Connected.
SQLPLUS> CREATE USER pdb1_admin
IDENTIFIED BY SpiderMan3
CONTAINER = CURRENT;
```

Storing Passwords

Users are expected to provide the password when they connect to the database, but applications, middle-tier systems, and batch jobs cannot depend on a human to type the password. Earlier, a common way to provide passwords was to embed user names and passwords in the code or in scripts. This increased the attack surface and people had to make sure that their scripts were not exposed to anyone else. Also, if passwords were ever changed, changes to the scripts were required. Now you can store password credentials by using a client-side Oracle wallet. This reduces risks because the passwords are no longer exposed on command-line history, and password management policies are more easily enforced without changing application code whenever user names or passwords change.

To configure password storage using an Oracle wallet, set the `WALLET_LOCATION` parameter in the *sqlnet.ora* file. The applications can then connect to the database without providing login credentials, as follows:

```
CONNECT /@hr_db.example.com
```

Authentication Methods

Users need to be authenticated before being allowed to connect to the database. Oracle supports different means of authentication including passwords stored locally within the database or in directories. Users can also be authenticated by the operating system, using the `IDENTIFIED EXTERNALLY` clause when creating the user, or by various third-party authentication services, including Kerberos, SSL/TLS, and RADIUS. Passwords are only used for one-way authentication of the user to the database, while Kerberos and PKI support mutual authentication, ensuring that the user is indeed connecting to the proper database.

Oracle clients and servers communicating over SSL/TLS must have a wallet containing an X.509 certificate, a private key, and a list of trusted certificates. An administrator sets up this configuration using Oracle Wallet Manager to create the wallet to store the PKI credentials and Oracle Net Manager to configure *sqlnet.ora* and *listener.ora* for SSL authentication. The following example shows how to create a user with the PKI certificate:

```
SQL> CREATE USER jsmith IDENTIFIED EXTERNALLY AS
'cn=jsmith,OU=HR,O=oracle,c=US';
```

Users can authenticate to the database using Kerberos in environments that support that service. This capability is configured by setting the required parameters in the Oracle Database server and client *sqlnet.ora* files using Oracle Net Manager. The following example shows how to create an externally authenticated user that corresponds to the Kerberos user:

```
SQL> CREATE USER jsmith IDENTIFIED EXTERNALLY AS 'jsmith@example.com';
```

You can now connect to an Oracle Database server without using a user name or password as follows:

```
$ sqlplus /@hr_db.example.com;
```

Centralized User Management

In an enterprise with a number of users accessing a number of databases, it is difficult to manage unique accounts for each user in every database. Oracle Enterprise User Security (EUS) enables centralized management of users and roles across multiple databases in Oracle Internet Directory, which integrates with other directories such as Microsoft Active Directory. Such users are called *enterprise users*, and they can be assigned enterprise roles that determine access privileges across multiple databases. An enterprise role consists of one or more global roles that grant database privileges to specific databases.

EUS allows users and administrators to be authenticated by Oracle Internet Directory using a password, Kerberos, or SSL. Upon connecting, the database refers to the directory for user authentication, authorization (roles) information, and schema mapping. Enterprise users can have their own schema, or they can share a global schema in the databases they access. Here is an example of an enterprise user with an exclusive schema, `jsmith`.

```
CREATE USER jsmith IDENTIFIED GLOBALLY AS 'CN=jsmith,OU=HR,O=oracle,C=US';
```

Users with Administrative Privileges

Certain users can connect with special *administrative privileges*, such as `SYSDBA` and `SYSOPER`, to allow maintenance operations even when the database is not open. These users can authenticate using a network-based authentication service such as Oracle Internet Directory or based on membership of the connecting user in a particular operating system group.

If a user must connect with administrative privilege using a password for authentication, the password is stored outside the database in a password file, which is administered using the `orapwd` command. User management functions such as locking an account after multiple failed login attempts are not available for users in the password file, although each failed attempt will cause an exponentially increasing delay to limit password guessing when the database is running.

Proxy Authentication and Authorization

Sometimes administrators need to connect to an application schema to perform maintenance. Sharing the application schema password among several administrators would provide no accountability. Instead, proxy authentication allows the administrators to authenticate with their own credentials first and then proxy to the application schema. In such cases, the audit records show the actual user who performed the maintenance activities. This form of proxy authentication is supported in Oracle Call Interface (OCI), JDBC, and on the SQL*PLUS command line. Here is an example where the user `app_dba` is allowed to connect to the database and act as `hrapp`.

```
ALTER USER hrapp GRANT CONNECT THROUGH app_dba;
```

Now the user `app_dba` can connect using his own password and assume the identity of the `hrapp` user by proxy as follows:

```
CONNECT app_dba[hrapp]
Enter password: <app_dba_password>
```

Basic Access Control

Every object in the database, such as a table, view, or procedure, is contained within a schema. A *schema* is a user in the Oracle Database that owns objects. The schema user generally has full access to the objects contained within that schema. Access by other users is determined by object privileges, which allow a user to perform a particular operation on one specific object. Some typical operations for objects are `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `ALTER`, and `EXECUTE`.

The schema user that owns an object has the ability to grant object privileges to other users. In addition, if an object privilege is granted with `GRANT OPTION`, the recipient of the grant also gains the ability to grant the same privilege to others. The ability to propagate grants in this way is powerful and should be used sparingly.

Here is an example of creating a user with just a few privileges: to create a session and connect to the database, to select from the DEPARTMENTS table, to execute the `ADD_DEPARTMENT` procedure, and full permissions to read and change data on the ADVENTURES table:

```
SQL> CREATE USER jsmith IDENTIFIED BY "Raider5!";
SQL> GRANT CREATE SESSION TO jsmith;
SQL> GRANT SELECT ON hr.departments TO jsmith;
SQL> GRANT EXECUTE ON hr.add_department TO jsmith;
SQL> GRANT SELECT, INSERT, UPDATE, DELETE ON hr.adventures TO jsmith;
```

The dictionary table `DBA_TAB_PRIVS` shows the object privileges that have been granted. This gives detail about the object including the schema owner and which privileges were granted. This table can be used for reporting privileges and managing the level of permissions.

```
SQL> SELECT GRANTEE, OWNER, TABLE_NAME, PRIVILEGE
   FROM DBA_TAB_PRIVS
   WHERE GRANTEE='JSMITH';
GRANTEE     OWNER  TABLE_NAME     PRIVILEGE
--------    ------ -----------    ----------
JSMITH      HR     DEPARTMENTS    SELECT
JSMITH      HR     ADVENTURES     SELECT
JSMITH      HR     ADVENTURES     INSERT
JSMITH      HR     ADVENTURES     DELETE
JSMITH      HR     ADVENTURES     UPDATE
JSMITH      HR     ADD_DEPARTMENT EXECUTE
```

When privileges are no longer needed on an object, they should be revoked.

```
SQL> REVOKE DELETE ON hr.adventures FROM jsmith;
```

System Privileges and Roles

Object privileges allow for very fine control over the data that a user can access, but sometimes an administrator may require access to many objects. System privileges allow access to all objects of a particular type; for example, `SELECT ANY TABLE` allows a user to select from any table in any schema, and `EXECUTE ANY PROCEDURE` allows execution of any PL/SQL procedure or function. Other system privileges apply to operations that do not involve a specific object, such as the ability to create objects, users, and roles; to change session and system parameters; and to export and import the database. As you can see, these are privileges for the administrator who can perform operations that have an impact across multiple schemas and objects.

Another convenient feature for managing privileges is the ability to group multiple object and system privileges into a role. Roles are especially useful when there is a need to grant a consistent set of privileges to several users. The roles are easier to manage than individual privileges and can be matched up with an application or a job function. Roles can be granted to other roles, allowing a large role like the DBA role for the database administrator to be built up out of smaller components. Like the `GRANT` option for object privileges, a system privilege or role can be granted with `ADMIN OPTION`, which allows the recipient to grant the role or privilege to others.

The dictionary tables listed in the following table show the roles and privileges granted to each user or role. For example, selecting from these tables shows that the DBA role is extremely powerful, with more than 200 system privileges including `CREATE` and `ALTER SESSION`; `CREATE` and `ALTER ANY TABLE`; `SELECT`, `INSERT`, `UPDATE`, and `DELETE ANY TABLE`; `EXPORT` and `IMPORT FULL DATABASE`; `DROP` and `CREATE TABLESPACE`; `EXECUTE ANY PROCEDURE`, and over a dozen roles.

Dictionary Table	Contents
`DBA_TAB_PRIVS`	Object privilege grants to roles or users
`DBA_SYS_PRIVS`	System privilege grants to roles or users
`DBA_ROLE_PRIVS`	Role grants to users or other roles
`DBA_ROLES`	All defined roles

Least Privilege and Separation of Duty

The principle of least privilege denotes the idea that each user of the system should be granted only the minimum set of privileges needed to accomplish their intended tasks or functions. When granting privileges to a user or role, it is preferable to grant specific object privileges that are needed rather than broad system privileges that allow access to all objects in the database. Similarly, it is better to create roles that each contain a few privileges designed to be used for a particular function instead of very powerful roles like the built-in DBA role. Granting several of these smaller roles to a user allows for a close match to the tasks that the user needs to perform without granting extra privileges that are not required.

Closely related to the principle of least privilege is the concept of separation of duty. This is the notion that privileges should be divided among several users instead of a single powerful individual. Dividing administrative privileges in this way improves accountability and makes trusted administrators less likely to abuse their privileges.

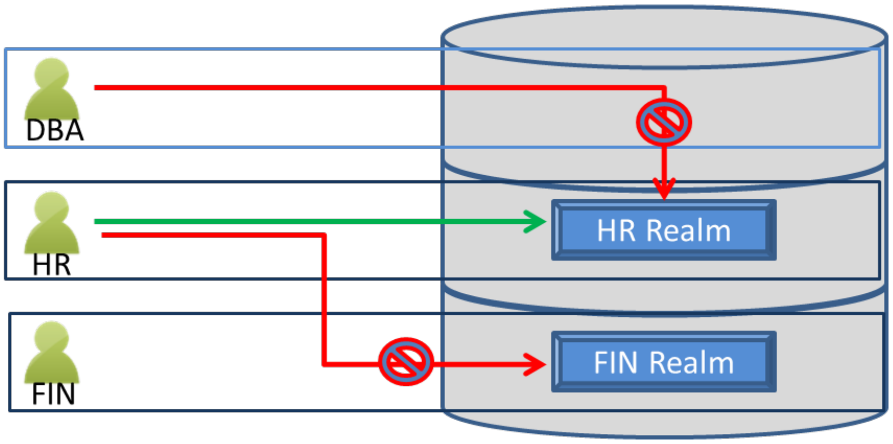
To support the principles of least privilege and separation of duty, Oracle Database has long included a `SYSOPER` administrative privilege, which allows an administrator to perform certain tasks like starting and stopping the database without having the full range of powers conferred by the `SYSDBA` privilege. Oracle Database 12*c* adds additional administrative privileges called `SYSBACKUP`, `SYSDG`, and `SYSKM`, to enable database backups, Data Guard administration, and key management, respectively. With these targeted privileges, one or more administrators can perform all of the normal operations to manage a database without needing the all-powerful `SYSDBA` privilege.

Controlling Privileged Users

System privileges and powerful roles give significant control of the database, including the ability to view all data and make changes to the data. Some administrative users need these powerful privileges for maintenance, tuning, and backups, but they don't need access to all of the data. Even though the administrative users are trusted, it is important to secure company data assets and personal information even from these privileged accounts in order to prevent unauthorized use by insiders or attackers.

Oracle Database Vault provides several kinds of operational controls within the database including realms, which enforce limits on access to specified objects such as tables and views. After creating a Database Vault realm, objects are added to the realm and database users can be designated as realm participants. This provides access only to the realm participants, and excludes other users, even if they have powerful system privileges like `SELECT ANY TABLE` that would otherwise allow them to access the objects in the realm.

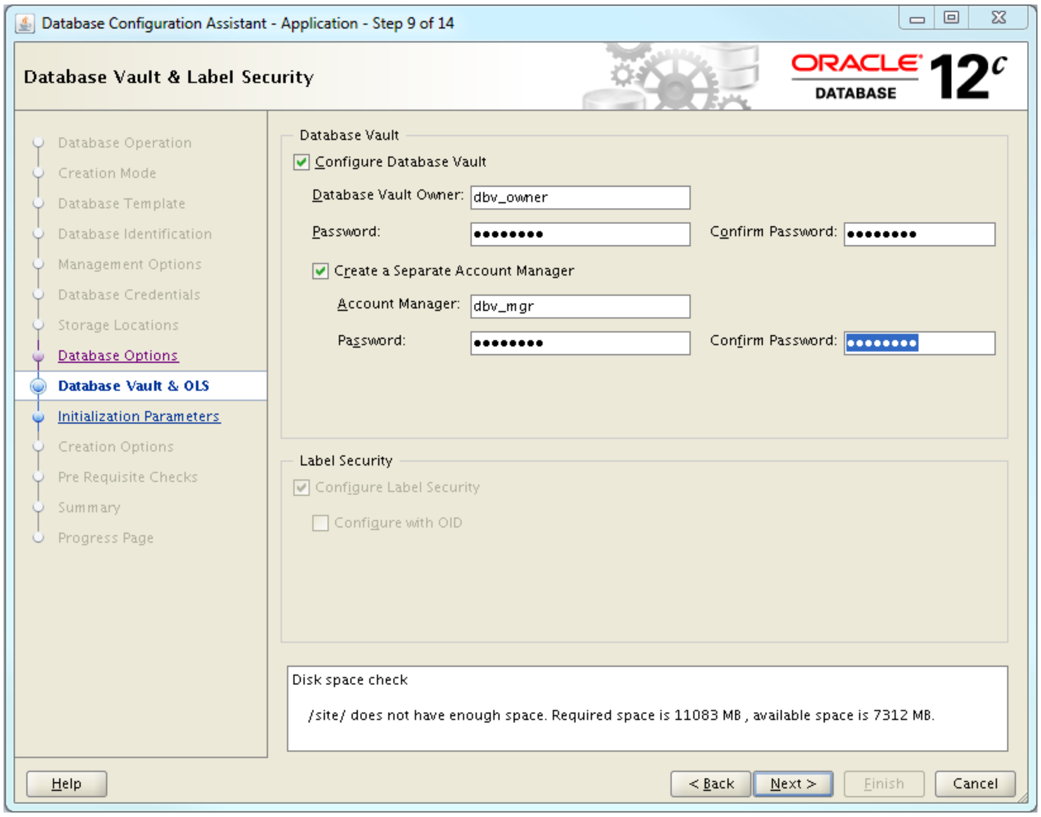
The following illustration shows an example of two realms, protecting database schemas containing human resources (HR) and finance (FIN) data. Once enabled, the realms prevent privileged administrative users or other application owners from using their elevated privileges to access data. The privileged application owner HR is prevented from accessing data inside the FIN realm, and even an administrator with the DBA role is unable to access data in the HR and FIN realms.



In addition to regular realms, Oracle Database 12*c* adds the ability to create mandatory realms. A regular realm will block the use of system privileges such as `SELECT ANY TABLE` if the user is not a realm participant, but it doesn't block the schema owner or other users who gain access to the data using object privileges. Mandatory realms prevent access by anyone who is not a realm participant. One popular use for a mandatory realm is to continue to protect sensitive data during patching and upgrades, when an administrator needs to make changes to the application schema but should not have access to the data tables in that schema.

When Oracle Database Vault is configured, a couple of additional users are created. The first of these is the Database Vault owner, who can create and manage realms to control access to sensitive data. The second user is the Database Vault account manager, who has the responsibility for creating users in the database. While a single user could perform both functions, the ability to divide these duties among multiple users allows for separation of duty as described earlier. Furthermore, there is a `DVOWNER` role that can be granted to other users to delegate the ability to manage Database Vault realms. This role should be granted to administrators who are responsible for the security configuration of the database, rather than the general database administrator.

The following illustration shows the use of the Database Configuration Assistant for enabling Oracle Database Vault. Management of Database Vault requires the use of these specialized users and roles. The `SYSDBA` administrative privilege cannot be used for realm or user management when Database Vault is enabled.


Database Configuration Assistant - Application - Step 9 of 14

Database Vault & Label Security

Database Vault

✓ Configure Database Vault

Database Vault Owner: dbv_owner

Password: •••••••• Confirm Password: ••••••••

✓ Create a Separate Account Manager

Account Manager: dbv_mgr

Password: •••••••• Confirm Password: ••••••••

Label Security

✓ Configure Label Security

☐ Configure with OID

Disk space check

/site/ does not have enough space. Required space is 11083 MB , available space is 7312 MB.

Managing Granted Privileges

Following the principle of least privilege means that each user, and each role granted to users, should have only the minimal set of privileges needed to perform their intended function. While there are dictionary tables to show which permissions and roles have been granted, it is much harder to determine which ones are actually needed. This is especially true in systems that have been in use for some time, since privilege and role grants tend to accumulate and it is difficult to know when it is safe to revoke them.

Oracle Database Vault 12*c* includes a new feature called Privilege Analysis that captures privileges as they are used at run-time and generates a series of reports. These reports can be used to find privileges that may no longer be needed or even to generate scripts to revoke unused privileges automatically.

Here is an example to enable the privilege capture for all users of the database.

```
SQLPLUS> BEGIN DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE
(NAME => 'dba_capture_all_privs',
DESCRIPTION => 'privilege_analysis_for_all_users',
TYPE => DBMS_PRIVILEGE_CAPTURE.G_DATABASE);
END;
/
PL/SQL procedure successfully completed.
```

After a suitable interval to capture the privileges used during normal operation, the DBA_USED_PRIVS and DBA_UNUSED_PRIVS views will reveal which privileges have been used and, more importantly, which granted privileges have not been used.

```
SQLPLUS> BEGIN DBMS_PRIVILEGE_CAPTURE.GENERATE_RESULT
(NAME => 'dba_capture_all_privs');
end;
/
PL/SQL procedure successfully completed.
SQLPLUS>SELECT USERNAME, USED_ROLE, SYS_PRIV, OBJ_PRIV, USER_PRIV,
OBJECT_OWNER,OBJECT_NAME
FROM DBA_USED_PRIVS;

SQLPLUS> SELECT USERNAME, USED_ROLE, SYS_PRIV, OBJ_PRIV, USER_PRIV,
OBJECT_OWNER, OBJECT_NAME
FROM DBA_UNUSED_PRIVS;
```

A script to revoke the unused privileges could be generated using a SQL statement like the following. An administrator should review the script to verify the list of privileges to be revoked before executing it.

```
--Generate a script to revoke
SQLPLUS> SELECT 'revoke '||OBJ_PRIV||' on '||OBJECT_OWNER||'.'||
OBJECT_NAME||' from '||USERNAME||';'
FROM DBA_UNUSED_PRIVS;
```

The views available to display the information generated in the privilege capture are as follows.

View	Description
DBA_PRIV_CAPTURES	Lists information about existing privilege analysis policies.
DBA_USED_PRIVS DBA_UNUSED_PRIVS	Lists the privileges that have (or have not) been used for reported privilege analysis policies.
DBA_USED_OBJPRIVS DBA_UNUSED_OBJPRIVS	Lists the object privileges that have (or have not) been used for reported privilege analysis policies.
DBA_USED_OBJPRIVS_PATH DBA_UNUSED_OBJPRIVS_PATH	Lists the object privileges that have (or have not) been used for reported privilege analysis policies. It includes the object privilege grant paths.
DBA_USED_SYSPRIVS DBA_UNUSED_SYSPRIVS	Lists the system privileges that have (or have not) been used for reported privilege analysis policies.
DBA_USED_SYSPRIVS_PATH DBA_UNUSED_SYSPRIVS_PATH	Lists the system privileges that have (or have not) been used for reported privilege analysis policies. It includes the system privilege grant paths.
DBA_USED_PUBPRIVS	Lists all the privileges for the PUBLIC role that have been used for reported privilege analysis policies.
DBA_USED_USERPRIVS DBA_UNUSED_USERPRIVS	Lists the user privileges that have (or have not) been used for reported privilege analysis policies.
DBA_USED_USERPRIVS_PATH DBA_UNUSED_USERPRIVS_PATH	Lists the user privileges that have (or have not) been used for reported privilege analysis policies. It includes the user privilege grant paths.

CHAPTER 2

Preventing Direct Access to Data

The previous chapter describes how to configure the Oracle database to control who has access to which data. Setting up this access control to be enforced by the database is critically important to ensure that all of the data is available to those who need it and not accessible to anyone else. But of course, attackers are not limited to attacking where defenses are the strongest; they will not keep trying the locked and deadbolted front door when there is an open window nearby.

If the database is correctly set up to control access to data, then the attacker will naturally try to bypass this enforcement by bypassing the database. One way to do that is to gain access to the system as a privileged operating system user such as `root` or `oracle`. Such a user can alter the behavior of the database program itself to ignore the access control settings that have been configured. The best defense against this type of attack is to prevent attackers from gaining access to the host system, following guidelines for operating system hardening such as those produced by the Center for Internet Security.

Another line of attack is to bypass the database by reading or writing the data directly, either when it is stored in disk files or when it is in transit on the network between two systems. Encryption is a useful technique for protecting data in both of these situations because it reduces the problem of protecting a large amount of data to the simpler problem of protecting a small encryption key. As long as the attackers do not possess the key, any encrypted data they manage to intercept provides nothing useful. Encryption is also frequently required in order to comply with regulations or security standards regarding sensitive or personally identifiable information.

This chapter explains how to configure and use encryption to protect data at rest and in transit. It also explains some of the considerations and options available for securely storing the encryption keys that ultimately protect the encrypted data.

Data at Rest

When information is written to the database, it will eventually be stored in files on disk. To ensure that an attacker cannot read the information directly from the disk files, each application could encrypt the data before storing it in the database. However, this would require extra programming for the application to encrypt data before storing it and decrypt the data it retrieves. It would also require the application to manage the encryption keys and securely store them somewhere. If multiple applications share the information in the same database, all of them need to cooperate to encrypt and decrypt the data.

To simplify this process, Oracle provides Transparent Data Encryption (TDE) as part of Oracle Advanced Security. The encryption is transparent because the database automatically encrypts data before it is written on disk and decrypts it when reading from the disk. Applications that store and retrieve data in the database only see the decrypted or plaintext data.

Because the encryption and decryption takes place automatically, this is not an access control mechanism for Oracle database users, but rather a way to prevent bypassing the database to access the data directly. Users and applications do not present a decryption key when they retrieve data using a `SELECT` statement. Instead, the database enforces the access control rules described in the previous chapter and denies access if the user is not authorized to see the data. If the access is allowed, the data is automatically decrypted before it is returned. Whether access is allowed or not, encrypted data is never returned to the user.

Transparent Data Encryption is configured by specifying which data is to be encrypted, along with a few options about how the encryption is to be performed. The two choices for specifying what to encrypt are called *column encryption,* where only specific columns within certain tables are encrypted, and *tablespace encryption,* where all of the columns in all tables within a tablespace are encrypted.

Column encryption is specified by setting the `ENCRYPT` attribute of a column as shown in the following example. Column encryption is most appropriate when only a few columns contain sensitive data that needs to be protected, and these columns make up a small portion of the total amount of data stored in the database. In this case, it may be more efficient to incur the overhead of encrypting and decrypting only the sensitive columns while storing the rest of the data in plaintext form. This method can also be used on existing data at any time by using the `ALTER TABLE` command.

```
CREATE TABLE employee (
  first_name VARCHAR2(128),
  last_name VARCHAR2(128),
  empID NUMBER,
  salary NUMBER(6) ENCRYPT
);
```

For most use cases, however, tablespace encryption provides some noteworthy advantages. Encrypting an entire tablespace removes the need to determine which data is important enough to protect and which is not. Furthermore, it ensures that no sensitive data is accidentally left out, either through oversight or changing requirements. When the database runs on modern hardware that includes special instructions in the CPU to accelerate cryptographic operations, the performance overhead for encrypting all data is negligible, so that there is no need to make choices about what to protect. Tablespace encryption is configured when the tablespace is created, as shown here:

```
CREATE TABLESPACE encrypt_ts
  DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf'
  SIZE 100M
  ENCRYPTION USING 'AES128'
  DEFAULT STORAGE (ENCRYPT);
```

Data in Transit

Another way for an attacker to bypass the database and gain direct access to data is to intercept the data as it travels over the network, such as between a client and the database server. On many networks it is relatively easy for an attacker to capture network traffic that is intended for another system and then extract whatever information was transmitted between the two systems. For this reason, it is important to encrypt data when it is sent over the network as well as when it is stored on disk.

The ability to protect data on the network, using either SSL or Oracle native encryption, was formerly a part of the Advanced Security Option but is now a standard feature of Oracle Database. This feature can be configured to provide both encryption (to prevent others from reading data sent over the network) and integrity protection (to prevent others from modifying or replaying the data).

The network data protection settings for each Oracle client and server are configured in the *sqlnet.ora* file. The applicable settings for each system include the list of cryptographic algorithms it supports and one of the four choices shown in the following list for encryption and for integrity protection. Based on these settings, the two systems negotiate when a connection is established to determine whether to enable encryption, integrity protection, or both and which cryptographic algorithms to use.

- **REJECTED** The system will not enable this service (either encryption or integrity protection). If the other system requires this service, the connection will fail.
- **ACCEPTED** The system will not initiate a request to enable the service, but it will agree to do so if requested by the other system.
- **REQUESTED** The system will request to enable the service, but will set up a connection without it if the other system refuses.
- **REQUIRED** The system will request to enable the service. If the other system refuses, the connection will fail.

Key Management and Storage

Modern cryptographic systems do not depend on secret encryption algorithms to keep data secure. Professional cryptographers learned long ago that it is safer to use well-known algorithms that have been carefully studied by experts to find any possible weaknesses. The practical importance of this fact is that the security of the data being protected depends entirely on maintaining the secrecy of the keys used for encryption. Proper key management is essential to keep an attacker from discovering or guessing a secret key and gaining access to whatever data it protects.

Administrators perform all of the key management operations needed for the Oracle database using a series of SQL commands or a management interface that invokes these commands. Beginning in Oracle Database 12*c*, there is a new option to connect to the database as `SYSKM` instead of `SYSDBA` to perform key management operations. Following the advice of the previous chapter about using the minimum privilege necessary to perform a task, the `SYSKM` administrative privilege is designed to provide the ability to perform all key management tasks without the unlimited power permitted by `SYSDBA`.

Oracle Database 12*c* also introduces a new unified set of key management commands under the `ADMINISTER KEY MANAGEMENT` statement. This statement supports all of the key management operations that formerly required a combination of the `orapki` utility and the `ALTER SYSTEM` statement. See the following examples of using the new commands.

```
Connect using the SYSKM administrative privilege.
sqlplus user/password AS SYSKM

Create a new Oracle wallet.
SQL> ADMINISTER KEY MANAGEMENT
  CREATE KEYSTORE 'keystore_location'
  IDENTIFIED BY keystore_password;

Open an Oracle wallet.
SQL> ADMINISTER KEY MANAGEMENT
  SET KEYSTORE OPEN IDENTIFIED BY keystore_password;

Make a backup copy of an Oracle wallet.
SQL> ADMINISTER KEY MANAGEMENT
  BACKUP KEYSTORE
  IDENTIFIED BY keystore_password
  TO 'backup_location';

Set or rotate the TDE master encryption key.
SQL> ADMINISTER KEY MANAGEMENT
  SET ENCRYPTION KEY
  IDENTIFIED BY keystore_password;
```

The most important aspect of key management is storing the keys in a safe location. The Oracle Database and other Oracle products use a special file called a "wallet" to store encryption keys and other secret data. The contents of the wallet, in turn, are encrypted using a key derived from a password that must be provided by a user whenever the wallet is opened to access the keys inside. The password used to encrypt the wallet is not stored anywhere, so there is nothing for an attacker to discover. Of course, if the password is somehow lost or forgotten, there is no way to open the wallet or access the data that is encrypted using that wallet.

Typically, an Oracle Database using TDE stores its master key in an Oracle wallet, which it opens when the database is started. In some installations, however, it is not feasible to have an administrator present to provide the wallet password when the database is started. For this situation, Oracle wallets can be configured with an "auto-login" option, which allows the database to open the wallet without a password. While the contents of the wallet are still obfuscated in this case, this does not provide the same level of protection as a regular password-protected wallet. This alternative should only be used when there are sufficient external protections in place to prevent an attacker from gaining access to the wallet file.

Another alternative to make keys conveniently accessible to the database and enjoy some additional security benefits is to store the keys remotely on a separate enterprise key management server, such as the Oracle Key Vault, which the database accesses over the network. Communications between the database and the key server are protected using a network protocol such as SSL. This also allows the database to be authenticated automatically by the key server without the need for an administrator to supply a password when the database is started. The key server provides a central location where keys, which may be the most valuable data that an organization possesses, can be safely backed up or replicated to ensure that they are always available. A specialized key server also automates the process of managing the lifecycle of each key, including tracking its creation and ownership, the purposes for which the key should be used, and when the key needs to be rotated or replaced with a new key.

Many regulations, such as those developed by the Payment Card Industry (PCI), require periodic rotation of encryption keys to limit the exposure if a single key is somehow disclosed. The Oracle Database uses a two-tier architecture to minimize the cost of key rotation and thus make it practical to rotate keys more frequently. Oracle Transparent Data Encryption uses a master key, which is stored in a wallet or key server. Instead of encrypting the data directly, the master key is used to encrypt the internally generated keys that are used for column and tablespace encryption. When the master key is replaced with a new key, there is no need to re-encrypt all of the data, only the much smaller set of data encryption keys.

CHAPTER 3

Advanced Access Control

The first chapter explained the importance of controlling which users have access to which data by selectively granting object privileges. Users should have access only to the tables containing data they need to perform their tasks. However, even when an object privilege such as `SELECT` or `INSERT` is granted to a user for a specific table, the privilege provides access to everything within that table. Database tables for most applications, however, contain much more data than any single user should be able to access.

As a simple example, consider a table of employees such as the EMP table in the `SCOTT` sample schema. It might be appropriate to allow all users to view the basic information about employee names and departments, but not information about compensation including the SAL and COMM columns. Similarly, an organization might want to restrict access to specific rows in the table, perhaps allowing employees to view data only for their own department or division.

The long-standing solution to this challenge in a relational database is to create a view containing a `WHERE` clause that restricts access as desired and grant users access only to the view, not the underlying table. For example, the view shown next provides access only to the noncompensation information about employees in a single department.

```
SQL> create or replace view emp_dept20 as
  2  select empno, ename, job, mgr, hiredate, deptno
  3  from emp where deptno = 20;

View created.

SQL> select * from emp_dept20;

     EMPNO ENAME      JOB              MGR HIREDATE      DEPTNO
---------- ---------- --------- ---------- --------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80         20
      7566 JONES      MANAGER         7839 02-APR-81         20
      7788 SCOTT      ANALYST         7566 19-APR-87         20
      7876 ADAMS      CLERK           7788 23-MAY-87         20
      7902 FORD       ANALYST         7566 03-DEC-81         20
```

The difficulty with using views for this purpose is that the criteria for selecting which data to include must all be embedded in the view definition. While it is possible to include subqueries within the `WHERE` clause, for example to determine the requesting user’s department number, it quickly becomes complicated to write a SQL query that accurately expresses the desired policy. The difficulty is compounded by the fact that there are typically exceptions to the regular policy, say to provide greater access for HR administrators or for the manager of a department.

The Oracle Database addresses this need with a feature called Virtual Private Database (VPD). Using VPD, a PL/SQL function provided by the administrator is executed to generate the appropriate `WHERE` clause each time the table is accessed. This makes it relatively easy to do whatever computation is needed to determine the allowed access. The following listing shows a simple PL/SQL function with the same policy as the view in the preceding example, extended to work for any department and provide increased access for HR administrators.

```
function emp_policy(p_schema in varchar2, p_table in varchar2)
  return varchar2 is
    user     VARCHAR2(100);
    match    NUMBER;
    my_dept  NUMBER;
  begin
    -- Get the requesting user
    user := sys_context('USERENV', 'SESSION_USER');

    -- Check if requestor is an HR employee
    select count(*) into match from emp
        where ename = user and job like 'HR%';
    if match > 0 then
        return '1=1'; -- match all rows
    end if;

    -- Find the requesting user's department
    select deptno into my_dept from emp
        where ename = user;

    return 'deptno = ' || my_dept;
  end;
```

Finally, VPD can support other policy variations that would be difficult or impossible to implement using views alone. Instead of always excluding a column containing sensitive information, a VPD policy can include the column but only allow access to its contents to certain users while others retrieve null values. It is also possible for the VPD policy to provide access to different rows depending on the operation the user is performing. This is useful, for example, to allow users to view information about all employees but only update their own.

While VPD offers full flexibility in determining the filtering predicate, the developer is responsible for the access control logic and managing any application-specific rules and constraints. For example, in this case, the access control was based on the JOB and DEPTNO columns. In some other cases, the access control decision may be controlled by a specific application role.

Controlling Access Using Data Labels

One particular method for using fine-grained access control as described in the preceding section involves attaching a label to each item of data that describes its sensitivity or importance. In many government and corporate environments, a document might be labeled as TOP SECRET or INTERNAL USE ONLY, and then only people who have a sufficiently high "clearance" level are allowed access to those documents. Typically the labels reflect an ordered set of levels, and each user is assigned a maximum level that he or she is permitted to access. This capability allows the database to inherently know which data is sensitive and restrict access to it accordingly.

To support this common access control model, Oracle Label Security (OLS) simplifies the process of assigning labels to data and users and enforcing access control based on those labels. Associated with every row in a table protected by OLS is a label that indicates the sensitivity of the data. The label for each row can be set explicitly based on business logic, but more often the system sets the label automatically based on the label of the user session that inserted the row in the table. The label of the user session, in turn, is calculated from a variety of factors, including the identity of the user, the type of connection to the database, and so on. A user's label can be thought of as an extension to standard database privileges and roles. Oracle Label Security is enforced within the database, providing strong security and eliminating the need for complicated application views.

The format of a label is expressive enough to accommodate virtually any data classification scheme. Every label includes a level, ordered from lowest to highest, to indicate the overall sensitivity of the data. Within a level, optional components called *compartments* and *groups* can be used to segregate information based on attributes such as project, department, or geographic region. Unlike with the level component, there is no ordering between compartments. Groups may include other groups, which provides a convenient way to represent divisions based on geography or organizational hierarchy. Figure 3-1 shows an example of a classification scheme using all three label components.

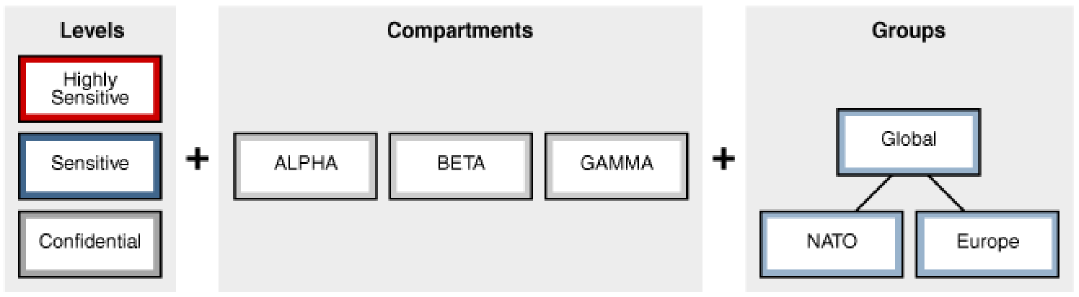


Figure 3-1: Components of a label

Once labels exist for both the data and user sessions, a simple set of rules is used to compare the labels and decide if access is permitted. A user can read from a labeled row in a table if the user's label has a level at least as high as the data label and has all of the compartments and at least one of the groups (or a parent of one of the groups) specified in the data label. Using the example in Figure 3-1, a user whose label includes the Global group can access data that is labeled with either the NATO or Europe group.

The key advantage to using Oracle Label Security is flexibility and simplified administration. OLS provides the same type of row-level access control as VPD, but does not require the administrator to create the PL/SQL policy function to enforce the access rules. Once a system is in place to assign labels to data and user sessions, access control is uniformly enforced everywhere without requiring explicit decisions about who should be able to access the data in each individual table.

Applying Access Control Policies to Application End Users

The methods described so far address the problem of controlling data access by database users. However, in modern three-tier applications, the application end users do not interact directly with the database. Database queries and updates are typically performed by a single database user that represents the entire application rather than by the individual end user.

Because the end user's identity is unknown to the database, per-user access control policies must be enforced by the application instead. Besides requiring extra software development in the application, this approach can lead to inconsistent enforcement, especially when there are elements of the system that can bypass the application and connect directly to the database.

Introduced in Oracle Database 12*c*, Oracle Real Application Security (RAS) provides the next generation of application access control framework within the database, enabling three-tier and two-tier applications to declaratively define, provision, and enforce their access control requirements at the database layer. Oracle RAS introduces a policy-based authorization model that recognizes application-level users, privileges, and roles within the database, and then controls access on both static and dynamic collections of records representing business objects.

With RAS, the identity of the application end user is propagated to the database so that access control policies can be enforced within the database itself. Each application end user is represented by a lightweight "application user" that does not have its own schema to store data objects or dedicated connection to the database, as a regular database user would. The application can create any number of application user sessions and switch between them while using a single connection to the database. This application user session allows the database to know which end user has initiated the operation being performed by the application so that the database can enforce the appropriate access control policies.

Real Application Security can enforce fine-grained restrictions on access to columns and rows within a database table, just like the Virtual Private Database feature. However, RAS uses a more general, declarative syntax for the administrator to specify these restrictions. First, the administrator creates one or more data realms for each table to be protected. Each realm identifies an applicable subset of the rows within the table using the same syntax as the `WHERE` clause in a SQL query. Then an access control list (ACL) is attached to each realm to identify which users or roles have permission to perform which operations on the data within that realm.

Furthermore, RAS allows the database to enforce additional security policies that are unique to each application. The application can define its own privileges in addition to the usual `SELECT`, `INSERT`, `UPDATE`, and `DELETE` to represent more complex operations that are specific to that application. The administrator can specify that access to a column requires the user to have a particular application-defined privilege. Also, the application can include a check for one of its application-defined privileges as part of a SQL statement, causing the statement to take effect only on data where the user has been granted the required privilege.

Figure 3-2 shows an example of a table called EMPLOYEES and its associated data security policy. The policy contains a single realm, which specifies all rows where the DEPARTMENT_ID column contains the value 60 or 100. The ACL associated with this realm allows only users with the application-defined Employee_Role or Manager_Role to select these rows from the table. The SALARY column is further protected, requiring the `VIEW_SENSITIVE_INFO` privilege—also defined by the application—to access the values in this sensitive column. The ACL grants this privilege to users who possess the Manager_Role.

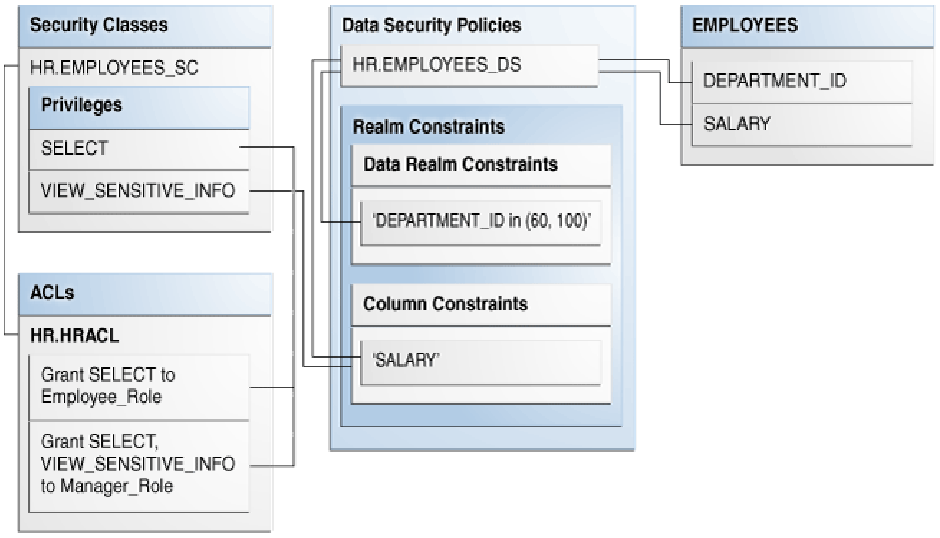


Figure 3-2: Table with data security policy

With built-in support for propagating application users' sessions to the database, Oracle RAS allows security policies on data to be expressed directly in terms of the application-defined users' roles, and data operations. The RAS security model allows uniform specification and enforcement of access control policies on business objects irrespective of the access path. Using declarative access control policies on application data and operations, Oracle RAS enforces security close to the data and enables end-to-end security for both three-tier and two-tier applications.

CHAPTER 4

Auditing Database Activity

Maintaining an audit trail of activity is an essential component of any defense-in-depth strategy for securing a database. Even when access controls are properly configured and privilege grants are minimized, two important risks still remain. The first is that users who need significant privileges to perform their jobs may misuse those privileges. The second is that a user may gain unexpected access via access controls or privilege grants that are unintentionally configured to be more generous than necessary. Auditing is the primary tool for detecting these incidents if they occur so that they can be corrected.

Effective auditing requires audit policies that are selective in capturing the important details about significant events while minimizing the noise from routine activity. It requires secure storage for the audit data so that it is a reliable record of events and especially so that it cannot be manipulated to hide suspicious activities. Finally, it requires convenient ways to search through the collected audit data to find specific information or detect unusual activity.

Research by the Independent Oracle Users Group (IOUG) with Oracle customers shows that 70 percent of enterprises are using native auditing on their databases. Customers audit databases to comply with SOX, HIPAA, PCI DSS, GLB, FISMA, and other international standards. Auditing provides a history of who did what and when and enables organizations to meet stringent controls and reporting requirements. Internal governance, local security policies, and forensic reporting also drive the need to audit. Most enterprises want to be able to attribute any request or change to data, or modification to the database, to specific events that are authorized in terms of who or what was issuing the command, and the business justification behind such interaction. Compliance auditors expect that the custodians of the database, the database administrator (DBA) and their management line, usually up to the CFO and CSO, can account for all accesses and changes to data and the database itself.

The Oracle Database provides the industry's most comprehensive auditing capabilities, enabling detailed information on events to be captured for reporting and alerting. Details such as database name, host name, client program name, event time, event status, event action, database user, object owner, and the SQL statement itself are among the information captured with auditing. Auditing can be configured to log both successful and unsuccessful events. For even finer-grained auditing, Oracle Database supports the ability to audit when specific columns in application tables are referenced. For example, a table containing credit limits could have a fine-grained audit policy that audits only when the credit limit column is updated.

Audit Changes in Oracle Database 12*c*

Oracle Database 12*c* advances the rich database auditing capabilities of prior Oracle releases with expanded auditing options and simplified administration. Oracle Database 12*c* audit policies can be configured to audit based on specific IP addresses, programs, time periods, or connection types, such as proxy authentication. In addition, specific schemas can be easily exempted from auditing. This dramatically reduces the number of audit records generated, and ensures that the relevant audit data can be found when needed.

Oracle Database 12*c* Auditing enables selective and effective auditing inside the Oracle database using policies and conditions. In addition, Oracle Database 12*c* supports the Oracle Database 11*g* auditing syntax. This "Mixed Mode" support enables scripts and any tools that used previous auditing syntax. Oracle Database 12*c* Unified Auditing creates one audit trail for all of the following audit sources:

- Audit records (including SYS audit records) from unified audit policies and fine-grained audit records from the DBMS_FGA PL/SQL package
- Management of the unified audit policies themselves
- Oracle Database Vault
- Oracle Label Security
- Oracle Recovery Manager

- Oracle Data Pump
- Oracle SQL*Loader Direct Load

The unified audit trail, which resides in read-only tables in the `AUDSYS` schema, makes this information available in a uniform format in the `UNIFIED_AUDIT_TRAIL` view. When the database is not writable, audit records are written to operating system files in the $ORACLE_BASE/audit/$ORACLE_SID directory and then loaded back into the unified audit trail when the database becomes writable.

To provide separation of duty from traditional DBA functions, Oracle Database 12*c* introduces two new roles: AUDIT_ADMIN for management of policies and audit trail, and AUDIT_VIEWER for viewing audit data. As these roles are not granted to the DBA role, the DBA is not able to alter the audit collection. Audit data can only be managed using the built-in audit data management package within the database and cannot be directly updated or removed using SQL `UPDATE` or `DELETE` commands. In addition, there is a list of mandatory audit activities in Oracle Database 12*c* that cannot be turned off. These are

- `CREATE/ALTER/DROP AUDIT POLICY`
- `AUDIT/NOAUDIT`
- `EXECUTE` of the DBMS_FGA PL/SQL and DBMS_AUDIT_MGMT PL/SQL packages
- `ALTER TABLE` on the AUDSYS audit trail table (even though this table cannot be altered)
- Top-level statements by the `SYS user or with the SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM administrative privileges`, until the database opens
- Auditing the actions of privileged users until the audit configurations in the system are available
- All configuration changes that are made to Oracle Database Vault

Predefined Audit Policies in Oracle Database 12*c*

Three default audit policies are configured and shipped out of the box. Each has been configured to provide audit settings for common audit use-cases.

1. The Secure Configuration (ORA_SECURECONFIG) Unified Audit policy provides all the secure configuration audit options. This policy is enabled by default.

SQL Command	**`CREATE AUDIT POLICY ORA_SECURECONFIG`**

PRIVILEGES	`ALTER ANY TABLE, CREATE ANY TABLE, DROP ANY TABLE,`
	`CREATE ANY PROCEDURE, DROP ANY PROCEDURE, ALTER ANY PROCEDURE,`
	`GRANT ANY PRIVILEGE, GRANT ANY OBJECT PRIVILEGE, GRANT ANY ROLE,`
	`AUDIT SYSTEM, CREATE EXTERNAL JOB, CREATE ANY JOB, CREATE ANY LIBRARY,`
	`EXEMPT ACCESS POLICY,`
	`CREATE USER, DROP USER,`
	`ALTER DATABASE, ALTER SYSTEM,`
	`CREATE PUBLIC SYNONYM, DROP PUBLIC SYNONYM,`
	`CREATE SQL TRANSLATION PROFILE, CREATE ANY SQL TRANSLATION PROFILE,`
	`DROP ANY SQL TRANSLATION PROFILE, ALTER ANY SQL TRANSLATION PROFILE,`
	`CREATE ANY SQL TRANSLATION PROFILE, DROP ANY SQL TRANSLATION PROFILE,`
	`ALTER ANY SQL TRANSLATION PROFILE, TRANSLATE ANY SQL,`
	`EXEMPT REDACTION POLICY,`
	`PURGE DBA_RECYCLEBIN, LOGMINING,`
	`ADMINISTER KEY MANAGEMENT`

ACTIONS	`ALTER USER, CREATE ROLE, ALTER ROLE, DROP ROLE, SET ROLE,`
	`CREATE PROFILE, ALTER PROFILE, DROP PROFILE,`
	`CREATE DATABASE LINK, ALTER DATABASE LINK, DROP DATABASE LINK,`
	`LOGON, LOGOFF, CREATE DIRECTORY, DROP DIRECTORY;`

2. Oracle Database Parameter Changes (ORA_DATABASE_PARAMETER) audits commonly used Oracle Database parameter settings. By default, this policy is not enabled.

SQL Command	**`CREATE AUDIT POLICY ORA_DATABASE_PARAMETER`**

ACTIONS	`ACTIONS ALTER DATABASE,`
	`ALTER SYSTEM,`
	`CREATE SPFILE;`

3. User Account and Privilege Management (ORA_ACCOUNT_MGMT) predefined Unified Audit policy audits commonly used user account and role settings. By default, this policy is not enabled.

SQL Command	**`CREATE AUDIT POLICY ORA_ACCOUNT_MGMT`**

ACTIONS	`CREATE USER, CREATE ROLE,`
	`ALTER USER, ALTER ROLE,`
	`DROP USER, DROP ROLE,`
	`SET ROLE,`
	`GRANT,`
	`REVOKE;`

Customized Audit Policies

Where default policies do not capture the desired audit data, or where fewer events need auditing, the customized policy can refer to any context attribute of the user session. An example of a customized policy is shown in the following listing:

```
CREATE AUDIT POLICY customer_audpol
  ACTIONS
    INSERT ON sales.CUSTOMERS,
    UPDATE ON sales.CUSTOMERS,
    DELETE on sales.CUSTOMERS
  WHEN 'SYS_CONTEXT(''USERNEV'',''IP_ADDRESS'') = ''192.0.2.1'''
  EVALUATE PER STATEMENT;

AUDIT POLICY customer_audpol;
```

In this example, the policy will monitor any `INSERT`, `UPDATE`, and `DELETE` statements from the client with IP address 192.0.2.1.

Monitoring with Oracle Audit Vault and Database Firewall

While audit generation is simple and highly automated, there are other aspects to developing a comprehensive audit plan. First, a large deployment of Oracle and non-Oracle databases can produce a great deal of audit data to consolidate. Secondly, good practice dictates that audit data should be transmitted to a remote centralized location where it is secure from tampering by the individuals whose activities are being audited. Finally, it is important to have a way to efficiently monitor the ongoing stream of audit data to find the particular events with security implications and identify problems that need immediate attention.

Oracle Audit Vault and Database Firewall (AVDF) simplifies the collection of audit records in a secure repository, creates separation of duty from the databases it is auditing, and produces consolidated reports and alerts on audit event data. AVDF collects audit data not just from Oracle databases, but also from Oracle MySQL, Microsoft SQL Server, SAP Sybase, and IBM DB2 for LUW. It is designed for enterprises that wish to collate information from hundreds of database instances and provide unified policy management and event propagation from a single platform.

AVDF has four main components:

- Audit Vault (AV) Server, which is the central repository of audit records (unsurprisingly, held in an Oracle database!). It includes Oracle technologies such as compression, partitioning, encryption, and privileged user controls. AV Server performs four primary roles:
    - Consolidation of audit data and event logs from Firewall, Oracle and non-Oracle databases, operating systems, directories, and other custom sources. Audit Vault can also be configured to encrypt data during transmission.
    - Management of the audit policies

    - Alerting and managing out-of-policy transactions
    - Reporting and distribution of ad-hoc or scheduled reports

- Audit Vault Agents, which are low-impact host-based programs that manage the collection of data from systems where they are installed and communicate with the AV server.
- Audit trails, the sources of audit data. There are five kinds of audit trails:
    - Database audit trails including Oracle, Microsoft SQL Server, SAP Sybase, IBM DB2 for LUW, and MySQL. These can be audit tables, audit files, or REDO records.
    - OS audit trails (Linux, Windows, Solaris)
    - Directory services such as Microsoft Active Directory
    - File systems such as Oracle ACFS
    - Custom audit data in either database tables or XML files

Database Firewall, the network component of AVDF. Database Firewall monitors SQL transactions over the network and can decide whether a SQL statement should be permitted, modified, blocked, or substituted before it reaches the database server. The next chapter describes the operation of Database Firewall; for now it is sufficient to know that it supplies audit data about its activity to Audit Vault. Figure 4-1 shows the deployment of Oracle Audit Vault and Database Firewall with many sources of audit data in a heterogeneous network environment.

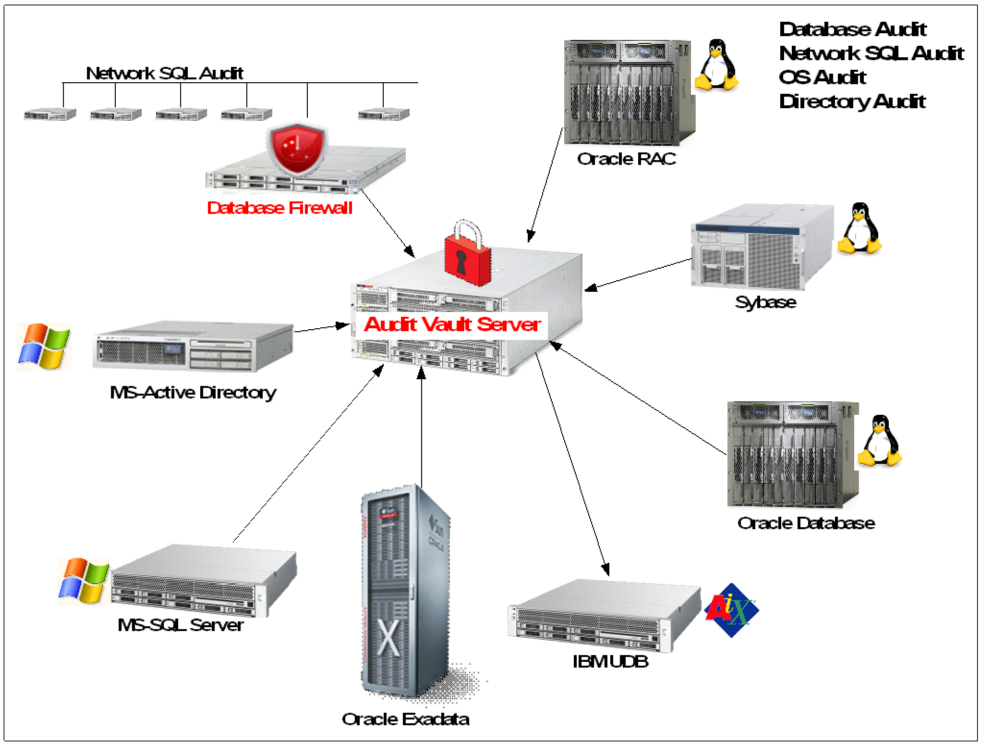
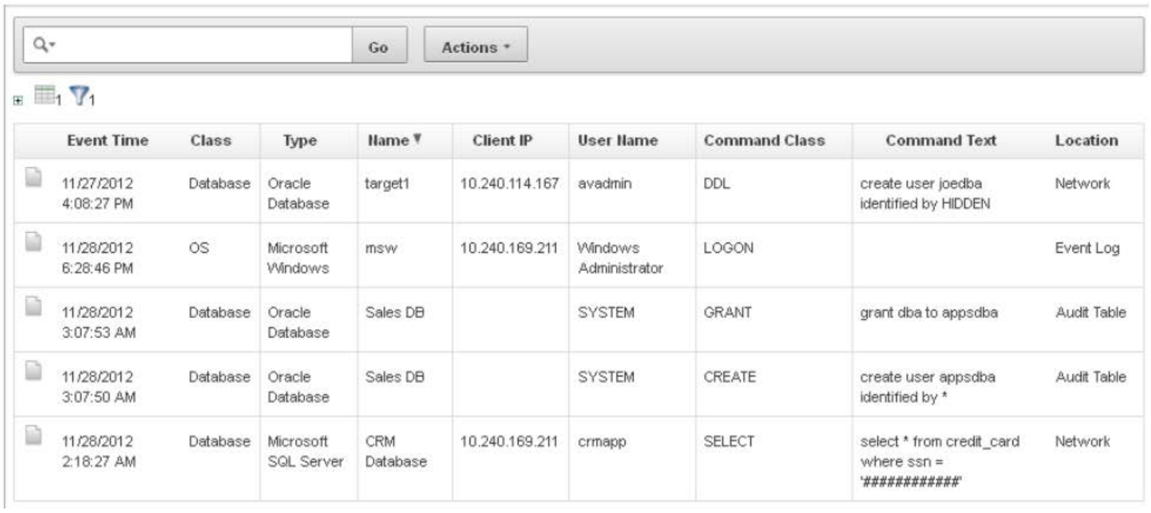


Figure 4-1: Oracle Audit Vault and Database Firewall architecture

Reporting and Alerting

The Audit Vault Server has comprehensive reporting providing a selection of standard security reports, ad-hoc reports, and forensic reports. There are also out-of-the-box audit assessment reports designed to help meet the requirements of standards including PCI-DSS, GLBA, HIPAA, SOX, and DPA.

Reports like the one in Figure 4-2 can be used to monitor a wide range of activity including privileged user activity on the database server, changes to database structures, and data on inbound SQL statements on the network. Reports can display consolidated audit information from databases, operating systems, and directories, providing a holistic picture of activities across the enterprise.



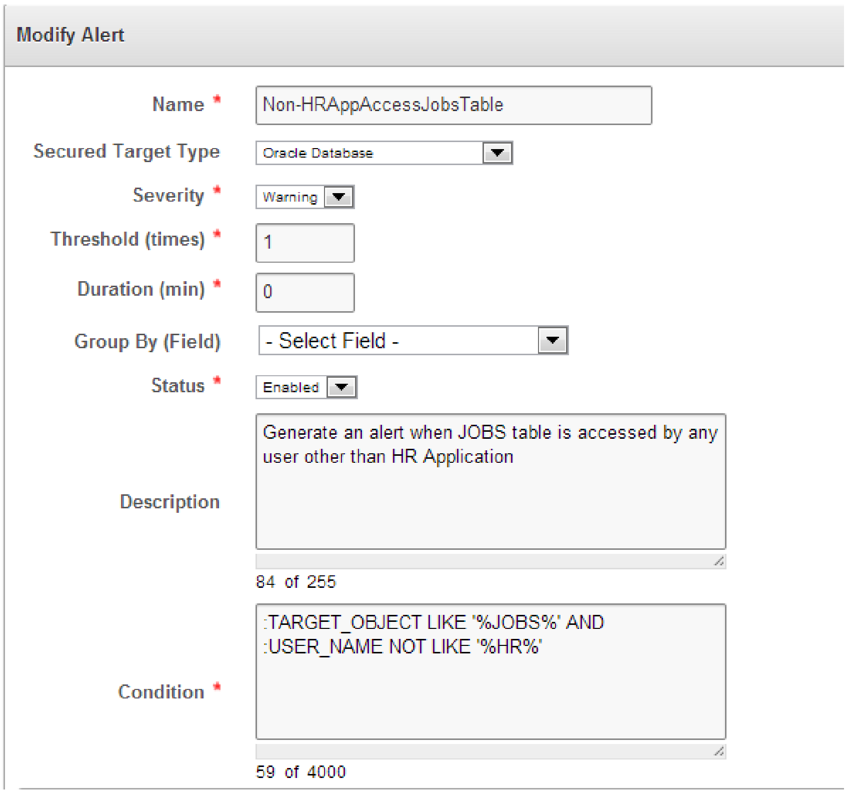
	Event Time	Class	Type	Name	Client IP	User Name	Command Class	Command Text	Location
	11/27/2012 4:08:27 PM	Database	Oracle Database	target1	10.240.114.167	avadmin	DDL	create user joedba identified by HIDDEN	Network
	11/28/2012 6:28:46 PM	OS	Microsoft Windows	msw	10.240.169.211	Windows Administrator	LOGON		Event Log
	11/28/2012 3:07:53 AM	Database	Oracle Database	Sales DB		SYSTEM	GRANT	grant dba to appsdba	Audit Table
	11/28/2012 3:07:50 AM	Database	Oracle Database	Sales DB		SYSTEM	CREATE	create user appsdba identified by *	Audit Table
	11/28/2012 2:18:27 AM	Database	Microsoft SQL Server	CRM Database	10.240.169.211	crmapp	SELECT	select * from credit_card where ssn = '###########'	Network

Figure 4-2: Consolidation from network, database audit, and OS event logs

An auditor can access reports interactively through a console Web interface or through PDF or XLS files and automatically distribute them to different organizational teams. Rules can automatically highlight specific rows for you to quickly spot suspicious or unauthorized activity. Oracle BI Publisher can be used to create new or customized PDF and XLS report templates to meet specific security needs.

Reporting takes place against a highly optimized Oracle 11*g*R2 database, which also can be configured to allow external reporting tools to access audit data. This means that existing Management Information (MI) tools or even Excel spreadsheets can access important security information, but the ability to change or delete data is strictly prohibited to retain the integrity of audit records. Furthermore, the Audit Vault repository schema is documented to enable integration with third-party reporting solutions.

Alert or Event Management lets you create complex alert definitions that will allow alerts to be raised on the Auditor console dashboard and notifications to be distributed to multiple users for attestation. Alerts can be defined specifying a Boolean condition using SQL comparison operators (=, <, LIKE, IN, NULL, and so on) and logical operations (NOT, AND, OR). Furthermore, groups (using parentheses) and wildcards (%, _) can be applied to give breadth to generalized alert conditions. For example, you may have a table JOBS containing sensitive data that should only be accessed by an authorized account such as `HR_App`. The example in Figure 4-3 shows that an alert will be raised if the JOBS table is accessed by any user that is not `HR_App`.



Modify Alert

Name *	Non-HRAppAccessJobsTable
Secured Target Type	Oracle Database
Severity *	Warning
Threshold (times) *	1
Duration (min) *	0
Group By (Field)	- Select Field -
Status *	Enabled
Description	Generate an alert when JOBS table is accessed by any user other than HR Application 84 of 255
Condition *	:TARGET_OBJECT LIKE '%JOBS%' AND :USER_NAME NOT LIKE '%HR%' 59 of 4000

Figure 4-3: AVDF alerting

CHAPTER 5

Controlling SQL Input

Studies conducted by government and academic institutions have concluded that a large percentage of data breaches are perpetrated using SQL injection or misused credentials of insiders who have authorized access to the system and its data. Most applications today operate using a single user account for communicating with the database, and many do not validate their input sufficiently. This application architecture, combined with the increasing number of attacks on databases via SQL injection or insiders with access to privileged accounts, has made database activity monitoring an important component of the overall security architecture.

Most monitoring solutions on the market today rely on regular expressions within their policies to determine which SQL statements should be blocked from reaching the database. The challenge with these first-generation solutions is that regular expressions do not match the expressive power of the SQL language. Because there are many different ways to write a SQL statement that will have some harmful effect, it is nearly impossible to write a regular expression rule that will detect all such statements.

Even if it were feasible to match the SQL statements using regular expressions, the set of harmful statements does not remain constant. Instead of blocking a fixed set of “bad” statements, it is much more effective to allow only “good” statements based on the normal activity of the applications and users that connect to the database. Effective monitoring of SQL input to the database can block or raise alerts for attempted policy violations and provide comprehensive reports about database activity for compliance purposes.

Oracle Audit Vault and Database Firewall

Oracle Audit Vault and Database Firewall provides a first line of defense for Oracle and non-Oracle databases and consolidates audit data from databases, operating systems, and directories. The Database Firewall component of the solution incorporates a highly accurate SQL grammar-based engine to monitor and block unauthorized SQL traffic before it reaches the database. Unlike solutions that leverage regular expressions, Database Firewall parses the SQL itself to achieve the level of accuracy required for enterprise database monitoring.

Database Firewall collects the SQL request from the network using highly optimized traffic capture techniques. The SQL query is then associated with as much session information as possible. This includes the grammatical structure of the query and context attributes such as:

• Database user name	(for example, dba_001)
• OS user name	(for example, oracle_dom1\fred.bloggs)
• Client program name	(for example, sqlplus.exe)
• Client IP address	(for example, 192.0.2.12)
• Time of transaction	(millisecond accuracy)
• Secured target name	(for example, db11gr2.internal.oracle.com)
• Database service name	(for example, accounts.service1)

This information is then combined with further analysis of the SQL statement, including

• SQL category	(for example, SELECT, DML-write, DDL, DCL, TCL, and so on)
• Table name	(for example, “tbl_customers”)

All of this work is completed in real time as network packets arrive and a comparison of the request against Database Firewall policy can then take place. Database Firewall groups SQL requests into clusters of queries with the same grammatical sense, enabling the distillation of hundreds of millions of SQL queries down to just a few hundred, which in turn can be subcategorized by IP addresses, client programs, SQL types, or user names. This process of distilling millions of seemingly unsystematic SQL queries to orders-of-magnitude fewer clusters of activity gives Database Firewall the ability to distinguish the “normal” transaction from the “abnormal” transaction.

Whitelists with Database Firewall

Database Firewall automatically analyzes all SQL traffic and provides a policy manager to quickly set up whitelists of “normal” behavior for applications and users who generally perform the same type of SQL interaction. Whitelist policies are created based on SQL requests that have been seen from attributes that are definitively and uniquely associated with an application enabling selective and effective monitoring. Typical examples would be:

- Application IP addresses (for example, 192.0.2.100, 192.0.2.101, 192.0.2.102)
- Application program names (for example, service_123abff00, service_123ccdd00)
- Application DB users (for example, db_appuser, db_appadmin)

Blacklists with Database Firewall

In addition to the positive security enforcement model based on whitelists, Database Firewall also supports blacklists. As with whitelist policies, blacklist policies can evaluate various factors such as user name, IP address, time of day, and program, before making the decision. Blacklists act as exceptions to the positive enforcement model and allow custom bypass policies to be created for specific events. For example, exception list policies could be used to enable a specific remote administrator coming from a predetermined IP address to diagnose a particular application performance issue without being bound by the whitelist or the blacklist. Blacklists can be thought of as exceptions that evaluate various factors. Blacklists are similar to “allow” and “deny” settings in traditional firewalls and are made up of:

- IP sets
- DB user sets
- OS user sets
- Client program sets

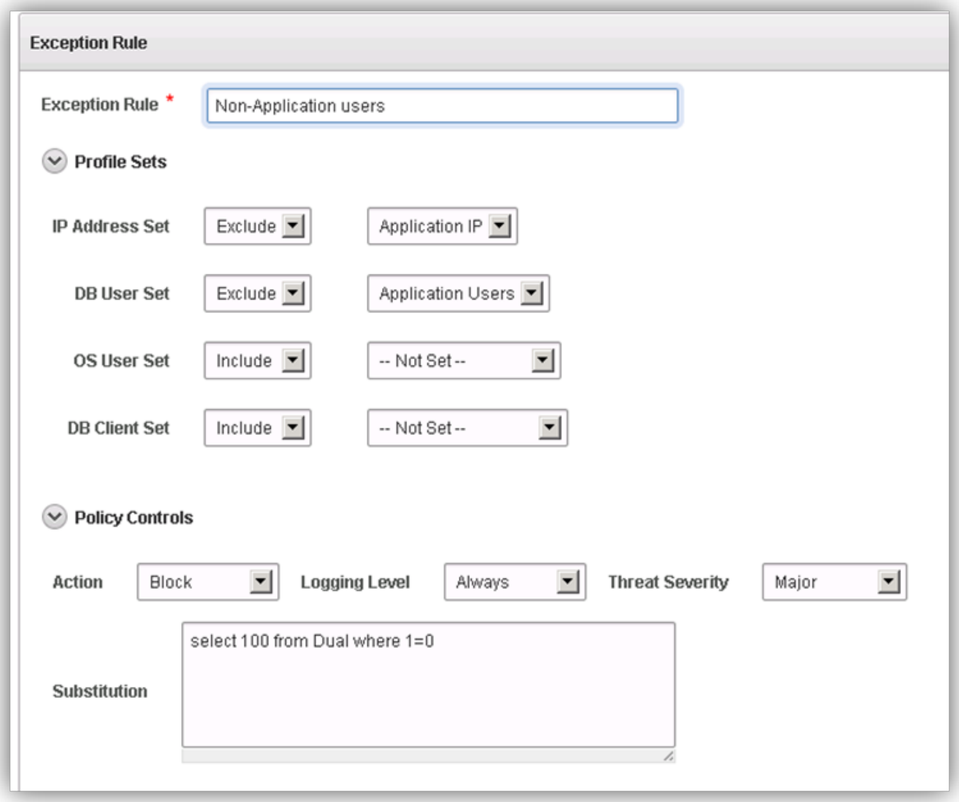
A “set” is a grouping of like attributes into a named set. For instance, four sets could be created with groups of users as shown in the following table:

Set 1 **Reporting Users**	**Set 2** **DBA Users**	**Set 3** **Application Users**	**Set 4** **Consultant Users**
marketing_01	mikesmith	peoplesoft_201	jennafortes
marketing_02	williesores	sap_user	louisegood
marketing_03	davejones	siebel_4501	cons_001
marketing_04	johnasher	app_001	cons_002
mgt_BI_rep		app_002	

A blacklist policy can be set against each of these sets. For instance:

Reporting Users	Pass, Log all
DBA Users	Pass, Alert
Application Users	Pass
Consultant Users	Block, Log all

Exceptions provide a powerful means of filtering SQL requests at a high level, simplifying setting of policies for applications, subnets, groups, or users. The following illustration shows an example exception policy. This policy will allow traffic from the account application on the secured target, but will block traffic from any other source or identity.



Exception Rule

Exception Rule *	Non-Application users	
Profile Sets		
IP Address Set	Exclude	Application IP
DB User Set	Exclude	Application Users
OS User Set	Include	-- Not Set --
DB Client Set	Include	-- Not Set --
Policy Controls		
Action	Block	
Logging Level	Always	
Threat Severity	Major	
Substitution	select 100 from Dual where 1=0	

Novelty Rules with Database Firewall

Novelty rules are used to control SQL input for high-level categories of SQL and for database tables. A novelty rule configuration can act on various categories, depending on the policy required for security. The categories include:

- **Data Manipulation** (`INSERT`, `UPDATE`, `DELETE`, `SELECT INTO`, and so on)
- **Data Manipulation Read-Only** (`SELECT`)
- **Procedural** (stored procedures or RPC commands)
- **Data Definition** (`CREATE`, `DROP`, `ALTER`, and so on)
- **Data Control Language** (`GRANT`, `REVOKE`, and so on)
- **Composite** (commands that are executed in a transaction)

- **Transaction** (`COMMIT`, `ROLLBACK`, and so on)
- **Composite with Transaction** (a DML statement with a TCL command, and so on)

Novelty rules can combine SQL commands with tables, providing an easy way to disable logging for entire classes of tables such as the underlying database dictionary tables that are not related to the application itself. Novelty rules can be used to prevent or allow actions against different tables. Novelty rules are often used for controlling behavior of DBAs over the network where it might be necessary to stop them from accessing specific application tables.

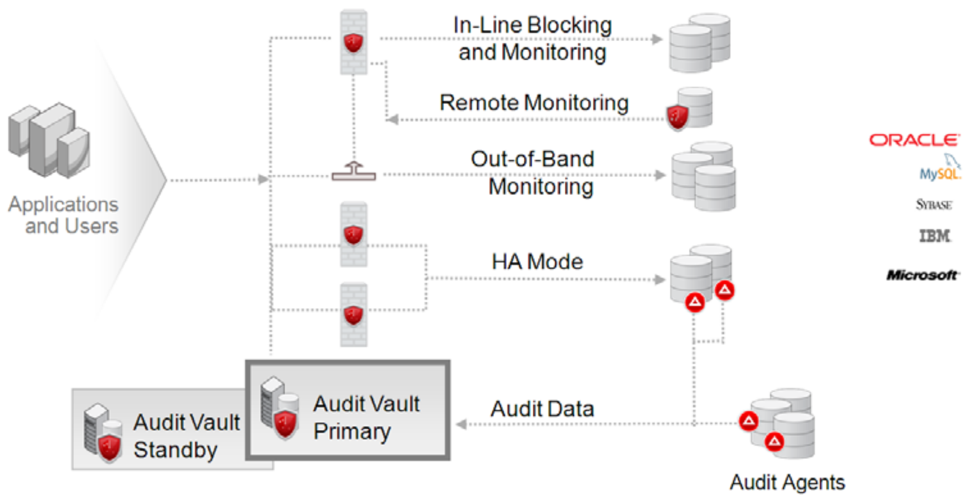
Handling Unauthorized SQL with Database Firewall

When Oracle Database Firewall finds an unauthorized statement, it handles the statement in one of the following ways:

- Alert on all out-of-policy SQL statements.
- Modify the request using SQL statement substitution by replacing an out-of-policy statement with a new harmless statement that does not return any data.
- Terminate the connection: This blocks all traffic from that specific database connection to go to the server as it kills the connection. This is the most aggressive action, and if the application is using connection pooling, this will impact all the users using the pool.
- Block the SQL statement: This specific statement is stopped from reaching the database server. The actual end-user experience would depend on how the application handles this case where the server does not respond. Client connection to the database can be configured to either continue or terminate.

Database Firewall Network Deployment

Database Firewall can be deployed as a transparent network bridge, simply inserted into the network in a segment that lies between database clients/application servers and the databases being protected, as shown in the following illustration. This “in-line” bridge architecture requires no configuration changes to database clients, applications, or the database itself, and provides the flexibility for both active and passive monitoring. To passively monitor the database activity, it is also possible to forward the traffic to the database firewall using the span-port.



In scenarios where it is difficult to add a network bridge, or if the database servers are in remote places, Database Firewall can also be configured as a proxy such that all traffic to the database server is routed through the database firewall. This requires the Database Server IP address/port on the database client or application to be changed to the IP address/port for the Database Firewall proxy, along with changes to the database listener to reject direct connections. Most enterprise network switches and traditional firewalls can also be used to redirect database traffic to a Database Firewall proxy port, allowing SQL traffic to be protected without any changes to database clients or applications. A given database firewall can operate as a transparent bridge for some databases and a proxy for others.

Database Firewall supports a local server-side, monitor-only agent to ensure flexibility in the choice of the network point at which the traffic is monitored. Host Monitor, part of the Audit Agent, captures SQL traffic reaching the database server and securely forwards it to Database Firewall. It can be used to remotely monitor database servers running on Linux and Windows platforms.

CHAPTER 6

Masking Sensitive Data

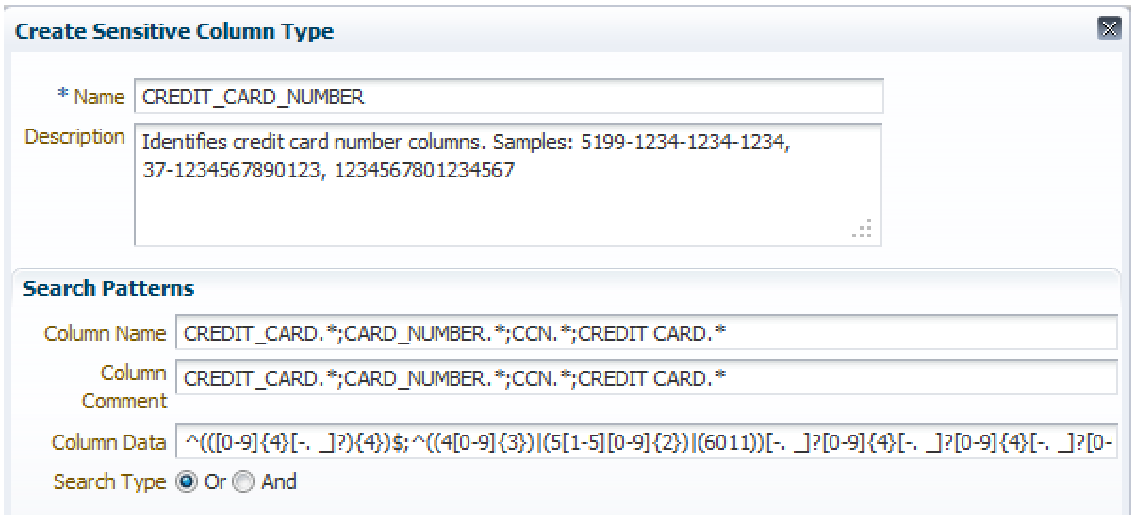
Reducing the exposure of sensitive data is a challenge faced by many organizations. Many organizations need realistic data for test and development systems, but they find it impossible to protect these systems to the same extent as their production systems. As a result, sensitive and regulated data in nonproduction environments has increasingly become the target of attackers. Masking data before it is moved from production environments eliminates the risk of data breaches in nonproduction environments by irreversibly replacing the original sensitive data with fictitious data so that it can be safely shared with IT developers or business partners.

At the same time, the overexposure of sensitive data in production applications is also a challenge faced by many organizations. While displaying sensitive data to application users may have been common a few years ago, organizations today must redact sensitive data shown on application screens to better safeguard the data as well as adhere to local and international regulations. However, this type of redaction inside application code can be complex and error-prone, especially in a consolidated environment where multiple applications access the same data. The best and most cost-effective approach to controlling the exposure of sensitive data in applications is by enforcing policies inside the production database so that the policies will be enforced regardless of the application accessing the data.

While masking of data physically alters the stored data so that it can be handed off to other organizations, redaction of data does not physically alter the stored data; it simply transforms the data before it leaves the database and is displayed in the application. In many cases, redaction will only be applied to part of the data, such as the first twelve digits of the credit card, or the username portion of an e-mail address before the domain name.

Sensitive Data Discovery

Knowing where your sensitive data resides is the first step in deciding how to protect it. A surprisingly large number of organizations have difficulty locating and identifying all of their sensitive data due to the complexity and size of large applications. Using the Data Discovery and Modeling capability in Oracle Enterprise Manager, enterprises can examine the metadata associated with the applications inside the database, such as column names and data types, and sample existing data to help identify the sensitive data. For example, Oracle Enterprise Manager can help locate credit card and U.S. Social Security numbers based on the column name, column comments, or characteristics of the data itself. The following example shows data patterns for credit card numbers.



Create Sensitive Column Type

* Name	CREDIT_CARD_NUMBER
Description	Identifies credit card number columns. Samples: 5199-1234-1234-1234, 37-1234567890123, 1234567801234567

Search Patterns

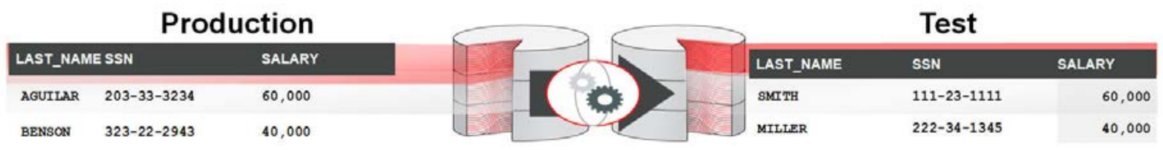
Column Name	CREDIT_CARD.*;CARD_NUMBER.*;CCN.*;CREDIT CARD.*
Column Comment	CREDIT_CARD.*;CARD_NUMBER.*;CCN.*;CREDIT CARD.*
Column Data	^(([0-9]{4}[-. _]?){4})$;^((4[0-9]{3})|(5[1-5][0-9]{2})|(6011))[-. _]?[0-9]{4}[-. _]?[0-9]{4}[-. _]?[0-
Search Type	[✓ Or] And

Data Discovery and Modeling also analyzes the relationships between application objects using the foreign key constraints defined inside the database. This information about the data types and relationships for an application is referred to as the Application Data Model (ADM) and is stored in the Oracle Enterprise Manager repository. Relationships can be manually defined inside the ADM for applications that do not use database-enforced referential integrity.

Oracle has created Application Accelerators for both Oracle Fusion Applications and Oracle E-Business Suite. The Application Accelerators list the sensitive data for each of the applications. Oracle Data Masking uses the Application Accelerators to facilitate masking of data from production databases to test and development environments.

Data Masking

With Oracle Data Masking, Oracle provides end-to-end automation for provisioning test databases from production in compliance with regulations, as the following diagram shows. Sensitive information such as credit card or Social Security numbers can be replaced and used for development and testing without expanding the security perimeter. This reduces the number of database systems that need to be monitored for compliance and security.

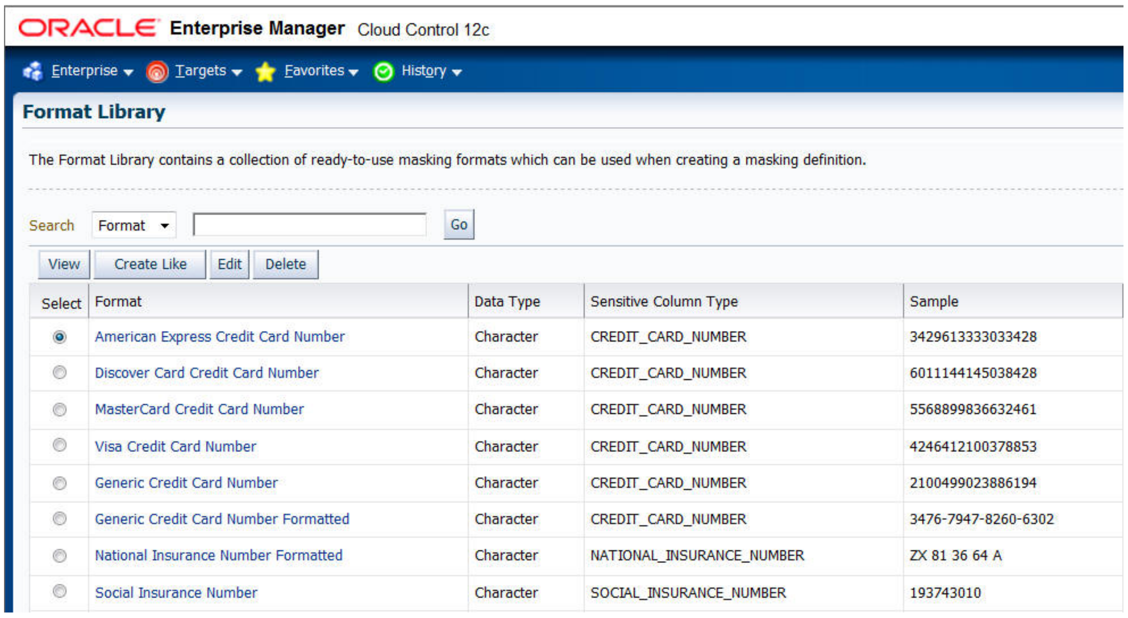


LAST_NAME	SSN	SALARY
AGUILAR	203-33-3234	60,000
BENSON	323-22-2943	40,000

LAST_NAME	SSN	SALARY
SMITH	111-23-1111	60,000
MILLER	222-34-1345	40,000

Masking Format Library

Oracle Data Masking provides an out-of-the-box library with mask formats for the most common types of sensitive data. Formats in the library include sensitive data such as credit card numbers, telephone numbers, Social Security numbers, and national identifiers. By leveraging the Format Library in Oracle Data Masking, enterprises can apply data privacy rules to sensitive data across enterprise-wide databases and ensure consistent compliance with regulations. Enterprises can also extend this library with their own mask formats to meet their specific data privacy and application requirements. Once the work of associating masking definitions with application attributes is complete, the formats and data associations can be saved in the Application Data Model and re-executed when test, development, or partners need a refresh of data. Oracle Data Masking Pack can mask data in heterogeneous databases, such as IBM DB2 and Microsoft SQL Server, through the use of Oracle Database Gateways.



Format Library

The Format Library contains a collection of ready-to-use masking formats which can be used when creating a masking definition.

Search Format Go

View | Create Like | Edit | Delete

Select	Format	Data Type	Sensitive Column Type	Sample
●	American Express Credit Card Number	Character	CREDIT_CARD_NUMBER	3429613333033428
○	Discover Card Credit Card Number	Character	CREDIT_CARD_NUMBER	6011144145038428
○	MasterCard Credit Card Number	Character	CREDIT_CARD_NUMBER	5568899836632461
○	Visa Credit Card Number	Character	CREDIT_CARD_NUMBER	4246412100378853
○	Generic Credit Card Number	Character	CREDIT_CARD_NUMBER	2100499023886194
○	Generic Credit Card Number Formatted	Character	CREDIT_CARD_NUMBER	3476-7947-8260-6302
○	National Insurance Number Formatted	Character	NATIONAL_INSURANCE_NUMBER	ZX 81 36 64 A
○	Social Insurance Number	Character	SOCIAL_INSURANCE_NUMBER	193743010

The Format Library (shown in the preceding illustration) simplifies the masking process, enabling cost-effective compliance. The library is extensible and can be customized to meet sophisticated business requirements such as generating e-mail addresses based on a fictitious set of first and last names.

Powerful Masking Techniques

Oracle Data Masking supports sophisticated masking techniques, enabling complex applications to function with masked data in test and development environments.

Condition-based masking	Apply different masking formats to different rows based on a condition. For example, data about citizens from multiple countries stored in the same table would have their unique ID (SSN, National Identifier) generated differently.
Compound masking	Mask related data stored in multiple columns as a group. For example, city, state, and zip code need to be masked together.
Deterministic masking	Enables data to be masked to a consistent value across multiple databases or applications. This is used to ensure that certain values (for example, customer number) get masked to the same value across all databases.
Key-based reversible masking	Enables data to be masked using a key-based reversible function. Data can be unmasked using the original key.

Cloned and At-Source Data Masking

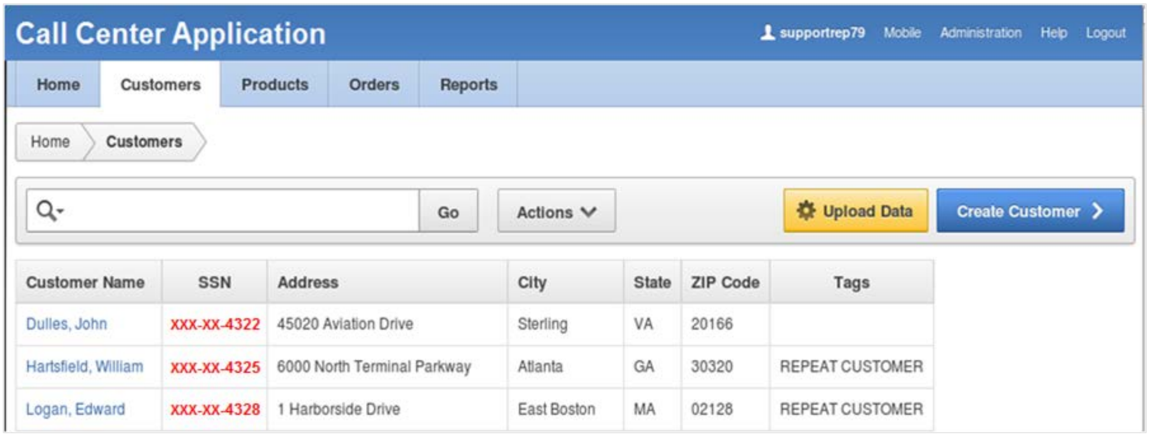
Oracle Data Masking supports masking data in a database cloned from the production environment or at-source. With the cloned database, Oracle Data Masking builds a worklist of the tables and columns chosen for masking and physically alters the stored data. During the masking process, Oracle Data Masking uses various optimizations so that objects not required for masking are not touched and tables that are touched are only touched once. Tables with primary keys are typically masked first. In addition, Oracle Data Masking uses the `NOLOGGING` option when creating the masked tables so that excessive REDO logs are not generated.

Oracle Data Masking now provides At-Source Masking, where the data is masked during the Oracle data export process. The resulting masked physical export files can then be directly shared with test and development organizations without the risk of breaching sensitive data. The new approach also enables data such as documents or images to be set to a fixed string during the export masking process. This reduces the size of the data set that needs to be transferred to other organizations.

Data Redaction

In addition to masking sensitive data before exporting it from the database, there is an increasing need to control the display of sensitive data within applications. For example, take a call center application with a screen that exposes customer credit card information and personally identifiable information to call center operators. Exposure of that information, even to legitimate application users, may violate privacy regulations, put the data at unnecessary risk, and quite simply expose the information to those without a true need to know. Traditional approaches to redacting sensitive data typically rely on custom application logic and result in disparate solutions that are inconsistent across the enterprise and costly to maintain over their lifetime. In addition, strict controls must be placed on new application development to make sure that custom application code and new objects are properly accessed.

Oracle Advanced Security Data Redaction is the selective, on-the-fly redaction or scrubbing of sensitive data in query results prior to display by applications, as shown in the following illustration. It enables consistent redaction of database columns across application modules accessing the same data. Data Redaction minimizes changes to applications because it does not alter actual data in internal database buffers, caches, or storage, and it preserves the original data type and formatting when transformed data is returned to the application.

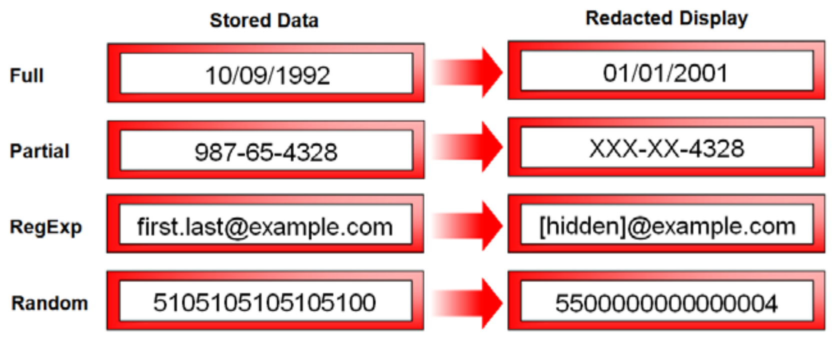


Customer Name	SSN	Address	City	State	ZIP Code	Tags
Dulles, John	XXX-XX-4322	45020 Aviation Drive	Sterling	VA	20166	
Hartsfield, William	XXX-XX-4325	6000 North Terminal Parkway	Atlanta	GA	30320	REPEAT CUSTOMER
Logan, Edward	XXX-XX-4328	1 Harborside Drive	East Boston	MA	02128	REPEAT CUSTOMER

Unlike historical approaches that relied on application coding and new software components, Data Redaction policies are enforced directly in the Oracle database kernel, providing consistency across application modules. Redaction can be conditional, based on different factors that are tracked by the database or passed to the database by applications such as user identifiers, application identifiers, or client IP addresses. A redaction format library provides preconfigured column templates to choose from for common types of sensitive information such as credit card numbers and national identification numbers. Once enabled, polices are enforced immediately, even for active sessions.

Data Redaction Policies and Transformations

Data Redaction supports a number of different transformations that can redact all data in specified columns, preserve certain pieces of the data, or randomly generate replacement data. Examples of the supported data transformations are shown in the following illustration.

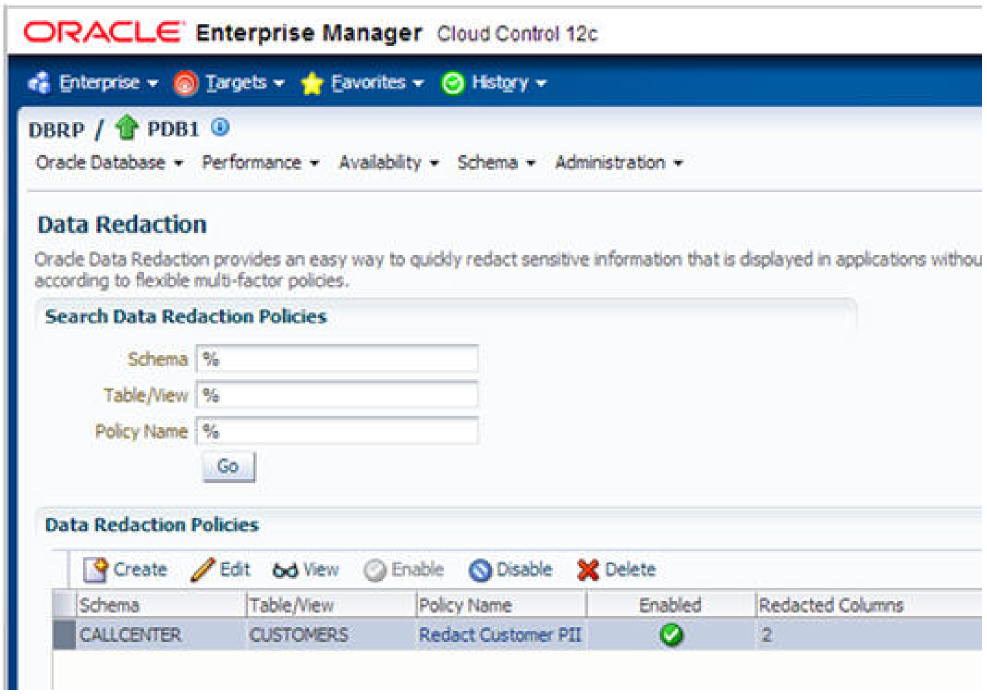


Data Redaction can be applied selectively, based on declarative policy conditions that utilize the runtime contexts available from the database and from the running application. Examples include user identifiers, user roles, and client IP addresses. Context information available from Oracle Application Express (APEX), Oracle Real Application Security, and Oracle Label Security also can be utilized. Redacting APEX applications is straightforward because policy conditions can use the application users and application identifiers that APEX automatically tracks. Multiple runtime conditions can be joined together within a Data Redaction policy for fine-grained control over when redaction occurs. Redaction policies are stored and managed inside the database, and they go into effect immediately upon being enabled.

On the surface, Data Redaction and Oracle Data Masking seem quite similar. However, there is an important difference in that Oracle Data Redaction does not physically alter the stored data. As a result, Oracle Data Redaction supports a subset of the transformations available with Oracle Data Masking. The power of Data Redaction resides in its superior performance, enforcement inside the database kernel, and the declarative policy conditions.

Deploying Data Redaction

Data Redaction can be deployed for existing applications quickly using either Oracle Enterprise Manager or a PL/SQL procedure to specify the protected columns, transformation types, and conditions. Oracle Enterprise Manager provides an easy-to-use interface for creating and applying redaction policies, as shown in the following illustration.


ORACLE Enterprise Manager Cloud Control 12c
Enterprise ▾ Targets ▾ Favorites ▾ History ▾
DBRP / PDB1
Oracle Database ▾ Performance ▾ Availability ▾ Schema ▾ Administration ▾

Data Redaction

Oracle Data Redaction provides an easy way to quickly redact sensitive information that is displayed in applications withou according to flexible multi-factor policies.

Search Data Redaction Policies

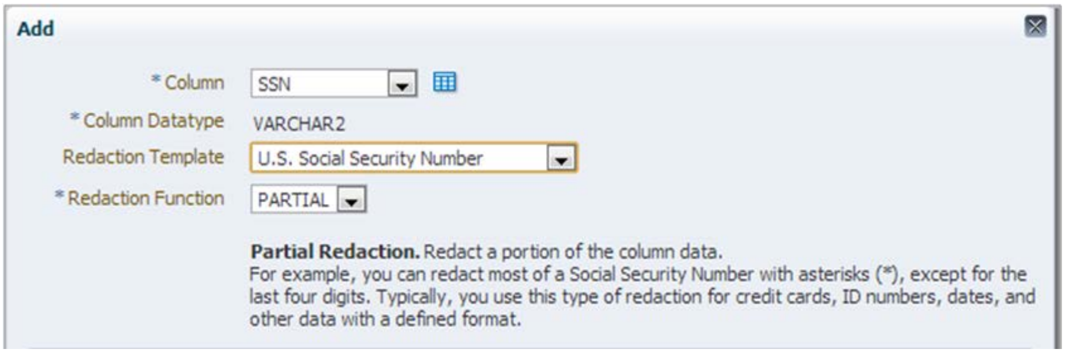
Schema	%
Table/View	%
Policy Name	%

Data Redaction Policies

Create Edit View Enable Disable Delete

Schema	Table/View	Policy Name	Enabled	Redacted Columns
CALLCENTER	CUSTOMERS	Redact Customer PII	✓	2

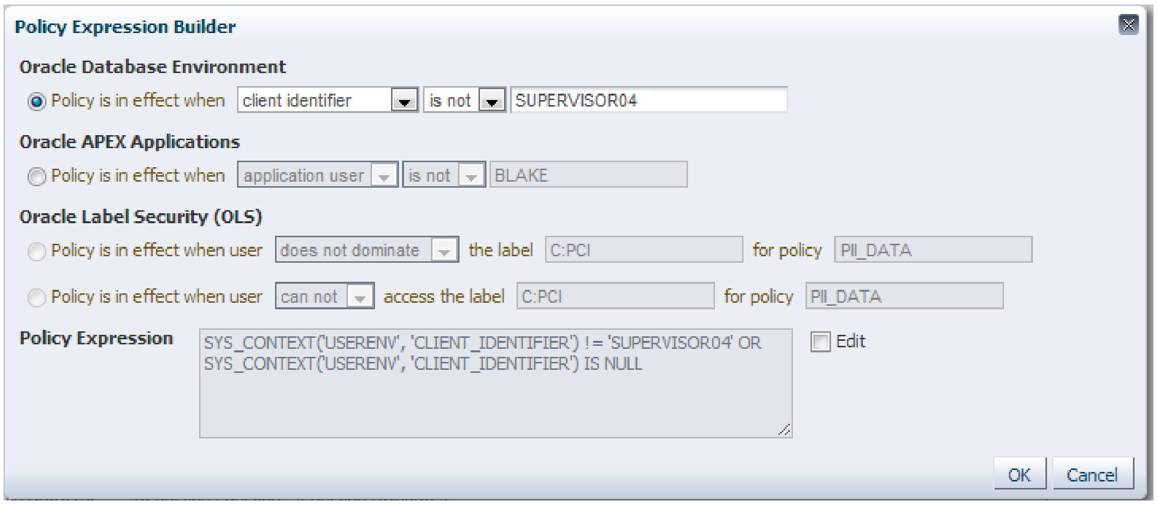
Predefined column templates are available in Oracle Enterprise Manager for redacting common sensitive data such as credit card numbers and U.S. Social Security numbers, as shown in the following illustration. Enterprise Manager Sensitive Data Discovery assists in locating columns to be redacted inside complex application schemas.


Add

* Column	SSN
* Column Datatype	VARCHAR2
Redaction Template	U.S. Social Security Number
* Redaction Function	PARTIAL

Partial Redaction. Redact a portion of the column data. For example, you can redact most of a Social Security Number with asterisks (*), except for the last four digits. Typically, you use this type of redaction for credit cards, ID numbers, dates, and other data with a defined format.

The Policy Expression Builder within Oracle Enterprise Manager enables administrators to define and apply redaction policies on existing applications. As shown in the following illustration, the Policy Expression Builder dialog guides the user through creating policy conditions that use context obtained from applications, the database, the APEX framework, and other database security solutions.


Policy Expression Builder

Oracle Database Environment

● Policy is in effect when client identifier is not SUPERVISOR04

Oracle APEX Applications

○ Policy is in effect when application user is not BLAKE

Oracle Label Security (OLS)

○ Policy is in effect when user does not dominate the label C:PCI for policy PII_DATA

○ Policy is in effect when user can not access the label C:PCI for policy PII_DATA

Policy Expression: SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') != 'SUPERVISOR04' OR SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') IS NULL

☐ Edit

In addition to the Oracle Enterprise Manager administrative interface, an API inside the database can be used for scripting and applying redaction policies.

```
DBMS_REDACT.ADD_POLICY(
  object_schema => 'CALLCENTER',
  object_name   => 'CUSTOMERS',
  policy_name   => 'Redact Customer PII',
  expression    => 'SYS_CONTEXT(''USERENV'',''CLIENT_IDENTIFIER'') !=
''SUPERVISOR04'' OR
      SYS_CONTEXT(''USERENV'',''CLIENT_IDENTIFIER'') IS NULL',
  column_name   => 'SSN',
  function_type => DBMS_REDACT.PARTIAL,
  function_parameters => 'VVVFVVFVVVV,VVV-VV-VVVV,X,1,5');
```

Another reason why Data Redaction is easy to deploy is its transparency to applications and the database. For application transparency, Data Redaction supports the column data types that are frequently used by applications and various database objects including tables, views, and materialized views. Redacted values retain key characteristics of the original data such as the data type and optional formatting characters. For transparency to the database, Data Redaction does not affect administrative tasks such as data movement using Oracle Data Pump or database backup and restore using Oracle Recovery Manager. It does not interfere with database cluster configurations such as Oracle Real Application Clusters, Oracle Active Data Guard, and Oracle GoldenGate.

CHAPTER 7

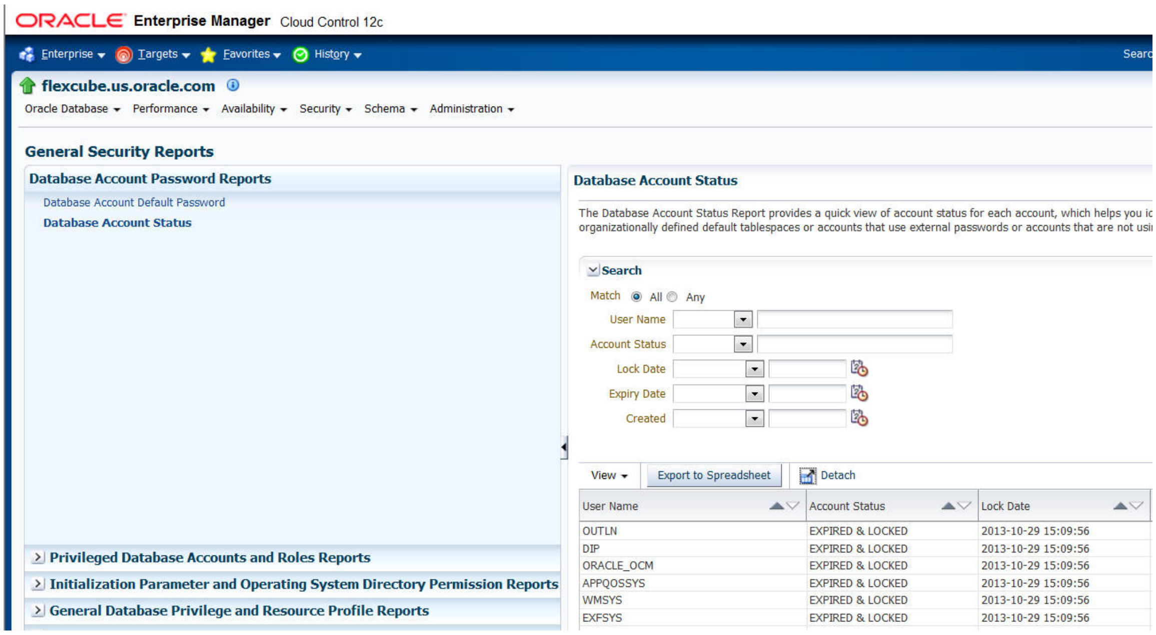
Validating Configuration Compliance

Today's databases have become very sophisticated with many configuration settings. Attackers target vulnerabilities in configuration to gain unauthorized access to databases. For example, well-known username and password combinations in the databases present an easy target. As a result, it is important to scan the database to ensure that it is securely configured, and to remediate the situation if there is a deviation. Configuration scanning can increase the overall security posture of organizations by flagging configuration mistakes and recommending controls that benefit the overall security architecture.

Security configuration scanning has also become an important part of many regulations such as the Payment Card Industry Data Security Standard (PCI-DSS), Sarbanes-Oxley, and numerous breach notification laws. Various organizations such as the Center for Internet Security (CIS) and U.S. Department of Defense also recommend configuration best practices. The complexity of compliance cannot be understated as new regulations are being released and existing regulations are evolving. As a result, having a solution that can adapt to new regulations is critical.

Security Configuration Scanning with Oracle Enterprise Manager

Oracle Database Lifecycle Management Pack provides numerous out-of-the-box reports for basic configuration checks as well as a comprehensive compliance framework. For example, the account status report shows all database accounts and their status. An additional report shows database accounts that have default passwords, matching known passwords for default accounts shipped with various Oracle products. Additional reports include information on initialization parameters, operating system directory permissions, user account profiles, and sensitive object reports. These out-of-the-box reports, shown in Figure 7-1, are accessible from the Reports item on the Security menu.


ORACLE Enterprise Manager Cloud Control 12c
Enterprise
Targets
Favorites
History
Searc
flexcube.us.oracle.com
Oracle Database
Performance
Availability
Security
Schema
Administration
General Security Reports
Database Account Password Reports
Database Account Default Password
Database Account Status
Privileged Database Accounts and Roles Reports
Initialization Parameter and Operating System Directory Permission Reports
General Database Privilege and Resource Profile Reports
View
Export to Spreadsheet
Detach

Database Account Status

The Database Account Status Report provides a quick view of account status for each account, which helps you ic
organizationally defined default tablespaces or accounts that use external passwords or accounts that are not usi

Search

Match [✓ All] Any

User Name

Account Status

Lock Date

Expiry Date

Created

User Name	Account Status	Lock Date
OUTLN	EXPIRED & LOCKED	2013-10-29 15:09:56
DIP	EXPIRED & LOCKED	2013-10-29 15:09:56
ORACLE_OCM	EXPIRED & LOCKED	2013-10-29 15:09:56
APPQOSSYS	EXPIRED & LOCKED	2013-10-29 15:09:56
WMSYS	EXPIRED & LOCKED	2013-10-29 15:09:56
EXFSYS	EXPIRED & LOCKED	2013-10-29 15:09:56

Figure 7-1: Oracle Enterprise Manager security configuration reports

Asset Discovery and Grouping with Oracle Enterprise Manager

Oracle Database Lifecycle Management Pack eliminates the need to manually track IT assets including databases. It provides unintrusive network scanning capabilities to discover servers. Once servers have been discovered, they are easily promoted to a managed state, automatically discovering all databases and other applications. This automated discovery simplifies the process of ensuring that all your servers and software are managed, along with assisting in IT infrastructure consolidation and optimization initiatives. Because of the ever-growing number of systems and services that administrators are responsible for, Oracle Enterprise Manager provides a view that includes only those targets you need to monitor. This view is called a group. Groups are user-defined sets of targets logically combined to be managed as one. You can use groups in Oracle Enterprise Manager to monitor and manage different targets collectively, easily perform administrative operations against the targets, and consolidate and monitor your distributed targets as one logical entity. For example, you can define a group called TEST that contains all hosts and database targets within your test environment. From the group's home page, you can easily see the overall status and availability of all the targets in your test group, instead of having to check the status of each individual member. Even if group membership changes, any jobs submitted to the group automatically keep up with group membership.

From a group's home page, whether the group is based on a homogeneous set of targets or a heterogeneous set of targets, such as a business's application, an administrator can

- Easily determine the overall security configuration compliance of all the members in the group and outstanding alerts
- Drill down and analyze the specifics of a particular target
- Easily determine the status of members of the groups through the rollup of alerts, with quick drill-down into alert details

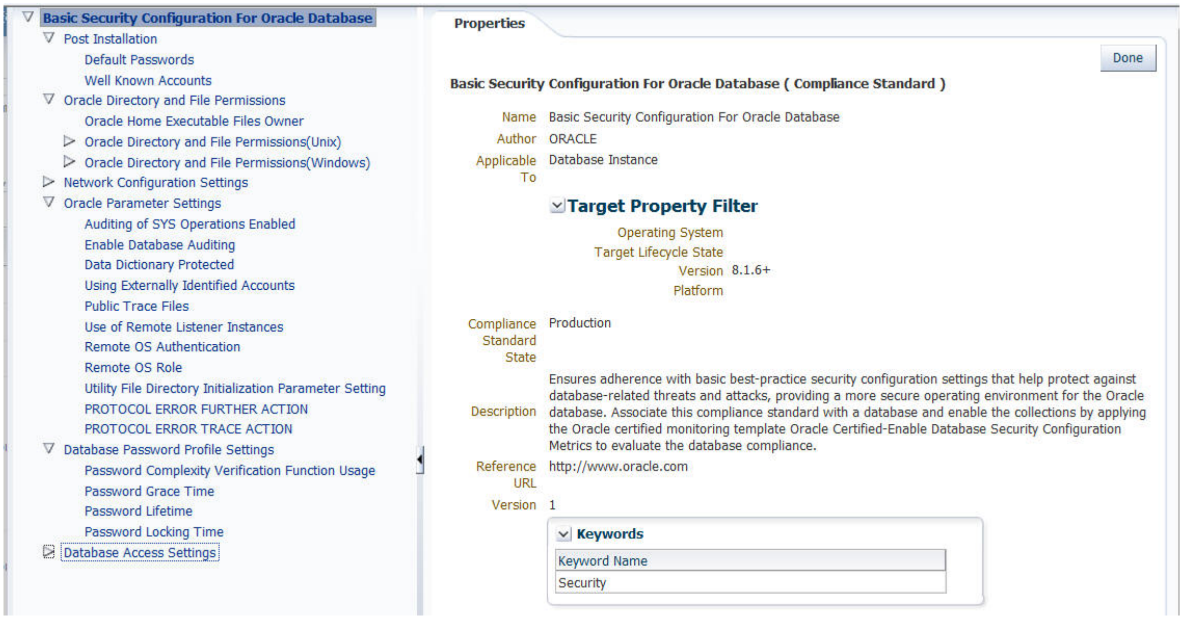
Oracle Enterprise Manager Configuration Scanning

Oracle Database Lifecycle Management Pack provides a powerful compliance scanning solution consisting of compliance frameworks, compliance standards, and compliance standard rules as described in the following table.

Compliance Framework	A compliance framework is an industry-specified best-practices guideline that deals with the underlying IT infrastructure, applications, and processes. Compliance frameworks are hierarchical to allow for direct representation of these industry frameworks. A compliance framework can be used to represent a framework such as PCI.
Compliance Standard	A compliance standard is a collection of checks or rules for a specific target type.
Compliance Standard Rule	A compliance standard rule is a test to determine a specific configuration parameter or status. Oracle supports the following types of rules. • Repository Rules evaluate the metrics collected periodically for a target. • Real-Time Monitoring Rules monitor changes to files, processes, and more.

Configuration Compliance Standards Provided by Oracle

Compliance standards serve as standards by which targets are measured. These standards represent best practices and allow the security administrator to maintain consistency across enterprise systems and configurations. In addition, the trend analysis feature allows fine-grained tracking of compliance scores over time.



- Basic Security Configuration For Oracle Database
  - Post Installation
    - Default Passwords
    - Well Known Accounts
  - Oracle Directory and File Permissions
    - Oracle Home Executable Files Owner
    - Oracle Directory and File Permissions(Unix)
    - Oracle Directory and File Permissions(Windows)
  - Network Configuration Settings
  - Oracle Parameter Settings
    - Auditing of SYS Operations Enabled
    - Enable Database Auditing
    - Data Dictionary Protected
    - Using Externally Identified Accounts
    - Public Trace Files
    - Use of Remote Listener Instances
    - Remote OS Authentication
    - Remote OS Role
    - Utility File Directory Initialization Parameter Setting
    - PROTOCOL ERROR FURTHER ACTION
    - PROTOCOL ERROR TRACE ACTION
  - Database Password Profile Settings
    - Password Complexity Verification Function Usage
    - Password Grace Time
    - Password Lifetime
    - Password Locking Time
  - Database Access Settings

Properties

Done

Basic Security Configuration For Oracle Database (Compliance Standard)

Name	Basic Security Configuration For Oracle Database
Author	ORACLE
Applicable To	Database Instance

Target Property Filter

Operating System	
Target Lifecycle State	
Version	8.1.6+
Platform	

Compliance Standard State	Production
Description	Ensures adherence with basic best-practice security configuration settings that help protect against database-related threats and attacks, providing a more secure operating environment for the Oracle database. Associate this compliance standard with a database and enable the collections by applying the Oracle certified monitoring template Oracle Certified-Enable Database Security Configuration Metrics to evaluate the database compliance.
Reference URL	http://www.oracle.com
Version	1

Keywords

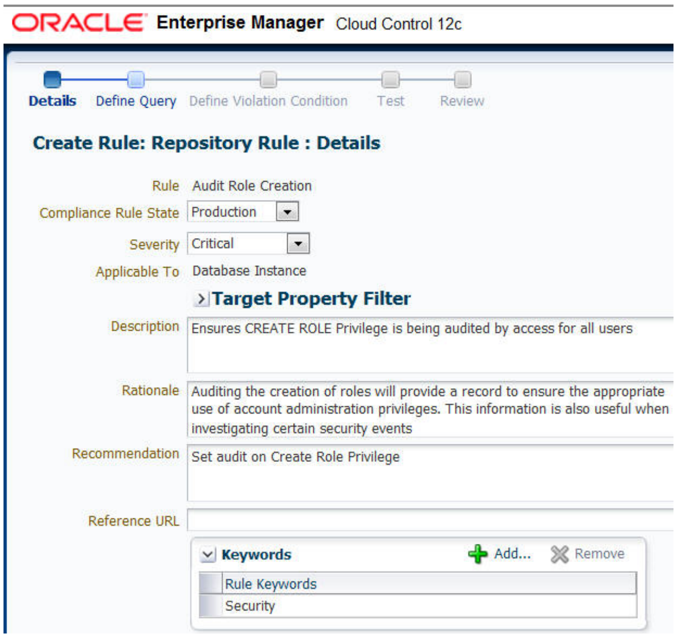
Keyword Name
Security

The preceding illustration shows the Basic Security Configuration Check for Oracle databases and shows the large number of checks that are performed out of the box. Checks performed include analyzing Directory and File Permissions, Oracle Parameter Settings, Database Password Profile Settings, Oracle executable file ownership, and Network Configuration Settings. Oracle Enterprise Manager ships over four dozen out-of-the-box compliance standards, which include:

- Basic Security Configuration for Oracle Cluster Database
- Basic Security Configuration for Oracle Cluster Database Instance
- Basic Security Configuration for Oracle Listener
- Configuration Monitoring for Exadata Compute Node
- Configuration Monitoring for Security Linux Packages

Managing Frameworks, Compliance Standards, and Rules

Oracle Enterprise Manager provides an extensible framework that supports the creation of new frameworks, standards, and rules. This extensibility enables organizations to adapt to the ever-changing world of compliance and also create custom configuration scans to test systems against internal best practices. Compliance standard rules can be created through Oracle Enterprise Manager, as shown in the following illustration. Rules can also be imported and exported in XML format.


ORACLE Enterprise Manager Cloud Control 12c

Details — Define Query — Define Violation Condition — Test — Review

Create Rule: Repository Rule : Details

Rule	Audit Role Creation
Compliance Rule State	Production
Severity	Critical
Applicable To	Database Instance
	Target Property Filter
Description	Ensures CREATE ROLE Privilege is being audited by access for all users
Rationale	Auditing the creation of roles will provide a record to ensure the appropriate use of account administration privileges. This information is also useful when investigating certain security events
Recommendation	Set audit on Create Role Privilege
Reference URL	

Keywords — Add... Remove

Rule Keywords
Security

Event severity is critical to security configuration scanning. Events in Oracle Enterprise Manager can have the following severity levels:

- **Fatal** The monitored target is down.
- **Critical** Immediate action is required in a particular area. The area is either not functional or indicative of imminent problems.
- **Warning** Attention is required in a particular area, but the area is still functional.

- **Advisory** While the particular area does not require immediate attention, caution is recommended regarding the area's current state. This severity is used primarily for compliance standards.
- **Informational** A specific area condition has just occurred.

Real-Time Monitoring with Oracle Enterprise Manager

Oracle Database Lifecycle Management Pack enables real-time monitoring to monitor any action that can happen against a file, a database object, or a Microsoft Windows Registry key. It can also monitor the starting and stopping of processes, and the logging in, logging out, and switching user (su) activity of users. The real-time aspect of the monitoring means that it captures the exact time the action occurred along with the name of the user who performed the action.

Results from real-time monitoring can be reconciled with a Change Management system such as BMC Remedy. This reconciliation can automatically determine if an action was supposed to happen (authorized) or not (unauthorized). Without a Change Management server, this audit status annotation can be made manually through the management console. By reconciling what is happening in the environment to the customer's change management process, real-time monitoring helps to identify out-of-policy actions that will lead to either a high-risk environment, or a compliance control that will fail audits.

Central to real-time monitoring is the concept of a facet. *Facets* are essentially items such as configuration files, log files, processes, and sensitive database tables. A given target can have many facets associated with it. Based on the facets, Multiple Real-Time Monitoring standards can be created for Core Linux Packages, Exadata Compute Nodes, and User Access Linux packages.

Out of the box, Oracle Enterprise Manager provides hundreds of predefined facets. As with frameworks and standards, facets can be defined and customized.

Target-Type Facet	**Description**
Log Files	Monitor when regular users modify a log file (not a system user).
Binary/Library Files	Monitor if a binary is tampered with or when a binary is patched. Instead of listing each individual binary, it can also list a whole directory, but exclude frequently changing files.
Configuration Files	Capture changes made to any configuration files.
Security Key Files	Monitor files that store certificates, keys, and so on.
Security Configuration Files	Monitor files that configure how security works in the target type, such as encryption configuration, and so on.
Utility Processes	Monitor processes that normally run during a maintenance period, but should not be run during production.
Registry Keys	Monitor Microsoft Windows registry keys that affect the configuration of the target.
Configuration Tables	Monitor any database tables that store configuration data.

Each facet has an entity type that defines what kind of entities the facet describes. For example, for OS-level monitoring, there are OS File, OS Process, OS User, Windows Registry, and several Active Directory entity types. For database monitoring, the entity types include Table, View, Index, and Procedure, among others. Creation of facets is possible through the Facet Library screen.

When you define a real-time monitoring rule, the first thing you have to decide is what entity type on a host to monitor. For Oracle Enterprise Manager Cloud Control 12*c*, entity types that can be monitored with Real-Time Monitoring Rules include those listed in the following table:

OS File	**OS Process**	**OS User**	**Oracle Database User**
Oracle Database Table	Oracle Database View	Oracle Database Procedure	Oracle Database Profile
Oracle Database Link	Oracle Database Function	Oracle Database Package	Oracle Database Sequence
Oracle Database Trigger	Oracle Database Tablespace	Oracle Database Materialized View	Oracle Database Cluster

Using Notifications with Oracle Enterprise Manager

Notifications are an integral part of the Oracle Enterprise Manager management framework. Notifications can perform actions such as executing operating system commands (including scripts) and PL/SQL procedures when specific incidents, events, or problems occur. This capability allows you to automate IT practices. For example, if an incident occurs, such as a change in the operational (up/down) status of a database, the notification system can use an OS script to automatically open an in-house trouble-ticket so that the appropriate IT staff can respond in a timely manner. By using Simple Network Management Protocol (SNMP) traps, the Oracle Enterprise Manager notification system also allows you to send traps to SNMP-enabled third-party applications such as HP OpenView or BMC Remedy for events published in Oracle Enterprise Manager, such as when a certain metric has exceeded a threshold.

Patch Management with Oracle Enterprise Manager

Oracle Database Lifecycle Management Pack supports the entire Patch Management Lifecycle including patch advisories, predeployment analysis, rollout, and reporting. It is integrated with My Oracle Support to provide a synchronized view of available and recommended patches. These patches can then be analyzed for conflicts before deployment. One can then apply multiple patches to multiple databases in a single downtime window. The Deployment Procedures for patching are designed to enable maximum ease and minimum downtime by using sophisticated techniques such as rolling patching for RAC and out-of-place patching. Change Activity Planner (CAP) enables you to plan, manage, and monitor complex operations that involve dependencies and coordination of owners as well as multiple processes. These operations can include rollout of security patches every quarter, building new servers to meet a business demand, migration or consolidation of data centers, and rolling out compliance standards across an environment.

Using CAP an administrator can

- Plan change activity, including setting start and end dates; and creating, assigning, and tracking task status.

- Manage large numbers of tasks and targets, using automated task assignment and completion support.
- Use a dashboard where you can monitor your plans for potential delays and quickly evaluate overall plan status.
- Have task owners manage their own tasks using priority- or schedule-based view support on their own dashboards.

Configuration Compliance Within Oracle Enterprise Manager

While the powerful `SYSMAN` account can be used to navigate the compliance framework, Oracle supports Separation of Duty with highly granular privileges to control who can view results and perform specific actions. The following privileges and roles are supported to manage access to the configuration standards:

Privilege	**Description**
`CREATE_COMPLIANCE_ENTITY`	Create compliance standards, rules, and Real-Time Monitoring facets
`FULL_ANY_COMPLIANCE_ENTITY`	Manage compliance standards and compliance standard rules
`VIEW_ANY_COMPLIANCE_FWK`	View compliance framework definition and results
`MANAGE_TARGET_COMPLIANCE`	Associate a compliance standard to a target
`VIEW`	View a single target

Role	**Description**
`EM_COMPLIANCE_DESIGNER`	Create, modify, and delete compliance standards, compliance standard rules, and Real-Time Monitoring facets.
`EM_COMPLIANCE_OFFICER`	View compliance framework definitions and results.

Further Information

Security Information Portals

- **Oracle Database Security**
- **Database Security on Oracle Technology Network**

Oracle Product Documentation

- **All Oracle Database Documentation**
- **Oracle Database Security Guide**
- **Oracle Audit Vault and Database Firewall**
- **Oracle Advanced Security**
- **Oracle Database Vault**
- **Oracle Label Security**
- **Oracle Data Masking**