

Best Practices for Using Oracle Label Security with Oracle E- Business Suite

An Oracle White Paper
December 2004

Best Practices for Using Oracle Label Security with Oracle E-Business Suite

Introduction	3
Definition of the business problem.....	3
How Oracle Label Security solves the above problem	5
Policy definitions.....	6
Enforcement of policy definitions in sessions.....	6
Applicability of OLS to E-Business Suite.....	6
Best practices in creating an OLS environment for EBS.....	7
Steps in creating an OLS-enabled environment.....	7
Development of Policy definitions.....	8
Identification of Minimal subset of tables for OLS.....	9
Design of Mapping of Responsibilities to OLS Labels	10
Development of Package to Identify Session User.....	12
Development of initialization routines to enable label privileges in sessions	13
Creation of appropriate Labeling functions to set Data Labels	14
Solutions for non-interactive processing	15
Oracle 10g enhancements for OLS and VPD technology.....	15
Directory Enabled Label Security.....	15
Column Level Security Using VPD	17
Conclusion.....	19

Best Practices for Using Oracle Label Security with Oracle E-Business Suite

INTRODUCTION

Oracle E-Business suite is widely deployed in many environments. It consists of two large classes of Business Applications: Enterprise Resource Planning (ERP), which contains applications such as General Ledger, Payroll and Customer Relationship Management (CRM), which contains applications-modules such as Order Management. These applications are typically deployed on the intranet as well as on the internet. They form the core of Information Systems for an Enterprise. They are mission-critical applications and offer the required competitiveness to the Enterprise through the use of its Information Architecture.

Many of today's enterprises are global and data is derived from many sources due to its deployment over the internet. Today's internet environment and data consolidation efforts present unique data access challenges for large applications. For instance, you may have one division of the company, such as the Eastern Region, overseeing sub-regions such as New York, California, Texas, and so on. Depending on your identity (or what your application responsibility is), you would want to operate on striped portions of data. This session uses Oracle E-Business Suite as a case study and discusses the principles and details of how such a data striping requirement can be met with Oracle Label Security (OLS). This paper uses the example of an actual customer implementation of E-Business Suite and Oracle Label Security to develop a framework of best practices. In addition, we also introduce several new features of OLS, especially those that are new in 10g Release of Oracle RDBMS. We discuss the enhancements and correlate their relevance and usefulness to E-Business applications, especially from a functionality and manageability perspective. Further, we have designed this white paper as an informal guide to a step-by-step implementation of OLS in E-Business suite.

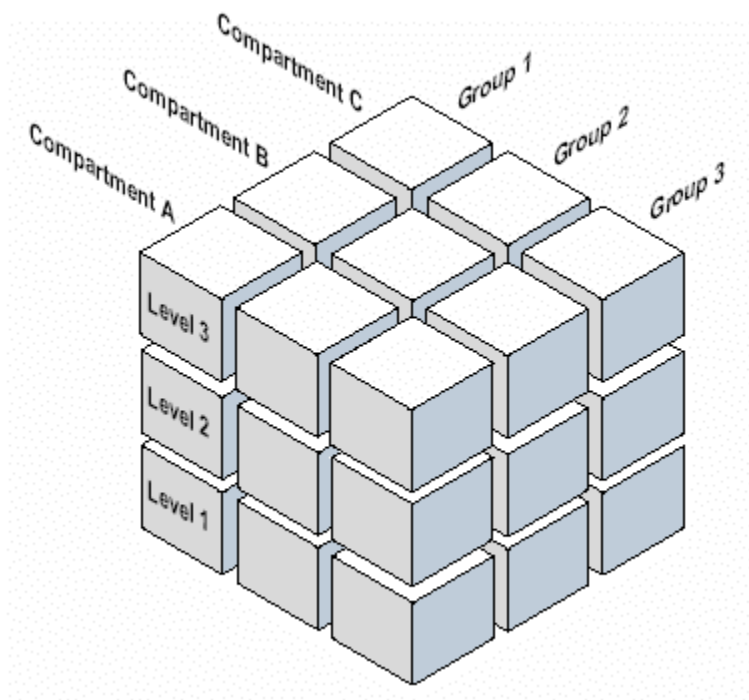
DEFINITION OF THE BUSINESS PROBLEM

If one division of your company oversees sub-regions, you may want to operate on striped portions of data, in many different complex ways. For instance, you may have a global customer, whose data is derived from many different regions in the country or even from many different countries around the globe. For instance, let us take the example of a Global Financial Services company, operating in many states within United States, as well as operating in many countries overseas. This

company may have a Customer contract that spans three countries, especially when the customer is an organization, using this firm for managing its investments. However, a Sales person working for the Financial Services Company may have the responsibilities only to operate on data pertaining to one country. Another individual may have the responsibility to operate on data for pertaining to two countries. A third individual, who may be more senior in the enterprise, may have the rights to operate on data from all three countries. In addition to these *Privileged groups*, an enterprise may also have classifications on the *type of data*, called *compartments*. For example, Consumer Loans information can be one compartment, Stock Holdings information may be another compartment, and Customer Location Information may be yet another compartment.

Further, the *operations* allowed on the data may also be restricted, depending on the responsibility of the person. In some cases, a specific employee may only be allowed to *view* the data. In other cases, a specific employee may be allowed to *modify* the data, but *not* be allowed to *create* any new data. These are discussed as *READ/WRITE permissions on compartments and groups* in this paper.

The operations that deal with data sensitivity are modeled as levels., e.g., Account Manager being able to view all confidential information while an employee would be able to view only unclassified internal company info. Thus, you have a complex matrix of responsibilities, with many dimensions and many possible values to these dimensions, as depicted below (using the above example):



Data Categorization with Levels, Compartments, Groups

Such multi-dimensional complexity used to be traditionally modeled by writing very complex applications. Another solution was also provided by using complex views that filter the data required. However, in a general case, such solutions can lead to a very high degree of complexity. For example, with 10 groups, 3 levels of data sensitivity and 5 compartments, we can be looking at 150 possible basic data filters. In addition, there may need to be more complex filters, if specific users are allowed to be in two or more groups. These can lead to mathematical *Combination* (N, R), where N is the total number of Groups and R is the number of Privileges given to a user from those groups. Implementation of such logic can be complex, expensive and can lead to performance degradation, as well. Oracle Label security offers a much simpler, supported and out-of-the-box solution, which does not require modification of ERP application code.

In addition to these Information Technology complexities, there are also other key business drivers that make Label Security an attractive option. There is a strong drive to control costs of an Enterprise and the cost of Information Technology, in many different ways, including the following:

- (1) Several data consolidation efforts are underway, where data is centrally stored in one place and access is provided across the network. In these scenarios, it is important to provide data striping to allow access to data only to users who have the necessary privilege.
- (2) In these efforts, multiple EBS *instances* are centrally consolidated to save IT infrastructure costs. In this methodology, data pertaining to all regions would co-exist in specific database tables and applications should be able to provide the *logical* data separation. OLS is one way to provide this separation, without having to significantly write or modify packaged applications such as EBS.
- (3) Globalization of the international business community requires the support of complex definitions of a customer, contracts, services, etc. In many cases these complexities require a security solution such as OLS to provide a *dynamic structured* view of the customer. We have discussed this with an example of a multi-region Financial Services firm, in this paper.
- (4) Consolidation of User Management is another effort that is ongoing. In this initiative, users are centrally managed; roles and responsibilities are centrally provisioned. Such centralization has several benefits, including a much better security infrastructure. Information Security is taking centre-stage in our current decade as a necessity for the stability and reliability of the entire structure of our society and its activities.

How Oracle Label Security solves the above problem

Oracle Label Security (OLS) consists of infrastructure that is enabled at the Database kernel, which allows restricted access to database rows and columns,

using information specific to a database session. Such restrictions are defined in *policy definitions* and enforced on any different types of database access, such as ODBC, 3-GI, sqlplus or any application interface to the database. Policy definitions allow the application designers to define complex Groups, Compartments and data access levels and allow OLS to enforce data filters by matching the policy definitions to the currently connected user's profile, allowing restricted access to data. This method provides for the implementation of the complexity described above, *without* having to make any changes to application code. In addition, as we discuss in a subsequent section, all policy and user authorization information can be centrally stored and administered via an Oracle Internet Directory, which uses the Industry-standard LDAP protocol. The policy definitions and user authorizations are provisioned and propagated automatically to multiple databases to allow OLS enforcement on individual tables and schema.

Policy definitions

Policy definitions are designed after studying the need for compartments, data access levels and Groups within the enterprise. We provide a complete example of a sample definition and its implementation in a subsequent section. Policy definitions are created either using OLS-APIs or using the Policy Manager Graphical User Interface (GUI). Based on the policy definitions data rows can be labeled using labels which are of the logical form Level:Compartment:Group, where numbers are internally used to represent these *labels*.

Enforcement of policy definitions in sessions

In order to enforce the policy definitions, when database access is made, the following steps are followed:

1. After defining the OLS policies, rows in tables are *labeled* to correspond to the logical form of the labels.
2. When OLS is enabled at the database level, each session that connects to the database is associated with its *label* or access rights.
3. If the session's label is more than or equal to the label of in the database row, the session is provided access to that row. This data restriction is enforced at the database kernel level and therefore it is uniformly applicable to all SQL operations within the session, including access to data using views or SQL-Joins.

APPLICABILITY OF OLS TO E-BUSINESS SUITE

When OLS is used with Oracle ERP/CRM (e-business suite), certain unique characteristics apply to the environment (in contrast with a typical custom database application where users are created in the database and access the database by logging into the database). Specifically, the following characteristics are distinguishing features of e-business suite, as it relates to the usage of OLS:

- There are no database users in e-business suite. All database sessions connect as 'APPS' database user. Thus, the roles and responsibilities of the end-user should be detected from the connected session and should not be enforced using APPS user. The users are 'global' users, either created in FND_USER table or authenticated using an Internet Directory.
- There are several batch jobs and other interfaces, such as Advanced Queues, which do not connect as APPS user and which tend to process transactions across multiple *compartments and/or* multiple *groups*.
- Users can switch responsibilities in the middle of the application and the user's labels and privileges need to be reset on the RDBMS side, depending on the new user's authorizations.
- Tables in e-business suite have been modeled with a few major tables and other *transactional* tables. These transactional tables are typically always accessed using the primary key or other values found from the major tables. Thus, regardless of whether transactional tables are OLS-enabled or not, their parent key references will automatically restrict the data, due to the presence of the major tables. We will illustrate this in a subsequent section with a complete example.

BEST PRACTICES IN CREATING AN OLS ENVIRONMENT FOR EBS

As we discussed in the prior section, EBS presents unique challenges and differentiates itself from a typical database application, where users are managed by the database itself. This section discusses some of the potential pitfalls and limitations of OLS, especially in the EBS environment and discusses how to mitigate such potential drawbacks.

Steps in creating an OLS-enabled environment

There are several steps in creating an efficient OLS environment for EBS. They are listed below and they are discussed in detail in separate sections:

- (1) Development of the policy, including Groups, Data levels and Compartments.
- (2) Identification of the *candidate* OLS-enables tables, keeping in mind that there could be a severe performance impact if too many tables are OLS-enabled.
- (3) Design of mapping of Roles/Responsibilities to OLS privileges.
- (4) Development of code to identify the actual user during a database session's initialization and set the session's OLS privileges

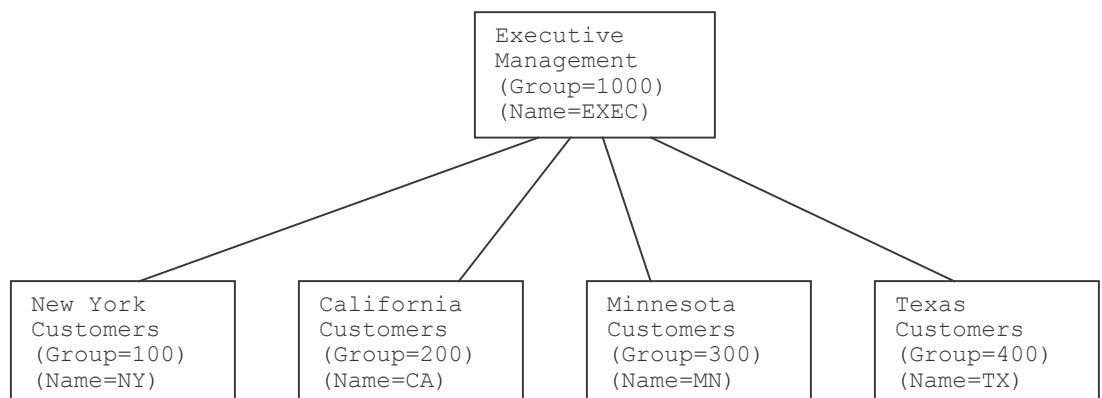
- (5) Development of Initialization and other routines to ensure that the session is re-initialized if the end-user switched to new Roles, logged into new EBS application or logged in as a different user.
- (6) Creation of appropriate trigger code that would create the labels on data, depending on the user's context.
- (7) Solutions for batch and non-interactive processes such as Advanced Queues, Materialized Views, Database Job processes.

We use an example that we have been discussing (Global Financial Services enterprise) to develop our model.

DEVELOPMENT OF POLICY DEFINITIONS

Oracle CRM consists of several applications, especially those that allow company employees to maintain and manage data related to the enterprise's customers. There are several different ways to categorize such data and restrictions on access to such data. To keep our model simple, let us focus on Groups and Hierarchy of groups. For instance, let us make the assumption that the enterprise consists of four distinct states in the USA. We can then develop a policy that models these four states as four groups. Further, let us assume that there is one group, which manages all these four groups, globally. Employees belonging to this global group, should be able to view data related to any of the four groups. Such data striping and enforcement of privileges is done by Oracle RDBMS, where Label security has been configured appropriately. When database session is initiated, Oracle RDBMS identifies Policies that are authorized to the session, and enforces the policy rules. For example, if the policy specifies that the user's session should have EXECUTIVE privilege, in order to access data across regions, then, only if the logged in user has the rights to *set* his/her privilege to EXECUTIVE Label (or privilege), access is granted to data across all regions. Definition of a policy is the first step in enforcing Label security. We also show the necessary APIs that achieve this result.

The following diagram depicts the business-hierarchy of the groups:



In our example, we may decide not to use any specific compartments. Further, we simply use two levels of data sensitivity, Public and Confidential. However, other compartments and levels can be defined for a specific enterprise, if necessary. The following piece of code uses OLS-APIs to create a policy called 'ENT' (short-form for Enterprise), using the design depicted above:

1. We can connect to the database as LBACSYS user (using SQLPLUS, for instance). This step creates the policy definition.

```
EXECUTE SA_SYSDBA.CREATE_POLICY ('ENT', 'XX_ENT');
```

2. These steps create the data levels.

```
EXECUTE SA_COMPONENTS.CREATE_LEVEL ('ENT', 22, 'C', 'CONFIDENTIAL');
EXECUTE SA_COMPONENTS.CREATE_LEVEL ('ENT', 11, 'P', 'PUBLIC');
```

3. These steps create the groups and the hierarchy of the groups.

```
EXECUTE SA_COMPONENTS.CREATE_GROUP ('ENT', 1000, 'EXEC', 'EXEC');
EXECUTE SA_COMPONENTS.CREATE_GROUP ('ENT', 100, 'NY', 'NY', 'EXEC');
EXECUTE SA_COMPONENTS.CREATE_GROUP ('ENT', 200, 'CA', 'CA', 'EXEC');
EXECUTE SA_COMPONENTS.CREATE_GROUP ('ENT', 300, 'MN', 'MN', 'EXEC');
EXECUTE SA_COMPONENTS.CREATE_GROUP ('ENT', 400, 'TX', 'TX', 'EXEC');
```

4. These steps create the labels that will be attached to the session as well as the database table rows for those OLS-enabled tables.

```
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL ('ENT', 10001, 'C::EXEC');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL ('ENT', 10002, 'C::NY');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL ('ENT', 10003, 'C::CA');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL ('ENT', 10004, 'C::MN');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL ('ENT', 10005, 'C::TX');
```

5. The next steps set specific logical session privileges (also known as user labels) that will allow access to the data values used in the tables that are OLS-enabled.

```
EXECUTE SA_USER_ADMIN.SET_USER_LABELS ('ENT', 'EXEC_USER', 'C::EXEC');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS ('ENT', 'NY_USER', 'C::NY');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS ('ENT', 'CA_USER', 'C::CA');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS ('ENT', 'MN_USER', 'C::MN');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS ('ENT', 'TX_USER', 'C::TX');
```

IDENTIFICATION OF MINIMAL SUBSET OF TABLES FOR OLS

This specific step represents the by far the *most important* step in the authors' opinion and also this is typically the step where care and attention needs to be paid to ensure a successful OLS implementation, especially for E-Business suite of applications. The next paragraph describes the design concepts in detail.

Ensure that you work with business analysts, consultants and other knowledgeable in CRM applications-architects, when such design of OLS enablement is made. More efforts spent on such design, better performance-efficient that the resulting OLS environment becomes.

Joins between two tables where both are OLS enabled can be expensive. Review the design and see if queries with such joins can be avoided.

When tables are OLS-enabled, there is a performance impact during queries, joins, views and several other complex SQL operations. The label security condition clause is executed for each *candidate* row in the table that is OLS enabled. For instance, if a table, such as AR.HZ_PARTIES is OLS enabled, then, each row that is not generally constrained by any other *where* clause in the query, would be filtered by executing the OLS filter code, generated by OLS mechanism. In practice, since the OLS filter code does not contain any I-O access and is usually cached in memory, execution times for each invocation of the filter code can be as low as 0.0001 seconds, for a typical, large enterprise-class operating platform. Even with such a small execution time per row, if 100,000 candidate rows have to be filtered out, this can lead to 10 seconds of performance bottleneck, *simply* to enforce OLS. On the other hand, it is unlikely that a table such as AR.HZ_PARTIES would be required to scan 100,000 rows for a typical customer. Since there are primary key references in this table and in others, in practice, only 1-2 records would be candidates, since the remaining records would be filtered out, using other *where* clauses.

As a contrasting example, a table such as GL.GL_JE_LINES_ALL should *never be* OLS-enabled. This table will always be accessed (in interactive mode, especially, as opposed to batch) using several *where* clauses and primary keys. If the *parent* tables that participate in the *where* clause are labeled, then there is really no need to label this particular table. In fact, labeling such *transactional, child* tables is the most common mistake in the design of OLS-enablement that leads to poor and often unacceptable performance.

In our example, we chose the following minimal CRM tables:

```
AR.HZ_PARTIES
AR.HZ_CUST_ACCOUNTS
AR.HZ_PARTY_SITES
AR.HZ_CONTACT_POINTS
AR.HZ_PARTY_RELATIONSHIPS
AR.HZ_PARTY_SITE_USES
AR.HZ_CUST_ACCT_SITES_ALL
AR.HZ_CUST_SITE_USES_ALL
OKC.OKC_K_HEADERS_B
CSI.CSI_INSTALL_PARAMETERS
CSI.CSI_ITEM_INSTANCES
INV.MTL_SYSTEM_ITEMS_B
```

DESIGN OF MAPPING OF RESPONSIBILITIES TO OLS LABELS

When the user logs into E-Business applications, he/she is connected to the database as 'APPS' user. We need to develop a design that would provide information on the users to enable the application context to determine the appropriate label to be set for the session. We designed this by developing a

Avoid frequently used queries with Trusted Stored Procedure Calls. Review the design to see if there is a real need to make such a procedure Trusted..

customized table of Users' responsibilities to their Groups. There could be several other ways to achieve this result. However, in the context of CRM, Customized Responsibilities are often created to ensure that the appropriate base applications are used for the end-user. For instance, a responsibility called 'NY SalesPerson' would be created for the Sales representatives in New York and a responsibility called 'EXECUTIVE VP' would be created for those who play the role of Executives at Head Quarters. A sample data of this table is provided below:

```
Rem
Rem Connect as lbacsys
Rem This creaes a mapping of responsibilites to ENTERPRISE_ID

set define off
DROP TABLE XX_ENT_RESPONSIBILITIES
/

CREATE TABLE XX_ENT_RESPONSIBILITIES
(
  ENT_ID          VARCHAR2(30) NOT NULL,
  RESPONSIBILITY_NAME VARCHAR2(100) NOT NULL
)
/

GRANT SELECT ON XX_ENT_RESPONSIBILITIES TO PUBLIC
/

INSERT INTO xx_ENT_RESPONSIBILITIES ( ENT_ID, RESPONSIBILITY_NAME )
VALUES ( 'NY', 'New York Customer Sales Representative')
/
INSERT INTO xx_ENT_RESPONSIBILITIES ( ENT_ID, RESPONSIBILITY_NAME )
VALUES ( 'EXEC', 'EXECUTIVE MANAGEMENT OFFICE')
/
<<< More responsibilities of this type would be created and added to
this table, depending on the number of regions and requirements for
finer control of other user functions >>>
COMMIT
/
```

Many of the routines given here assume valid data. It is always a recommended practice, however, to check for errors and traps PL/SQL exceptions, at every stage. An error logging mechanism, especially while developing the code would be useful to debug any issues with the code or the environment. We have tried to keep the essence of this paper simple in providing only the minimally required statements.

DEVELOPMENT OF PACKAGE TO IDENTIFY SESSION USER

The next step in enabling label security is to create procedures that can be executed from within the EBS applications, that would set the labels of the session, logged in by the user. In order to do this, we need to be able to identify the actual user. Oracle EBS provides PL/SQL functions that can provide all the required profile of the connected user. For instance, we can get the user's name, user's current responsibilities, etc. Until the previous release of EBS, user management was in database tables, such as FND_USER and FND_USER_RESPONSIBILITY. In the new releases of EBS, these can be stored in an Oracle Internet Directory, and administered centrally across all platforms. We will discuss these enhancements in a subsequent section. A sample procedure is shown below, which can be called from any part of EBS code, which will verify the privileges of the user and set his/her session label according to the privileges. Typically, the function APPS.XX_ENT_APPS.CLASSIFY_USER would identify the user. The procedure APPS.XX_ENT_APPS.SET_SESSION_LABEL would set the OLS label for that session.

```
CREATE OR REPLACE PACKAGE XX_ENTS_APPS AS
FUNCTION CLASSIFY_USER RETURN VARCHAR2;
FUNCTION GET_USER_LABEL RETURN NUMBER;
PROCEDURE SET_SESSION_LABEL;
END;
/
CREATE OR REPLACE PACKAGE BODY XX_ENT_APPS AS
FUNCTION CLASSIFY_USER RETURN VARCHAR2 IS
vCurrResp VARCHAR2(200) := NULL;
vCurrUser VARCHAR2(200) := NULL;
vUser VARCHAR2(100) := NULL;
vEntResp VARCHAR2(200) := NULL;
vApp1 VARCHAR2(200) := NULL;
BEGIN

    SELECT APPS.FND_GLOBAL.RESP_NAME INTO vCurrResp FROM DUAL;
    SELECT APPS.FND_GLOBAL.USER_NAME INTO vCurrUser FROM DUAL;
    SELECT APPS.FND_GLOBAL.APPLICATION_SHORT_NAME INTO vApp1 FROM DUAL;
    SELECT ENT_ID INTO vEntResp FROM LBACSYS.XX_ENT_RESPONSIBILITIES
        WHERE RESPONSIBILITY_NAME=vCurrResp;
    vUser := vEntResp;
    RETURN vUser;
END;

PROCEDURE SET_SESSION_LABEL IS
vResp VARCHAR2(200);
vMsg VARCHAR2(200);
BEGIN
    SA_SESSION.SET_ACCESS_PROFILE('ENT',
        APPS.XX_ENTS_APPS.CLASSIFY_USER||'_USER');
END;
END;
/
```

DEVELOPMENT OF INITIALIZATON ROUTINES TO ENABLE LABEL PRIVILEGES IN SESSIONS

The initialization routines shown here are examples. However, depending on the actual deployment environment, these routines may have to be enhanced or supplemented. Also, only the ASF application has been shown to be initialized here. In actual deployments of CRM or ERP, there may be many more applications that would require similar initializations.

Oracle EBS allows customers to extend the basic infrastructure by providing entry points, where code or data specific to an end-user environment can be provided. Such code or data is used and executed by EBS infrastructure, subject to certain rules of developing them. One such method is the concept of *Application Initialization routines*. These can be user-defined code, which will be executed when various conditions or events occur, including the following:

- (1) When a user connects and creates a new session
- (2) When a user switches his/her responsibilities
- (3) When a user switches his/her user-id to another user
- (4) When a user changes his/her application from one to another.

We have provided a sample code here that would be the initialization code, which would be executed when these events occur. EBS provides a mechanism to *deploy* this code, by *registering* these routines in the appropriate applications.

A sample initialization code that sets the user's label (with an appropriate call to the APPS.XX_ENT_APPS.SET_SESSION_LABEL, which was shown in the previous section), is shown below: (This code is executed, while connected as the APPS user)

```
BEGIN
FND_PRODUCT_INITIALIZATION_PKG.REGISTER('ASF',
    'APPS.XX_ENT_APPS.SET_SESSION_LABEL', 'APPS');

FND_PRODUCT_INITIALIZATION_PKG.AddInitCondition('ASF',
    'APPL', 'APPS');
FND_PRODUCT_INITIALIZATION_PKG.AddInitCondition('ASF',
    'RESP', 'APPS');
FND_PRODUCT_INITIALIZATION_PKG.AddInitCondition('ASF',
    'USER', 'APPS');
FND_PRODUCT_INITIALIZATION_PKG.AddInitCondition('ASF',
    'ORG', 'APPS');
END;
```

In addition, we also need to provide OLS privileges to the APPS and other database users, so that they can switch their sessions to any of the defined OLS groups. The following code should be executed to provide this functionality:

```
EXECUTE sa_user_admin.set_user_privs('ENT', 'APPS', 'PROFILE_ACCESS');
EXECUTE sa_user_admin.set_user_privs('ENT', 'APPLSYS', 'PROFILE_ACCESS');
EXECUTE sa_user_admin.set_user_privs('ENT', 'APPLSYSPUB', 'PROFILE_ACCESS');
EXECUTE sa_user_admin.set_user_privs('ENT', 'AR', 'PROFILE_ACCESS');
```

CREATION OF APPROPRIATE LABELING FUNCTIONS TO SET DATA LABELS

The labeling function should also be written efficiently: Avoid table reads or other lookups. If variables have to be referenced, they are best derived from context using OLS APIs, as given in this example.

Creating data labels dynamically in insert/update operations can be expensive. Create all the data labels upfront.

When data is inserted into the tables, we need to have a mechanism of capturing the end-users labels and updating the tables with the appropriate label for the data. For instance, if a customer belongs to New York, you would want to create the AR.HZ_PARTIES record for that customer with 'NEW YORK' label. The following labeling function is created on the above tables to provide that functionality. The two parameters for this script file are the schema name (such as AR) and the table name (e.g., HZ_PARTIES). The user-defined packed procedure, discussed in a prior section is used here to determine the user's privilege and set the label accordingly.

The Labeling function needs to be specified when applying the policy to the table.

```
CREATE OR REPLACE FUNCTIONli$&2
RETURN LBACSYS.LBAC_LABEL
AS
vUser VARCHAR2(200);
BEGIN

    vUser := APPS.XX_ENT_APPS.CLASSIFY_USER;
    return TO_LBAC_DATA_LABEL('ENT', 'C: ' || vUser);
END;
/

GRANT EXECUTE ON &1.li$&2 TO LBACSYS;
GRANT EXECUTE ON TO_LBAC_DATA_LABEL TO &1 WITH GRANT OPTION;

SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
POLICY_NAME => 'ENT' ;
SCHEMA_NAME=> AR;
TABLE_NAME => 'HR_PARTIES'
TABLE_OPTIONS => 'READ_CONTROL,WRITE_CONTROL,CHECK_CONTROL'
LABEL_FUNCTION => 'HR.LI$&1' .
PREDICATE => NULL);
```

Non-interactive processing is primarily oriented towards a 2-tier and/or a database-oriented architecture, since there is no browser session acting on behalf of the logged-in user. Oracle RDBMS provides several entry points to initialize and configure sessions. Built-in functions and routines such as SYS_CONTEXT can provide information on the session's context and this information can be used to set the appropriate session labels.

SOLUTIONS FOR NON-INTERACTIVE PROCESSING

In an EBS environment, there are several situations that require non-interactive activity. These include the FND_JOBS, database jobs, several reports, Materialized Views, usage of Advanced Queues to process data from one application to another such as Interfaces between Order Management and Inventory or Invoicing. In these cases, the above mentioned Product Initialization routines are not called. For batch environments, there are several solutions to enforce labels, including the following:

- (1) Reporting jobs that span several groups can be run as EXECUTIVE user group. Alternatively, a specific privileged database user and a privileged label can be created with access rights to all of the stripes of data.
- (2) When Advanced Queues process transactions, we can run session initialization routines to determine that it is not being run as an interactive user. At this point, more privilege can be given to the session to allow for the processing of ALL of the data pertaining to the transaction. A business transaction may also span multiple groups and in those cases, such enablement of privileged access would be a necessity. Oracle RDBMS provides several entry points, including the ON-LOGON triggers, where the end-user's context can be set differently.
- (3) If Materialized Views are used, the database owner of the Materialized views should be given the appropriate label, in order to process the data pertaining to the view. In some cases, data from more than one group may be needed to create the view. The connected session will have to set its context to perform this function.

ORACLE 10G ENHANCEMENTS FOR OLS AND VPD TECHNOLOGY

Oracle RDBMS 10g provides several new and enhanced features in the enablement of OLS. Further, 10g is also built with centralized user management and data consolidation in mind. Oracle Internet Directory (OID) is Oracle's flagship platform for Centralized User Management. It uses Industry-standard Lightweight Directory Access Protocol (LDAP) to support several different business object-classes and their attributes. User entries that consist of several complex attributes, as well as OLS policy definitions can be stored in an OID. This would also act as the master database of record for all Information Security implementations. The following subsection discusses the architecture of OLS, which is OID-enabled.

Oracle Internet Directory also provides a complete set of Industry-standard LDAP-compliant infrastructure to manage users, including Authentication, Authorization and other Account Information, also denoted as the users' AAA.

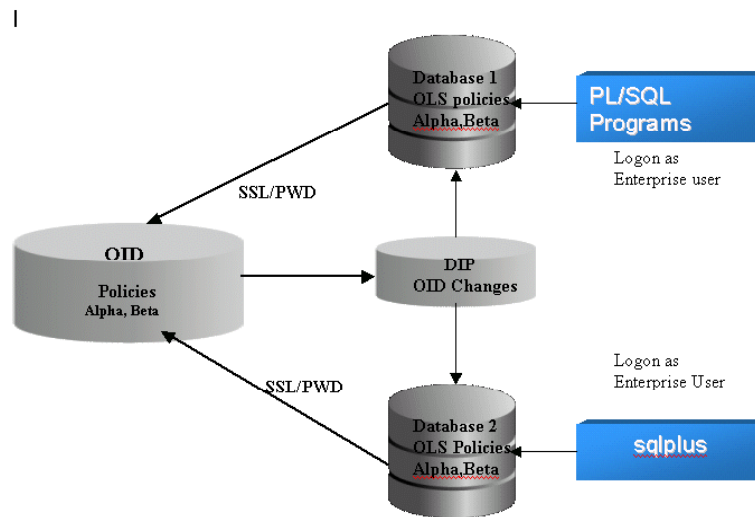
Directory Enabled Label Security

In release prior to 10gR1 Oracle Label Security has relied on the Oracle database as the central repository for policy and user authorizations. This architecture leveraged the scalability and high availability of the Oracle database. But, in a typical enterprise wide application there would be number of Databases involved. Label Security Policies and user authorizations would have to be maintained in a number of databases scattered through out an enterprise. Often an administrator may not

be able to make timely modification of authorizations, and may lead to increased security risks.

Directory enabled Label Security feature allows for Management of policy information in a centralized LDAP repository. It leverages Oracle's Identity Management infrastructure, which includes the Oracle Internet Directory.

The restriction that Policy authorizations can be assigned to only database users is no more present. Labels and Privileges can be directly assigned to enterprise users. Directory (or global) users would be able to login into a Database using the Enterprise User Security feature. The policy definitions and user and authorizations which were assigned in the central directory will be automatically provisioned and propagated to the individual databases using the Directory integration platform (DIP) server. The global user's labels and privileges are automatically available during session initialization.



Directory enabled Label Security Architecture

Following are the steps in creating an efficient OLS environment using the directory enabled Label Security feature.

- (1) Identifying the Databases that need to be protected with Label Security and registering them with the central directory where all the user and policy information is managed.
- (2) Development of the policy, including Groups, Data levels and Compartments in the central Oracle Internet Directory.
- (3) Design of mapping of Roles/Responsibilities to OLS privileges.

- (4) Creating user profiles (with labels and privileges) in the directory based on the identified Roles/Responsibilities and assigning enterprise users to the profiles.
- (5) Identification of the *candidate* OLS-enabled tables in the registered databases, keeping in mind that there could be a performance impact based on the queries in the application (joins with too many OLS-enabled Tables).

The above 5 steps are similar to configuring E-Business suite with standalone Label Security. The steps (4) and (5) discussed in the section on “Steps in creating an OLS environment” would no longer be necessary. Here the application users who are directory users are granted authorizations, which take into effect when they login into the directory enabled Database. An Example directory enabled OLS implementation is stated below in the lines of the same Global Financials example so far discussed in the paper

1. Create a policy

```
olsadmintool createpolicy -name ENT -colname XX_ENT -options  
READ_CONTROL,WRITE_CONTROL
```

2. Create Label Components

```
olsadmintool createlevel -polname ENT -tag 22 -shortname C -longname CONFIDENTIAL  
olsadmintool createlevel -polname ENT -tag 11 -shortname P -longname public  
olsadmintool creategroup -polname ENT -tag 1000 -shortname EXEC -longname EXEC  
olsadmintool creategroup -polname ENT -tag 100 -shortname NY -longname NY  
-parentname EXEC
```

3. Create datalabels

```
olsadmintool createlabel -polname ENT -tag 10001-value C::EXEC  
olsadmintool createlabel -polname ENT -tag 10002 -value C::NY
```

4. Create userprofiles

```
olsadmintool createprofile -polname ENT -profname EXEC_PROFILE  
-maxreadlabel C::EXEC  
olsadmintool createprofile -polname ENT -profname NY_PROFILE -maxreadlabel C::NY
```

5. Assign users to profiles

```
olsadmintool adduser -polname ENT -profname EXEC_PROFILE -userdn  
cn=jdoe,dc=acme,dc=com
```

**CONNECT as jdoe to any registered database gets EXEC user labels
NO NEED FOR USER IDENTIFICATION PACKAGES OR INITIALIZATION
ROUTINES!**

Note: For all the steps 1 to 5 ,the directory details and credentials need to be specified

Column Level Security Using VPD

Yet another requirement with regard to protecting sensitive information and at the same time making available general non sensitive information, can be met by using Column-level VPD Solution. Security policy can be defined such that , you enforce row level security only when a security-relevant column is referenced in a query

The following example shows an employee table protected by a VPD policy. An employee should be able to view his SALARY and SSN , but should not be able to view such information for other employees. Security relevant columns for this policy are SALARY and SSN. Policy is enforced only when queries referencing sensitive columns (security relevant columns) SALARY and SSN are executed. When the query does not reference any security relevant columns say only employee name and location, then all the rows are fetched.

```
/* Create a policy function */
CREATE OR REPLACE FUNCTION empf1 AS
con VARCHAR2 (200);
BEGIN
    con := 'ename = sys_context('userenv',session_user)';
    RETURN (con);
END empf1;

/* Then the policy is added with the DBMS_RLS package as
follows: */
BEGIN
    DBMS_RLS.ADD_POLICY (object_schema=>'scott',
object_name=>'emp', policy_name=>'emp_policy',
function_schema=>'pol_admin', policy_function=>'empf1',
sec_relevant_cols=>'SALARY, SSN');
END;

Rem An Employee SKING who is in deptno 30 connect to DB
SELECT ENAME, DEPTNO, LOCATION, SALARY FROM EMPLOYEE
WHERE DEPTNO=30;

Rem Returns only the row for employee "SKING"
SELECT ENAME, DEPTNO FROM EMPLOYEE WHERE DEPTNO =30;

Rem Returns the rows for all employees in deptno 30
```

In a variant of column -level VPD sensitive columns can be masked. The column masking behavior displays all rows, but returns sensitive column values as NULL. To set this behavior set the sec_relevant_cols_opt parameter of the DBMS_RLS.ADD_POLICY procedure to dbms_qls.ALL_ROWS. In the Example discussed following is the way to do column masking

```

/* add the ALL_ROWS policy */
BEGIN
    DBMS_RLS.ADD_POLICY(object_schema=>'scott',
        object_name=>'emp', policy_name=>'emp_policy',
        function_schema=>'pol_admin', policy_function=>'empfl',
        sec_relevant_cols=>'SALARY,SSN',
        sec_relevant_cols_opt=>dbms_rls.ALL_ROWS);
END;

Rem An Employee SKING who is in deptno 30 connect to DB

SELECT ENAME, DEPTNO, LOCATION, SALARY FROM EMPLOYEE
WHERE DEPTNO=30;

Rem Gives rows for all the employees in department 30,
except that the SALARY and SSN columns for other
employees will show as NULL.

```

CONCLUSION

We examined the Business Problem in EBS implementations, where several business requirements, including data consolidation, user consolidation and other business drivers, require us to stripe the data and provide multiple, complex combinations of these stripes, depending on the end-user and the processing context.

Many of the Business drivers are motivated by cost savings. Some others are motivated by complex business requirements that have come about as a result of the current global market-place. Virtual Private Databases, especially Oracle Label Security is an implementation that does not require a lot of development or changes to basic applications and their functionality. This paper demonstrates how the OLS can be applied to an Oracle EBS environment. It is the authors' implementation experiences that have helped in shaping this white paper, where a viable solution was created to solve a complex business and Information Technology problem. It is our belief that if all of the principles discussed in this paper are followed, OLS is an appropriate and often cost-effective solution to provide complex, striped views of data, without having to resort to multiple database instances and multiple, unconsolidated views of data. Without OLS, the costs of maintaining multiple sources of data, as well as providing an interface to consolidate complex data views, would be very high. Further, new enhancements to VPD and OLS technologies in Oracle RDBMS 10g, would facilitate additional functionality as well as provide a centralized, Industry-standard, LDAP-based security infrastructure.



Best Practices for Using Oracle Label Security with Oracle E-Business Suite
December 2005

Authors: Roger Raj (Roger.Raj@oracle.com)
Srividya Tata

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Copyright © 2003, Oracle. All rights reserved.

This document is provided for information purposes only
and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to
any other warranties or conditions, whether expressed orally
or implied in law, including implied warranties and conditions of
merchantability or fitness for a particular purpose. We specifically
disclaim any liability with respect to this document and no
contractual obligations are formed either directly or indirectly
by this document. This document may not be reproduced or
transmitted in any form or by any means, electronic or mechanical,
for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.