Oracle Database Security - Top Things You Could & Should Be Doing Differently

Simon Pane
Pythian

Keywords:

oracle database security

Introduction

When reviewing existing database security configurations (i.e. performing a security audit) DBAs often focus on privileges and passwords. And while privilege reviews are important, they are only the tip of the iceberg. Similarly, when considering additional ways to enhance database security, DBAs often look to add-on components and tools. Most of which are at-cost or require additional licenses.

However, there is typically considerable room for increasing Oracle database security by simply adjusting the current usage of the existing Oracle database technology or by modifying DBA operational practices.

This paper reviews (at a high level) opportunities for improvements, best practices, and common mistakes which can lead to security exposures.

Avoid Clear Text Passwords in Oracle Net

DBAs often fail to consider that by default, Oracle network traffic is unencrypted. When performing a TNS connection the initial credential information is automatically encrypted but the rest of the session traffic is not. Hence all DDL & DML commands and the returned data are transferred in clear text and is easily readable by anyone who can access (or "sniff") network traffic.

This can be an issue when DBAs change password using "`ALTER USER …. IDENTIFIED BY` …" syntax whereas the entire command including the password is transferred through Oracle Net in clear-text (just like any other command).

Fortunately, this situation is simple to remedy via a number of techniques:

1) Don't use "`IDENTIFIED BY`" syntax over Oracle Net – use only on local BEQ IPC connections instead.
2) Use the "`PASSWORD`" command instead. The "`PASSWORD`" command will automatically encrypt the password information as it travels through Oracle Net. (Surprisingly many Oracle DBAs and Developers are unaware that the "`PASSWORD`" command exists.)
3) Encrypt all Oracle Net traffic. Recently Oracle Net encryption was decoupled from the Advanced Security Option meaning that all 11.2.0.4+ customers can encrypt all Oracle Net traffic easily and without additional cost.

Protect Password Hash Values & Salts

DBAs often make efforts to properly secure actual passwords but do not similarly protect the password hash values and salts. Sometimes they may even quite cavalierly post hash values and salts in work logs and change management systems as a "back-out" mechanism when implementing password changes. Additionally, password hash strings are often included in Data Pump export files.

However, DBAs should be cognizant that cracking Oracle database passwords (offline) from their hash values is not overly onerous. Especially the old 16-digit HEX DES password hashes.

Consequently, DBAs should be mindful of this and should treat password hash values and the associated salts (including those in Data Pump export files) with the same protection level as the actual passwords themselves.

Avoid Passwords in Script Variables

Almost all DBAs have OS based scripts. For example, RMAN backup scripts that are typically scheduled via CRON. However, how those scripts connect to the Oracle database is often not implemented properly.

DBAs frequently try to protect credentials in a hidden and permission restricted OS file and then have their scripts access that file to connect to the database. Sometimes the scripts themselves may in avertedly expose these passwords as operational "environment variables" to other Oracle related system processes.

The optimal way for configuration scripts to access the Oracle database is to use a "Secure External Password Store" and an Oracle Wallet file. The Secure External Password Store and associated Oracle Wallet file is relatively simple to implement and remarkably functional. For example, it is compatible with all sorts of connection types including but not limited to:

- Sqlplus connections.
- RMAN connections including to the catalog database.
- Oracle Data Pump connections.
- JDBC Thin Connections.

Use the Scheduler

Another way to avoid the overhead of securely managing credentials for DBA OS scripts is to move the scripts from the OS to the database and schedule them via the Oracle Database Scheduler. If the scripts are doing database centric processing anyway, then why have their logic defined at the OS level and have to worry about DB connections. Instead, run the script logic from within the database (as PL/SQL scheduled using the Oracle Scheduler) and therefore never have to worry about database connectivity.

Realistically though, DBAs usually do not want to translate OS or shell scripts into PL/SQL. However, there still are advantages to using the Oracle Scheduler as a scheduling tool (in place of CRON). But when doing so, it is important to ensure that the Oracle Scheduler security is implemented optimally.

How Oracle Scheduler jobs interact with the OS can be configured in a few different ways. With Oracle 11g and 12c, the OS credential can be securely stored in the database and associated with the job. For Oracle 10g, non-SYS owned jobs interact with the OS via credentials stored in the `externaljob.ora` file.

The OS user and group defined in the `externaljob.ora` file is sometimes mistakenly set to "`oracle`" and "`dba`" by DBAs when trying to properly implement this mechanism. However, such a configuration is potentially an OS injection risk as any database user with permission to create external jobs will interact with the OS as the Oracle software owner!

This risk is mitigated by specifying a minimally privileged user in the `externaljob.ora` file (which defaults to "`nobody:nobody`") or if using Oracle 11g or 12c, not to use this mechanism at all and instead use a database stored credential.

Further, as a best practice, Oracle Scheduler jobs should be owned by a dedicated and minimally privileged Oracle database account and if OS interaction is necessary, they should similarly connect to the OS using a dedicated and minimally privileged OS account (and not the Oracle software owner account).

Set Security Initialization Parameters

When asked to look at initialization parameters affecting security, DBAs often focus on parameters affecting password case sensitivity and auditing. Other parameters are often overlooked, such as:

```
SEC_MAX_FAILED_LOGIN_ATTEMPTS
SEC_PROTOCOL_ERROR_FURTHER_ACTION
SEC_PROTOCOL_ERROR_TRACE_ACTION
SEC_RETURN_SERVER_RELEASE_BANNER
```

The default values for these parameters are set reasonably well since Oracle 11g (but were less optimal for Oracle 10g). However, it's important to recognize that the default values for these parameters change between Oracle versions. And not just between major versions such as Oracle 10g to 11g but between smaller increments such as going from 12.1.0.1 to 12.1.0.2.

These parameters should all be considered when defining corporate or departmental security policies and DBAs should monitor them during upgrades to ensure that they continue to conform to their policy defined values.

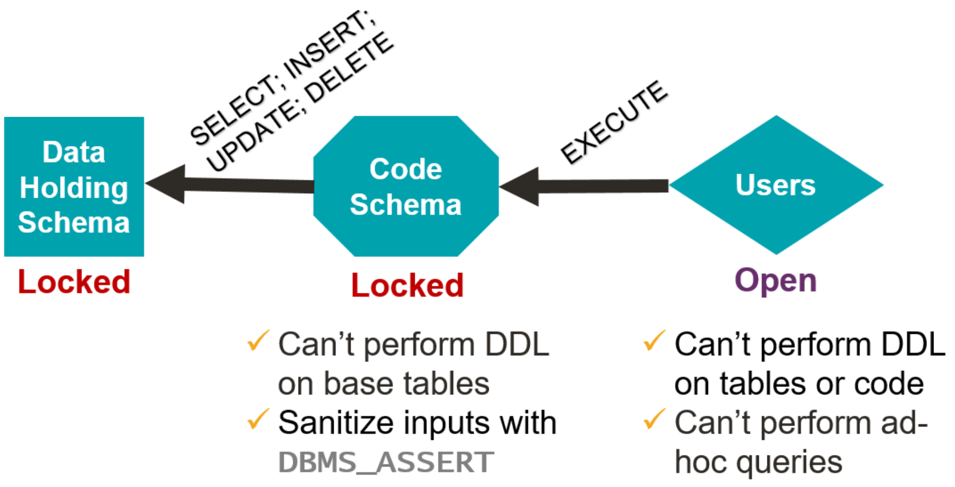
Additional security parameters are defined in the `sqlnet.ora` file. For example, in Oracle Database 12c the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` and `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameters should be defined and set according to internally agreed upon values. These parameters affect which password hash versions Oracle creates and records in the data dictionary.

Consider Secure Application Design

If developing applications in-house, consideration should be given to the optimal security architecture. Further such configurations and architectures should be properly documented in internal "development standards documents" and be enforced.

One of the securest design architectures is a 3+ schema model where:

- A "Data" schema holds all application data (tables and indexes, etc).
- An "Application" schema holds PL/SQL objects (packages) that query and update that data.
- A "User" tier of schemas which executes the PL/SQL objects in the application schema to run the application.


SELECT; INSERT;
UPDATE; DELETE
EXECUTE
Data Holding Schema
Code Schema
Users
Locked
Locked
Open
✓ Can't perform DDL on base tables
✓ Sanitize inputs with DBMS_ASSERT
✓ Can't perform DDL on tables or code
✓ Can't perform ad-hoc queries

Through a multi-tier model such as this. end users cannot execute ad-hoc DML or DDL against the raw data. And the schemas which hold the data objects and PL/SQL packages can be locked.

For additional security and to minimize the chance of SQL Injection type of attacks, the `DBMS_ASSERT` package can be used in the PL/SQL interfaces to sanitize input fields.

Use Role Security

A common request to DBAs is how can they prevent "*application bypass*"? Meaning how can they prevent Developers or even end users from connecting to the databases using an unauthorized tool (such as SQL Developer or even MS Excel) bypassing the application tier.

There are numerous ways to accomplish this though none of them are completely fool proof. However, one simple technique that usually provides an adequate level of protection is to use "role security".

Roles can be secured either using a password (meaning that the password must be supplied to make them active) or via some PL/SQL logic. In either case, as the end users wouldn't know the role password, or satisfy all of the conditions of the application logic (i.e. values obtained from `SYS_CONTEXT(USERENV)`), they would still be able to connect to the database using the unauthorized tool but would have minimal capabilities once connected due to the roles being inactive.

Listener Protection

The listener is often overlooked when it comes to database security. However, there are several areas where the listener can increase security above and beyond the listener password.

First of all, the listener can be configured for "valid node checking" meaning that connections can only be made from a list of invited servers. Or conversely are blocked from a list of non-permitted servers. This can be valuable in protecting unintentional connections to production from non-production tiers. For example, a DBA may refresh a test or development database from a production database (which has database links to other production databases) using a RMAN duplicate command but forget to update the associated database links. Hence the database links now exist in the test/dev database pointing back to another production database.

A few other more advanced listener parameters worth at least considering are the `RATE_LIMIT`, and `CONNECTION_RATE_<listener>` parameters.

Protect at the OS Level

Reviewing database security is often focused on what's happening inside the database or how the database is accessed. However, it's critical to also consider data "outside of the database". Backups are a common example of data that resides outside of the database and is often not ideally secured.

Common mistakes with backup handling include writing to NFS shares where the UID or GID mappings may not match. Or even simpler, not properly managing the file permissions on RMAN backup or Data Pump export files. It is important for DBAs to keep in mind that in many cases, if a bad actor can access backups, they can likely access the data (unless the backups are encrypted which requires the Advanced Security Option and hence is not commonly used).

Finally, the DBFs themselves must be properly secured at the OS level as data can be accessed directly from the DBF files (if not encrypted). Tools even exist to allow unloading of Oracle data directly from DBF files without requiring an Oracle instance.

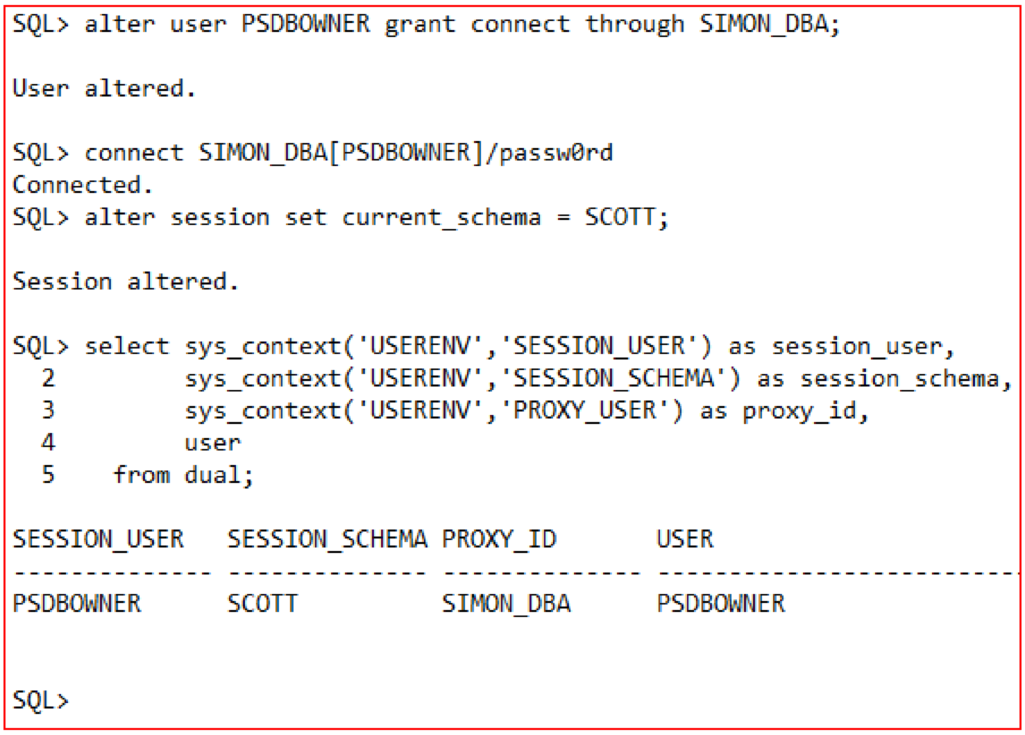
Use Proxy Authenticated Connections

One of the most powerful under-utilized features of the Oracle database is the "Proxy Authenticated Connection". This technology has existed since 10gR1 yet many DBAs are unware of it and even if they are, few often use it.

The Proxy Authenticated Connection has several capabilities (including database role restrictions) but the concept in its simplest form is that users (with the necessary privilege) can "***connect to a specified user account but using <u>their own</u> credential***". In other words, connect to a database account without knowing that account's password – only using their own password.

This is different to and more powerful than the "`ALTER SESSION SET CURRENT_SCHEMA`" command which only changes the default namespace for object name resolution.

An example for illustration:



```
SQL> alter user PSDBOWNER grant connect through SIMON_DBA;

User altered.

SQL> connect SIMON_DBA[PSDBOWNER]/passw0rd
Connected.
SQL> alter session set current_schema = SCOTT;

Session altered.

SQL> select sys_context('USERENV','SESSION_USER') as session_user,
  2         sys_context('USERENV','SESSION_SCHEMA') as session_schema,
  3         sys_context('USERENV','PROXY_USER') as proxy_id,
  4         user
  5    from dual;

SESSION_USER   SESSION_SCHEMA PROXY_ID       USER
-------------- -------------- -------------- -------------------------
PSDBOWNER      SCOTT          SIMON_DBA      PSDBOWNER


SQL>
```

Further, Proxy Authenticated Connections are completely audited in the audit trail and are identifiable in operational dynamic performance views such as `v$session`.

Among the many benefits of using Proxy Authenticated Connections is that application schema passwords no longer need to be shared with DBAs and/or Developers. Many organizations face the challenge of the complexity of changing passwords to key schemas when staff with knowledge of those passwords leave. Proxy Authenticated Connections mitigate this by not requiring that staff to have known those sensitive passwords in the first place.

Regardless of which of the many benefits of this technology is most advantageous to a particular individual or organization, this is a tremendously powerful functionality of the Oracle database and one that DBAs and Developers should be using on a daily basis!

Contact address:

Simon Pane
Pythian
Calgary, Alberta, CANADA

Phone: +1 613-565-8696
Email pane@pythian.com