Follow the Trail of Identity:
Aiming for Complete Authorization

Mag. Dr. Thomas Petrik
Sphinx IT Consulting
Wien

Keywords

Oracle Database, Security, VPD, Authorization, Authentication, LDAP, Radius, Kerberos, SSO, Roles, Privileges, Context

Introduction

In the light of GDPR and other regulations like BCBS-239 (for financial institutions) it is not any more an option to rely on generic technical database users and shared user accounts. Also the argument that the application handles security anyways and there is no need to build up a personalized database security does not apply any longer. In each and every system there are the so called power users who access the database via 2-tier architectures for their daily reporting and maintenance work using tools which do not even provide an application security layer. Still the most popular reporting tool is MS Office (Excel and Access). Therefore, it is key to build a consistent database security architecture which meets the users' needs on the one hand and satisfies the regulatory requirements on the other hand.

Luckily, Oracle database provides nearly all the tools you need to accomplish this task. However, there is no out of the box solution, you have to build your own security solution.

The Basic Principles of an Elaborated Security Architecture

1. Security is always bound to the human user.
2. Security is tool-agnostic.
3. Audit shows the human user on each and every access path.
4. Human users cannot delegate their access rights without permission.
5. Security must not prevent users from collaboration in various projects.

The first two points refer to the fact that it must be irrelevant which tool is used by a human user or over which path the human user enters the database. It can be via a 2-tier or 3-tier application, it could be with a native client like SQLPlus or via a sophisticated BI application. In any case the user should always see the same data, not more and not less.

Point 3 refers to the basic requirement that an audit should always reveal the real human user behind an action.

Point 4 and 5 are sometimes underestimated or even ignored by architects and DBAs. A human user (who is granted access to specific data) should have basically no permission to grant this data to other colleagues. Otherwise data will be spread all over the company within short time and by this all the security which was built before becomes useless. On the contrary, people must have the possibility to work together in projects or within departments, to share data in a controlled way and even to play around with them in a sandbox environment.

In a real world scenario it will be a combination of both approaches: DB security for human user access and application security as an add-on in order to enhance the user friendliness in the first place.

Furthermore, in case of an application with a high amount of users who will never ever access the DB directly (e.g. WEB shops, e-business systems, etc.) application security will be enough for this part while all other (internal) back office users will again be subject to human user based database security.

The three Levels of Database Security

These three levels of DB security must be taken into account:

1. Object level security (OLS – do not confuse with Oracle Label Security!)

   This level covers classical object privileges (select, insert, update, delete) via roles or direct granted privileges.

2. Row Level Security (RLS)
3. Column Level Security (CLS)

The latter two levels refer to the so called fine grained access security and make use of the Virtual Private Database (VPD) concept in an Enterprise Edition. For SE2 users, unfortunately, no such completely transparent method is available.

For OLS object groups need to be defined first, meaning groups of tables and views which have some business meaning in common. These groups will later be associated with human users. In fact an object group will be related to a database role. Object groups can be overlapping.

For RLS and CLS tenants need to be defined up front. A tenant can be any common (security relevant) selection criteria which can be found in all rows of a table, e.g. an institute, a subsidiary, a product group. Also time based tenants like months or years could be considered.

Multiple tenants could be part of the same table and, therefore, could be used in an AND or OR relationship at the same time.

Furthermore, it should be noted that data models are never perfect and that the same tenant code might appear in different formats (e.g. ‘0100’ as a varchar2 or 100 as a number). This means that the VPD policy must be prepared to handle such inconsistencies.

RLS basically means a horizontal partitioning based on tenants, while CLS means that sensitive columns will return only NULL values based on the tenant combination.

The n-dimensional Space of Authorization

Users, object groups and tenants are combined in an n-dimensional space. It can be easily seen that with this degree of complexity it is no longer possible to manage privileges in the database manually. It is, therefore, beneficial to create a metadata repository which acts as an abstraction layer between some central identity management system (IDM) and the data dictionary.

An API should be provided to populate the metadata repository (the IDM system or an APEX GUI would use this API) and a set of background jobs (dbms_scheduler) takes care to translate the security metadata into proper DDLs in order to persist the security in the data dictionary as privileges, roles and VPDs.

One important task of such a framework is to ensure consistency between metadata and dictionary all the time.

The Complex Story of Authentication

The common approach is to use shared user accounts in the database. In this case the real human user can only be identified via the session context (normally by using the “osuser” attribute from v$session).

Although this solution is easy to manage for DBAs it has several drawbacks:

1. Password changes are nearly impossible.
2. Authentication is weak.
3. All users implicitly get the same (shared) privileges.
4. Less flexible solution for project oriented organizations.

The better and much more secure solution is to use dedicated human user accounts. Beside these accounts an unlimited number of shared schemas (maybe project or department related) may be created. The human user accounts are configured as proxy users for these shared schemas. This method guarantees personal authentication on the one hand and enables the use of shared schemas for better collaboration on the other hand. The human user information can be retrieved from the "proxy_user" attribute of the session context while the user is actually logged into the shared schema and has inherited all the roles and privileges of this sandbox user.

A more fine grained approach is the use of the "with roles" clause for the proxy user configuration. In this case the target schema has to have all possible roles granted (in addition to the "create session" privilege) but the human user has only access to these objects which are part of the roles he has been granted by the "with role" clause.

However, the Oracle role concept has two big limitations:

1. Only 148 roles per session can be active concurrently (this is a hard coded limit in Oracle).
2. Roles have no effect for object access via PL/SQL.

Therefore, especially if PL/SQL should be used by human users the shared schemas must be provided with explicit object privileges.

A common danger with shared schemas is the uncontrolled distribution of data. By default a human user logged into a shared schema (as being the owner of the objects in this schema) has the possibility to grant privileges on these objects to other users. This should be prevented by DDL-triggers or by use of Database Vault means. The same is true, of course, for the individual human user accounts if they have been granted quotas and the "create table" privilege.

While talking about dedicated human user accounts it becomes clear that a central authentication method is key for a successful implementation of such a concept. The use of LDAP (MS-AD or OUD), Kerberos and Radius is a cost-free option for Enterprise Edition as well as for SE2 since 11.2.0.4.

3-Tier Architectures

In 3-tier architectures it is imperative to hand over the human user to the DB session context. Most application servers need a technical user in order to connect to the database with a session pool but they provide user exits in the data source definitions where any PL/SQL procedure call may be placed with the logon user being handed over via some server variable. Such a procedure must be executed with invoker rights in order to set roles depending on the actual logon user. The technical user itself has just a "create session" privilege and an execute privilege on this very procedure. Furthermore, the procedure places the human user information in some secure application context which enables audit and VPD usage.

Modern application servers (especially some BI tools) support the use of Kerberos SSO delegation. In this case the 3-tier architecture behaves exactly like a 2-tier authentication. No technical user is needed.

VPD Performance

The Virtual Private Database is the key concept for RLS and CLS as already pointed out. However, it is important to understand how VPD works internally in order to classify the potential impact on the overall database performance.

VPD injects literals into the where clause for RLS and it adds CASE-statements for CLS. This means that the hard parse rate increases and the cursor cache grows on the one hand but execution plans become much more accurate on the other hand due to perfect partition pruning and less problems with bind peeking. Therefore, it strongly depends on the query characteristic and on the overall system load if VPD affects the application in a positive or negative way. In general it can be said that for heavy load OLTP systems one should be very careful with using VPDs while a DWH system benefits always.

It should also be noted that for mixed load systems access to the same table can be enabled via VPD for interactive human users (e.g. analysts and other people from the back office) and without VPD via a technical user which serves the online application at the same time. The simple trick is to grant the technical user the "exempt access policy" privilege.

Conclusion

A successful and complete implementation of an authorization architecture should follow these rules:

1. Use central authentication.
2. Manage authorization configuration centrally and persist it in the DB.
3. Fully automate the privilege management (roles, privileges, VPD, etc.)
   Build a security framework.
4. Use declarative configuration only, no DBA intervention, no coding.
5. Setup audit in order to verify the configuration.

Contact:

Mag. Dr. Thomas Petrik

Sphinx IT Consulting

Aspernbrückengasse 2

A-1020 Wien

Phone:	+43 664 155 8304
Fax:	+43 (1) 599 31-99
E-Mail	Thomas.Petrik@sphinx.at
Internet:	www.sphinx.at