VISCOSITY
NORTH AMERICA

On the Sixth Day of 18c/19c, Viscosity Gave to me...

# Security

**December 15, 2020**

One of the often underappreciated, but crucially important features of any database management system is that of database security. The rationale is simple: the contents of the database are among the highest value targets in an organization to a threat actor and can contain a wealth of information including basic confidential employee information, legally protected personally identifiable information or even proprietary intellectual property of the organization. While a thorough treatise on properly securing an Oracle database is beyond the scope of this article, Oracle 18c/19c does provide some features that work towards inherently increasing database security by reducing the attack surface.

## Technology Overview

Database: 18c / 19c

Company: Oracle Corporation

Topic: Security

## Viscosity can help with any of your Database Upgrade needs

Viscosity has performed numerous zero-downtime database migrations and upgrades over the years and has a proven track record with business critical and mission critical databases.

Viscosity's Database Migration & Upgrade Services can plan, upgrade, validate and migrate all database content - quickly and effectively with our automated approach and proven methodology.

Learn more about how you can maintain and maximize your investments at viscosityna.com or email us at hello@viscosityna.com.

**www.viscosityna.com**

## The Problem

Historically speaking, Oracle has not differentiated between the idea of a *user account* (accounts used to establish a connection to the database) and an *application owner/schema account* (an owner of the collection of tables, indexes, and other objects used to store the database data) except in a purely logical sense. While certain accounts were used as the ownership accounts for these database objects, they were identical to user accounts and each had their own password that could be used to login to the account. Over the evolution of the Oracle RDBMS, the number of these types of accounts has continued to grow and current releases have in excess of 30+ individual, distinct accounts (not including those created by third-party applications or individual sites for their own custom development). Breaching one of these accounts meant complete control of the database objects stored within that schema. Even worse, if the account had been setup without adhering to the principle of least privilege (DBA role, elevated system privileges, use of ANY privileges, etc.) then the impacts could be more far reaching, not contained to only the impacted schema. Unfortunately, that scenario is far more common than it should be.

Up until recently, the toolset to deal with this situation was limited to a handful of solutions; such as restricting the password to a trusted administrator and/or setting it to an incredibly complex password value, designed to thwart casual guesses. However, none of these solutions removed the core issue, which was that the schema was a login account that was still needed to make changes to the schema objects, preventing it from being permanently locked. Moreover, password protections at the database could be easily misconfigured to disable password locking, password expiration, and password lockouts leading to accounts whose passwords were set once and never changed – or even left at their well-documented default values. These became well-known vulnerability points that could be quietly attacked and used to gain an initial foothold in the database.

## The Solution

Beginning in Oracle 18c, database accounts may be setup as *Schema Only* accounts in addition to traditional user accounts. The Schema Only account allow accounts to be created in the database which do not have a password associated with them, and, thus, cannot be logged into. This is done by the use of the NO AUTHENTICATION command syntax for the CREATE USER command and is able to be done for both administrative and non-administrative accounts (but only in database instances, not ASM instances).

Consider the following example:

```
SQL> create user normal_user identified by C0mpl3x#Passw0rd;

User created.

SQL> create user hr_schema no authentication;

User created.

SQL> set lines 1024
SQL> col username format a20
SQL> select username,
  2         account_status,
  3         authentication_type
  4  from dba_users
  5  where username in ('NORMAL_USER','HR_SCHEMA')
  6  order by 1
  7  /

USERNAME             ACCOUNT_STATUS                   AUTHENTI
-------------------- -------------------------------- --------
HR_SCHEMA            OPEN                             NONE
NORMAL_USER          OPEN                             PASSWORD
```

Above, two user accounts have been created: *normal_user* which can be authenticated to using the specified password to logon to the database, and *hr_schema* which is created as a user account on the database, but has no associated password (a Schema Only account). This configuration can be confirmed by viewing the AUTHENTICATION_TYPE column of DBA_USERS, which shows that *normal_user* has a traditional PASSWORD authentication mechanism, while *hr_schema* has NONE associated with it.

Without a password associated with the account, the *hr_schema* account serves only as a repository to contain databases objects (for example) and cannot be logged onto directly.

Of course, schema accounts do not (and never have) existed in a vacuum, and database objects have to be modified, indexes have to built/rebuilt, and stored objects have to be created. Obviously, this can be done by someone using a variety of DBA or ANY privileges, but those privileges have the potential for abuse; as they are not narrowly focused for proper segregation of duties and principles of least privilege used. Thankfully, there are better tools to accomplish this while restricting access to only the specific schema.

Oracle has had the ability to use a *proxy connection* for several versions now, and so it's a trivial matter to setup our *normal_user* account to login to the *hr_schema* account via this mechanism.

```
SQL> alter user hr_schema grant connect through normal_user;

User altered.

SQL> connect normal_user[hr_schema]/C0mpl3x#Passw0rd@PDB1
Connected.
SQL> sho user
USER is "HR_SCHEMA"
```

This is conceptually similar to the *sudo* utility on a Unix/Linux system. The ALTER statement modifies the schema only account (*hr_schema*) and allows a specified user account (*normal_*user) to connect, by authenticating with their own individual password. In this case, the CONNECT command establishes a login to the bracketed target account name (hr_schema), by authenticating to the specified account (normal_user) as usual. However, using this specific syntax, when a connection is established to the database, does so as the schema account and not the standard user account (as demonstrated with the SHO USER command).

The ability to make this connection can also be removed by changing the GRANT in the aforementioned command to a REVOKE:

```
SQL> alter user hr_schema revoke connect through normal_user;

User altered.
```

However, what about specific cases in which direct access is needed to make schema changes, because proxy authentication is not viable? In those cases, the schema only account can be temporarily set to a password authenticated account, and then converted back to schema only account once the work is completed.

```
SQL> alter user hr_schema identified by T3mp0rary#ComplexPassw0rd;

User altered.

SQL> set lines 1024
SQL> col username format a20
SQL> select username,
  2         account_status,
  3         authentication_type
  4  from dba_users
  5  where username in ('NORMAL_USER','HR_SCHEMA')
  6  order by 1
  7  /

USERNAME             ACCOUNT_STATUS                   AUTHENTI
-------------------- -------------------------------- --------
HR_SCHEMA            OPEN                             PASSWORD
NORMAL_USER          OPEN                             PASSWORD

SQL> alter user hr_schema no authentication;

User altered.

SQL> set lines 1024
SQL> col username format a20
SQL> select username,
  2         account_status,
  3         authentication_type
  4  from dba_users
  5  where username in ('NORMAL_USER','HR_SCHEMA')
  6  order by 1
  7  /

USERNAME             ACCOUNT_STATUS                   AUTHENTI
-------------------- -------------------------------- --------
HR_SCHEMA            OPEN                             NONE
NORMAL_USER          OPEN                             PASSWORD
```

For Oracle 19c, a few further enhancements of the Schema Only accounts have also been made:

- Most of the accounts furnished with the Oracle database (ORACLE_MAINTAINED='Y') have been changed to use Schema Only accounts. In previous versions, these accounts were still password managed accounts.

**[Oracle 18c]**

```
SQL> select authentication_type, count(*)
  2  from dba_users
  3  where oracle_maintained='Y'
  4  group by authentication_type
  5  order by 1
  6  /

AUTHENTI   COUNT(*)
-------- ----------
NONE        5
PASSWORD   30
```

**[Oracle 19c]**

```
SQL> select authentication_type, count(*)
  2  from dba_users
  3  where oracle_maintained='Y'
  4  group by authentication_type
  5  order by 1
  6  /

AUTHENTI   COUNT(*)
-------- ----------
NONE       31
PASSWORD    5
```

- Prior to Oracle 19c, it was not possible to convert accounts with administrative privileges (those stored in the password file – SYSDBA, SYSOPER, SYSDG, etc) to Schema Only accounts. However, with Oracle 19c that is now an option.

**[Oracle 18c]**

```
SQL> select username, sysdg from v$pwfile_users where username =
'SYSDG';

USERNAME   SYSDG
---------- -----
SYSDG        TRUE

SQL> select username, authentication_type from dba_users where
username = 'SYSDG';

USERNAME   AUTHENTI
---------- --------
SYSDG        PASSWORD

SQL> alter user sysdg no authentication;
alter user sysdg no authentication
*
ERROR at line 1:
ORA-40367: An Administrative user cannot be altered to have no
authentication type.
```

**[Oracle 19c]**

```
SQL> select username, sysdg from v$pwfile_users where username =
'SYSDG';

USERNAME   SYSDG
---------- -----
SYSDG        TRUE

SQL> select username, authentication_type from dba_users where
username = 'SYSDG';

USERNAME   AUTHENTI
---------- --------
SYSDG        PASSWORD

SQL> alter user sysdg no authentication;

User altered.
```

```
SQL> select username, authentication_type from dba_users where
username = 'SYSDG';

USERNAME   AUTHENTI
---------- --------
SYSDG        NONE

SQL> select username, sysdg from v$pwfile_users where username =
'SYSDG';

USERNAME   SYSDG
---------- -----
SYSDG        TRUE
```

## Considerations for Upgrades

One thing to keep in mind during an Oracle 19c upgrade is the status of the default Oracle accounts. Any of these accounts which are in an *EXPIRED* and *LOCKED status* will be converted to a Schema Only account at the end of the upgrade process. This can be avoided by simply changing the password on the default accounts to a sufficiently strong password. Before the upgrade starts or corrective actions begin, convert the account back to password authentication once the upgrade has been completed.

## Summary

The proliferation of database schema accounts which serve only to hold database objects has always represented a potential vulnerability and area of exposure for Oracle databases. Now, by providing the ability to effectively distinguish between types of accounts (user login and schema), Oracle has taken a huge step forward in improving the security posture of one of the customer's most critical assets. While not as flashy as some other features, it represents a welcome evolution for long-time database administrators who have long had to manage passwords for an increasing number of accounts that were seldom (if ever) used for purposes other than object ownership.

One of the sayings we have at Viscosity is our customer's, "have four aces in their pocket". Over the next 6 days, the talented staff at Viscosity along with our Oracle ACEs will address more Oracle Database 18c and 19c new features. Continue to join us next year, as we continue our Oracle Database 19c hands-on-lab workshops.

Happy Holidays!