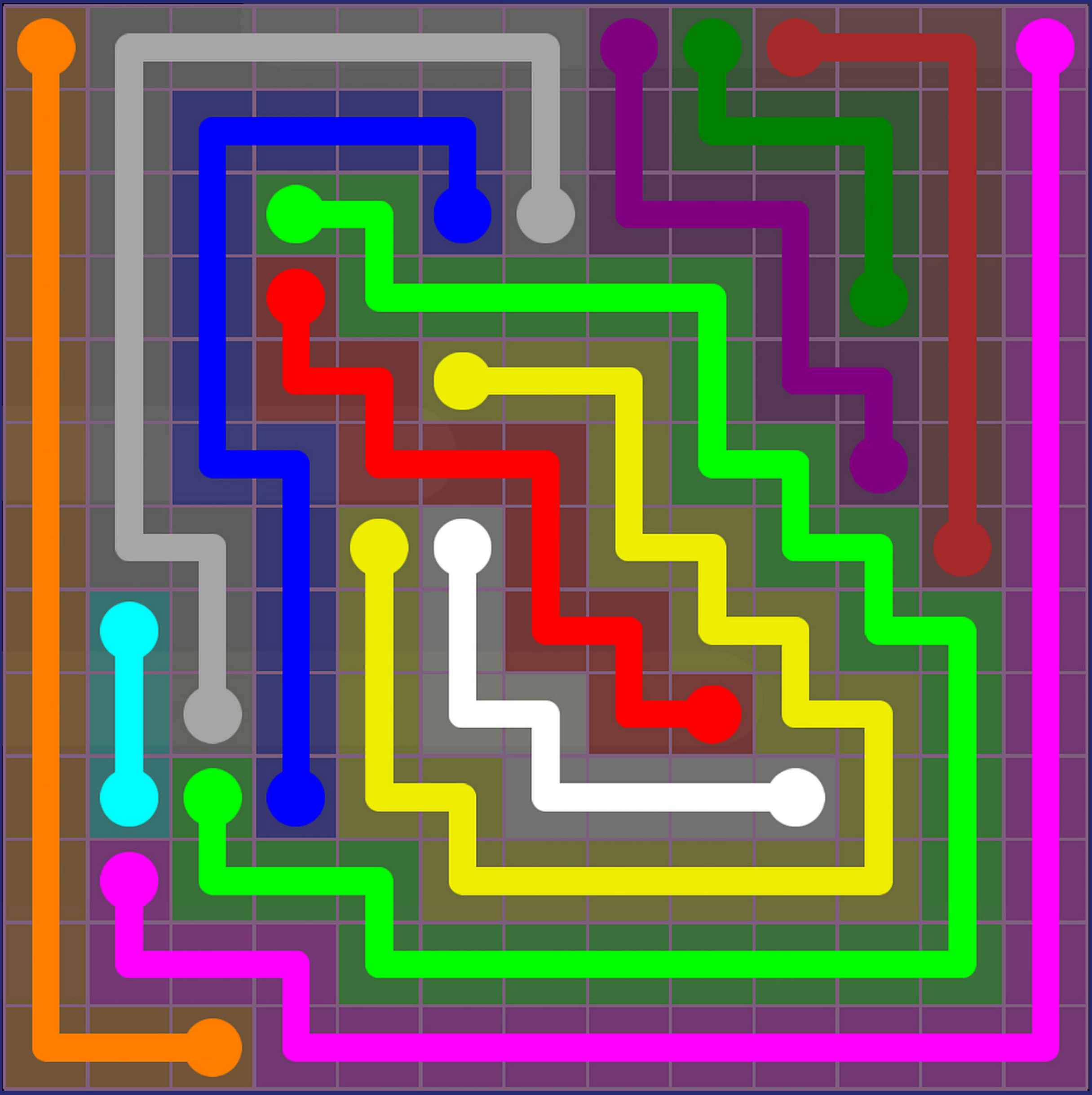
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 29, No. 4 · NOVEMBER 2015 $15



YesSQL(T)

With Carlos Sierra and Mauro Pagano.

See page 4.

Brian's Notes

Oracle Database 12c Security.

See page 7.

The Easy Button

Packaged Analytics.

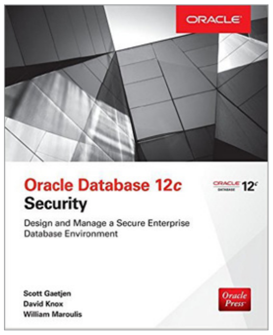
See page 20.

Much more inside . . .

Oracle Database 12*c* Security

A Book Review by Brian Hitchcock

Details



Author: Scott Gaetjen, David Knox, William Maroulis

ISBN-13: 978-0071824286

Pages: 560

Year of Publication: 2015

Edition: 1

List Price: $70

Publisher: McGraw-Hill Osborne Media

Summary

Overall review: A very good way to learn a lot about all the many security features and options available for the Oracle 12*c* database. Most of the material applies to recent pre-12*c* versions as well. You don't have to know a lot about 12*c* to understand what is being discussed, although it would be easiest if you knew about container and pluggable databases before you get started. The authors do explain basic 12*c* features as they go along, but if it is the first time you are hearing about containers, it might be a little confusing at first.

Target audience: DBAs, application developers, and security specialists.

Would you recommend this book to others?: Yes

Who will get the most out of this book?: Anyone who is involved with security issues for systems that include an Oracle database.

Is this book platform specific?: No

Why did I review this book?: I saw a copy at a recent NoCOUG conference and liked the table of contents.

Introduction

The overall layout of the book is broken down into three sections: Essential Database Security, Advanced Database Security, and Security and Auditing for the Cloud. For Essential Database Security we are told that user database account mapping is one of the most important decisions we will make, and this will be discussed later. Mapping refers to how users will connect to applications and databases to do their jobs without causing security issues. Most users connect to applications through a browser, and the application connects to the database, which means that for most users, their identity is not known to the database. The terms "access control," "authorizations," and "privileges" will be covered, as they describe the foundations of database security. A new term to me, "entitlement analytics," describes the ways we can examine which privileges users have and how they were assigned. Another new term for me is "Oracle Real Application Security," described as a declarative security model for applications.

For Advanced Database Security, we will learn about Virtual Private Database (VPD), data redaction (DR), and features in 12*c* that help us find sensitive data in our databases. Oracle Label Security (OLS), Oracle Database Vault, and Transparent Data Encryption (TDE) will also be discussed. If you feel your life lacks acronyms, relax, you will have many more to deal with as we move along.

For Security and Auditing in the Cloud, we will learn about what the term "cloud" means from a security perspective and about meeting compliance regulations, and review an approach to setting up all of this database security for yourself.

Chapter 1: Security for Today's World

The first thing we are told is that securing your database can be an overwhelming task, and I agree. If, as you read this book, you feel overwhelmed, you're paying attention. Security is a complicated subject, and it would be even more complicated to try to implement all the things that will be described. We are told that we will discuss how to secure the Oracle database in general and then cover the new features that Oracle has developed specifically for Oracle Database 12*c*. The authors also point out that while they focus in on the Oracle database, we need to look at the entire system within which the database lives. Again, I agree: it is so easy to slip into a detailed discussion of any given Oracle product when, in reality, all of the Oracle products you are dealing with live in an overall system that has many other components and technologies from multiple vendors. This points out just how complicated security issues really are. It is good to learn about Oracle database security, but we need to be careful: it is easy to get lost in what we know best and to ignore the complexity we don't understand as well. Another good point is offered next: many times we assume that our infrastructure will prevent any and all bad actors from getting into our environments. This assumption is identified as among the worst mistakes we can make. We are encouraged to assume that no matter what we do, someone can still get in, and bad things can still happen. This leads to another important point, perhaps my personal favorite, that the insider threat is always present. For all of the new and old technologies we have for securing the database and other components, how can you prevent someone like me, who has full access to the database, from simply doing bad things, ones that you may never even know I have done? We are told that what we learn in this book will protect against insider threats, but I remain skeptical.

Security issues have changed just as quickly as the Oracle database and we are told that many things we do now in the name of database security are based on security concepts that are roughly 15 years old. I don't disagree, but I think the choice of 15 years is interesting. Not to worry: the 12*c* database has lots of new stuff that will replace all the old stuff.

The evolution of security technologies is covered next, starting with the Evolving Four A's, which are authentication, authorizations, access, and auditing. I was waiting to see if it would come up, and it does, that multi-factor authentication is described as essential in modern systems. I have been required to use two-factor authentication for years at work. I have also seen this become available for online brokers, and it is an option I use on my WordPress website. I'm glad to see this book really does seem to be up to date. The discussion centers on users interacting with the database; this has changed, as most users connect to applications that use connection pools to access the database.

Chapter 2: Essential Elements of User Security

The focus shifts to discuss the importance of database accounts and how to manage this efficiently in 12*c*, although it is largely relevant to previous versions as well. Since 12*c* is all about consolidating many databases into one multitenant database, we now have even more sensitive data stored in one place. How you map users to accounts is covered in detail. Identification and authentication are reviewed. Identification covers user-supplied information, such as usernames, and what is called technological identification, such as biometrics.

Once a user has been identified, the critical step of authentication is next, and multifactor authentication is said to be best practice. This means more than one, perhaps a password and a token or biometric. Database account types are covered, including end users, connection pools, non-person entities (SkyNet?), application schemas, operational DBAs (do we still have them?), and application DBAs. The 12*c* database architecture is described to show that the account types haven't really changed. Further, 12*c* supports all the mechanisms from previous releases. There are new Administrative Privileges in 12*c*, such as SYSDG for Dataguard, SYSKM for key management, and SYSASM for ASM administration. These are to support separation of duty, where different people are responsible for different aspects of database administration so that no one person has all the privileges. In the text the phrase "another set of administrators" is used, which I find amusing, because in my group, we don't have enough bodies to cover the shifts, let alone staff another set of administrators. There are new system accounts associated with these new privileges and new internal tables, such as V$PWFILE_USERS, where we can see which account has which privilege. We are told that it is best practice to lock and expire the password for these accounts and create separate named accounts for individual administrators. I haven't seen this done in the wild. Further, the authors tell us that the operating system for the database is assumed to be secure. I guess I have to assume that, but I was looking for more details about this.

Oracle OS authentication is discussed, including creating users and groups for each admin privilege. It turns out the pluggable databases (PDBs), which are the individual databases you consolidate into one container database (CDB) in 12*c*, require a different process for OS authentication. You have to connect to the container database first and then alter your session to the pluggable database where you want to work. You can't use OS authentication to connect directly to a PDB.

Many options are discussed and many syntax examples are given.

New in 12*c* are common accounts: database accounts that are present in all the PDBs that have been consolidated into the CDB. How to create these accounts and the issues this brings up are covered. For example, common account names must start with "C##," which will take me a while to get used to. It looks like some kind of coding prefix or some kind of wildcard that must get replaced later. How to create accounts in each PDB is discussed as well as some issues with grant and revoke. Options are presented for managing local database accounts and common accounts. This includes passwords and maximum failed logins.

Chapter 3: Connection Pools and Enterprise Users

As mentioned earlier, most clients connect to applications through a browser. For performance reasons we don't want to have users creating their own database connections, so we create connection pools where persistent database connections are used by application users as needed. Each time a database connection is created or destroyed it takes system resources. The application users connect to the application, which connects to the database through the connection pool. This creates the issue that the database doesn't know anything about the end user. This affects security auditing and performance. Connection pools are discussed and the security issues described. Database security can't be based on information specific to the application user. Within 12*c* Oracle offers external authentication mechanisms, including Proxy Authentication, Enterprise User Security (EUS), Kerberos, and Real Application Security (RAS).

Proxy Authentication is part of the Oracle Call Interface (OCI) connection pool. This mechanism starts with creating end user accounts that have an "impossible password," which is an undocumented Oracle feature. My training has been that we don't use undocumented features, but the authors tell us to and provide examples of the required syntax. The idea is that the end users won't have a password in the database at all, but will use a proxy account called APP_POOL to connect. The end user doesn't know the APP_POOL database account password. The example continues showing how this works for application servers, and includes using LDAP.

Enterprise User Security (EUS) uses LDAP to authenticate and authorize database users, which means database users are created in LDAP, not in the database itself. This centralizes user administration in LDAP, supports Single Sign On (SSO) and is more efficient for large numbers of end users. Directory Services are discussed as well as the Oracle components of Identity Management, such as Unified Directory, Internet Directory, and Directory Integration Platform. I promised you lots of acronyms and here they are! The process to set up EUS is shown, including screenshots. For converting an existing database, the User Migration Utility is provided for bulk migration of existing database users. EUS has a performance concern as each user has to be authenticated against LDAP versus the local database. It is also pointed out that this creates a single point of failure; if LDAP is down, the whole database environment is inaccessible.

Kerberos Authentication is covered with a simple example and Real Application Security is covered in Chapter 6.

Chapter 4: Foundation Elements for a Secure Database

Now we look at database privileges and roles, starting with the terms "access control," "authorization," and "privilege." Access control is controlling user access to a resource and access control lists (ACLs) are an example of this. An ACL could be set up to describe a security policy.

Authorization represents the connection between a security policy and the specific privileges a user has. Authorization gets translated into privileges for users.

Privileges are permission to execute an action in the database. Privileges get a lot of coverage here. 12*c* has system and object privileges, as we would expect from previous database versions, and introduces intra-object privileges, which are discussed in Chapter 5. A diagram illustrates the relationship between a security policy, access control, authorization, and privileges for two different users.

System privileges can affect the entire database, which means the container database and all the pluggable databases or can be limited to one specific PDB. This is an example of new considerations we need to be aware of for 12*c*. Examples of the various types of system privileges are covered, along with how to view them in the database, including in the SYSTEM_PRIVILEGE_MAP view.

Object privileges and how to view them is next, followed by column privileges. New to 12*c* is the ability to grant INSERT, UPDATE privileges to individual columns on a table. This is useful with other database security features such as Virtual Private Database (VPD) and Oracle Label Security (OLS), both of which have separate chapters later in the book. The SQL needed to set up column privileges is provided for the mythical EMPLOYEES table. Clearly the authors have a dry sense of humor, as the example shows managers authorizing raises for the employees. Does this really happen? Perhaps I've worked at Oracle too long! Synonyms in 12*c* can have user-friendly names for objects, and SQL is shown for this new feature. The example also points out some subtle security issues that can be created if you aren't careful with how you implement synonyms, specifically how this can affect previously granted privileges when the synonym is dropped. Similarly, the possible interactions between system and object privileges are illustrated using the ever-popular SELECT ANY TABLE privilege. If this is granted to a user, it doesn't matter if you revoke SELECT on a specific table; the user still has access. The WITH GRANT OPTION gets the same treatment as a possible security concern. We are told that good database security requires that we understand how granting and revoking database privileges work. Not surprisingly, roles are addressed next. Roles are shown to be more efficient for managing database privileges for large numbers of database users. We see an example where a specific grant takes effect right away, while a role assignment requires logging out and back in. In 12*c*, roles can be created in the current container (i.e., the current pluggable database), or for all containers (i.e., the single container database and all the pluggable databases). Like common users, roles that affect all databases must have names that start with C##.

Selective privileges are explained, and an example is given where a user can only access the database through an application and not directly. The application can be written to enable the role for the user as needed and then disable it. The user can't access the database directly with the role using something like SQL*Plus. The SQL to set up password-protected roles is shown. This chapter also covers global roles that use LDAP and the confusion that can come up when this is used in addition to some local database roles.

This chapter ends with advice on how to use roles wisely, having too many, how to name them, and some issues around role dependencies.

Chapter 5: Foundational Elements of Database Application Security

While application security may be built into the various layers of an application in JavaScript, Java Servlets, and Java Authentication and Authorization Service (JAAS), among others, the question is posed as to where is best to put the code for security. We are told that the answer is to enforce security as close to the data as possible. An example is that security code in a web application doesn't help if users can access the database directly using SQL*Plus. Any security must be enforced no matter how the user connects to the database. Keeping the security out of the way of the users is described as security transparency, meaning that users are not aware of how security is being enforced, that it doesn't interfere with their work to the point that they demand it be removed.

As a user connects to an application, your security policies will need to store information about the user, and while a database table is one way to go, we are told about the better, faster, more secure way, which is to use an application context. We see a detailed diagram showing how the application context is stored in memory for both a dedicated and a shared server session mode. The context stores name-value pairs that can be queried and changed by users and applications. This feature was created to speed up fine-grained access control (FGAC), part of the virtual private database. The three types of application context are described: Database, which holds data for each user's database session and has a default namespace of USERENV; Global, which is shared among multiple database sessions; and Client, for OCI context values. Examples of what is stored in the USERENV namespace include client IP address, session ID, and many others. All of these values can be accessed using the SYS_CONTEXT SQL function. I found this interesting: one of the optional parameters to this function is the length value. You use this to truncate a value to the specified length, and this is used to verify the data to prevent an attack based on buffer overflow.

Creating an application context requires the CREATE ANY CONTEXT privilege; when you create a context, you need to assign a namespace manager such as a PL/SQL package, used to set the context values. Many other aspects of managing application contexts and using them to improve security are covered, all of which were new to me. For example, when you have connection pools, you have to set and reset the application context as different end users connect to the database. Since I am not an application developer, I don't see these features often.

Chapter 6: Real Application Security

Real Application Security (RAS) is new feature in 12*c*. I had not seen this before, and it turns out that this feature was created specifically to provide a common, declarative security model that can be shared among enterprise applications. This provides better performance because it avoids the overhead of switching database connections to switch between applications. It also deals with the problem of not having the end-user identity information due to using a shared connection pool. This feature was designed to work with Oracle Fusion Middleware (FMW) security as well as any Java-based database application. This means RAS supports FMW products such as Identity Management (IDM), SOA Suite, WebCenter, and Application Development Framework (ADF).

Real Application Security is based on application users and roles, and a lightweight session model, each of which are discussed in detail in this chapter. There are many new things to learn about this, such as using the XS_PRINCIPAL package to create direct login application user accounts, and several examples of the SQL used to manage these accounts are given.

RAS roles and how they are integrated with standard database roles is shown, and a diagram shows how RAS application users, roles, and standard database roles are interrelated. RAS lightweight sessions are created once a traditional database session has been created, and then used to create multiple long-running application sessions, each with its own set of roles, privileges, and context. This is done to support efficient switching between sessions while maintaining needed end-user and application session information.

This leads to coverage of the Java class oracle.security.xs.XSSessionManager, which is used to manage the lightweight sessions in Java. This is one of several sections that cover the details of RAS. Much of this is not familiar to me, as I usually deal with the database and administering FMW, and not application development. A lot of code is shown, which I assume would be useful to developers. More detail is given for server-side event handling and namespaces, and then session performance is discussed. Code is shown that demonstrates that RAS provides a 60% improvement in the time needed to attach and detach from a session. Here we find that RAS was designed and built by the Oracle Database Security team to support the performance, auditing, and session context needs of Oracle Fusion Applications. This helped me understand that RAS wasn't created for me to use but was created to support the much bigger needs of Fusion Applications. That alone made this chapter worthwhile to me. Sometimes I can't really see why some of features of a new database version are so important, but like RAS, it turns out some features were created to support other areas of the overall Oracle technology stack.

This chapter ends with more details of RAS, including privilege management, data security, security classes, and data security policies. This chapter has a lot of detail that would be useful for application developers.

Chapter 7: Controlled Data Access with Virtual Private Database

The previous chapter advocated the use of RAS, which is new to 12*c*. This chapter is for all those environments that are not on 12*c*—at least not yet—and describes the security framework provided for database 8*i* through 11*g*, which is Virtual Private Database (VPD). First up is an introduction to how VPD works, which explains that VPD is implemented using the Row-Level Security (RLS_ package DBMS_RLS), which restricts which rows or columns SELECT or DML statements can access. The restrictions are based on a security function, which is PL/SQL written based on your security policy. Note that 12*c* has VPD as well as RAS, and that in 12*c* VPD is limited to working on the current database; it can't work across multiple pluggable databases.

VPD works by registering a table or schema with a policy function and the type of SQL statement that will be protected. When the SQL statement is executed, the policy function is evaluated and returns a predicate that is appended to the SQL statement's where clause. Note that the execution of the policy function is done first, before the original SQL statement, and the result used to modify the statement, which is then parsed and executed. For example, a VPD policy could look at a user's ID and limit the rows that user sees by adding to the where clause of the original SQL.

One of the benefits of using VPD is that it puts security very close to the data in the database, not in the application where it could be bypassed by a user that could connect directly to the database. It is important to note that while VPD may sound good and may be used simply to get started, it is easy to get lost in the process of defining your security policy requirements. It can be harder to define your security policies than it is to set up VPD.

The following sections describe the VPD components, the row-level and column-level fine-grained control options, and how to use VPD. The authors assert that many times the VPD security policies that are created are too restrictive. A very simple example that I would not have thought of is presented. If you have a table SALES_HISTORY, and the VPD security policy restricts users to only seeing the rows that represent their own sales, then users can't do a simple count of the total number of sales in the table. This is very good information. It is a simple example, but it really shows how easy it is to set up security policies that no one will be willing to live with.

A flowchart is provided to help assess what kind of VPD policy is best based on what you are trying to protect.

This chapter ends with an interesting comment that for any security feature, performance is always a concern. Since VPD adds to the where clause of each SQL statement, there will be a performance impact. We are told that VPD performance issues become SQL statement tuning issues.

Chapter 8: Essential Elements of Sensitive Data Control

The previous two chapters discussed how to limit a column from appearing in a result set using RAS and VPD, which is called static or full redaction. Here we see that in Oracle 12*c* you can set up partial or dynamic redaction that hides part of a result such as a credit card number using Oracle Data Reduction (DR). I get confused because DR to me will always be Disaster Recovery. This is part of the Advanced Security option and works with 11*g*R2 databases as well. Another feature of 12*c* is Transparent Sensitive Data Protection (TSDP), which can find sensitive data in your database. Part of Oracle Enterprise Manager Cloud Control (OEMCC) called Quality Management Data Discovery (QMDD) will analyze databases for sensitive information. With all these acronyms in place, we look at the challenges of protecting sensitive data: what data is sensitive, where it lives, and how to protect it.

Within TSDP we have new packages, DBMS_TSDP_MANAGE, which we use to define sensitive data by logical name, and DBMS_TSDP_PROTECT to define policies for DR or VPD. An example is shown, with screenshots, of using OEMCC to examine your database for sensitive information. The example is fine, but it assumes that sensitive data is going to be really obvious, like Social Security numbers, for example. If they are stored alone I guess that is okay, but how do we know there isn't sensitive data that would be much harder to define and detect? A lot of data that could be sensitive will not be so obvious. Configuring TSDP to detect sensitive data types is shown, which includes mapping to specific columns. How to create policies and map them to the sensitive data types is next, followed by setting up redaction. This uses the new package DBMS_TSDP_PROTECT, which provides procedures to enable policies to control how sensitive data will be displayed.

This chapter ends with a good example of the impact of all this security stuff, namely the need to redact bind variables in audit trails. I had not thought about it, but the value of a bind variable could be shown in an audit trail and that might need to be redacted. I think this chapter is interesting, as it shows how to set up a needed functionality, but I would like to see how this would be applied to a real enterprise application where the sensitive data is much more complicated than just Social Security numbers. I warned you about lots of acronyms, didn't I? This chapter really delivers on that front!

Chapter 9: Access Controls with Oracle Label Security

Oracle Label Security (OLS) controls row-level access to table data and is built on top of VPD, which has already been covered, and uses the DBMS_RLS package. Can you imagine what the PL/SQL manual for 12*c* must be like? The features that make OLS uniquely different from VPD and RAS are the declarative policies and the lack of technical programming, and it turns out we can use VPD, RAS, and OLS separately or together. A history is given of where all this came from, starting with the need for finer-grained access control than GRANT statements. Oracle consulting came up with Secure Access (SA), and the SA acronym still appears in some of the package names.

A functional example of OLS is given that shows exactly how OLS works using the SALES_HISTORY table. OLS works with the GRANT privilege to control access to each row of a table; you have to have access to the table before you get to the OLS restrictions. Each user has a label and each row in the table has a label, and OLS adds to the WHERE clause of the SQL to control each user's access to the rows in the table.

OLS and VPD are compared to answer the question of which is better and when to use each one. The answer is that OLS has all the features that have been built and tested for a security setup, whereas VPD requires that you set up and maintain your own code and other pieces of the installation. Since OLS has been tested, it may also make it easier to pass certain security accreditations. The OLS label component is shown to have three parts: levels, compartments, and groups, and we see that OLS allows for ten thousand levels for a single OLS policy. I can't really imagine how you would design and maintain such a complex set of policies, but 12*c* will support it. The row labels used by OLS have numeric values as does the user's label. Typically, when the user's label value exceeds that of the row, access is granted. You can also have lots of compartments and groups, and groups can be set up in a hierarchical structure. It is interesting to see how we display the labels using SQL, using the LABEL_TO_CHAR function. The many different types of user labels are shown. This could get very complicated, and I wonder how many customers really set all this up.

The processes of installing OLS and administering it are covered, including screenshots from the Database Configuration Assistant. Registering and enabling OLS has to be done in the root container first and then in the pluggable database, and the SQL for this is shown. You can use the LBACSYS database account for OLS administration, but as with other database features, we are told to lock this account and set up named administrator accounts, which are granted the LBAC_DBA role.

The chapter ends with a detailed example of setting up OLS for the SALES_HISTORY table. When I say it is detailed, I am not exaggerating: it really covers all the steps, to the point that I wonder again who actually does all this for all the tables needed for an enterprise application. The performance impact of OLS is also discussed, and table partitioning is recommended. The authors tell us that OLS and VPD are sometimes described as being "too complicated" to use. I can see why.

Chapter 10: Oracle Database Vault: Securing for the Compliance Regulations, Cybersecurity, and Insider Threats

DBV was introduced in 2005 to offer the features needed by businesses to deal with all the emerging cybersecurity threats. I noticed that the threat from internal employees was brought up, something that I think needs a lot more attention. It doesn't matter how good your security is if your own DBA is selling your data. We are told that DBV changes the way DBAs do their job.

First is a history of privileged accounts, specifically for the Oracle database, starting with the SYS user, which has all access—similar to the root user for Linux. This leads to the DBA role and then the need to audit these powerful users. We need to have these roles, but we need to be able to keep track of what they are doing.

We need SYS to own the core database objects and to do things like apply patches, so we can't remove SYS. But SYS presents a big security risk, so we need to control SYS, and this is one of the things DBV was designed to do. SYS can't get around the security provided by DBV. Security features of DBV are discussed, starting with multifactor authentication, which requires more than just a password to gain access. Conditional security provides context-based security, which can be seen as a conditional grant—access is only granted under specific conditions, such as where a user is connecting from. It also provides adaptive security to respond in near real time to threats, and separation of duty (SoD), so that no single person has all authorizations needed to perform administrative tasks. Conditional auditing provides fine-grained auditing, so that only what we decide needs to be audited is audited. The message here is that with DBV, security is not a rigid set of roles with a defined set of privileges but is much more responsive to the context of the request for access.

The components of DBV are discussed, primarily the declarative framework provided that transparently evaluates all SQL and determines if execution should be allowed. This is based on a set of tables in the DVSYS schema. A web-based interface within OEMCC (remember, that is Oracle Enterprise Manager Cloud Control 12*c*) can be used to set up the security policies for DBV or a PL/SQL package called DBMS_MACADM.

The DBV policies are specific to one container, the root database or one of the pluggable databases. Many more pieces of

DBV are described, factors to be considered when granting access, rules for decision points in the authorization process, realms for collections of objects that need to be secured as a group, command rules, and secure application rules. The process of configuring and enabling DBV is covered, with many SQL examples and the observation that DBV must be configured in the root container first before it can be used in a PDB. Some of the many PL/SQL packages and database views available for DBV administration are covered.

Finally, a detailed example of setting up DBV is presented, which covers a lot of ground. It is clear that implementing DBV is a large project.

Chapter 11: Oracle Transparent Data Encryption: Securing for the Compliance Regulations, Cybersecurity, and Insider Threats

Each of the many security components available in 12*c* provides features to deal with different security issues. Now we cover encryption, which helps with several specific issues. It happens regularly that an employee laptop with sensitive data is lost or stolen, and with it, a large amount of sensitive data. If the hard drive is encrypted, then the risk is reduced. Similarly, the data stored in the Oracle database is at risk anytime one of the hard drives is removed or a backup is made and stored off-site. If the data is encrypted when it is stored in the database, then we prevent access to the data when a hard drive is removed or a backup made.

The history of database encryption offered in the Oracle database is documented, going back to 8*i*. Transparent Data Encryption (TDE) was first seen in 10*g*R2 and was a big change. TDE is much more integrated with the database than previous offerings. Implementing encryption is done within the DDL commands used to define how columns or tablespaces are stored. Once implemented, the word "transparent" means that you don't have to encrypt or decrypt data, it is all done transparently without needing any changes to your applications. This is a big deal because security measures that get in the way are less likely to be used.

This chapter covers the basics of cryptography and some history of its use, going all the way back to the Roman Empire, and the reasons you don't want to set up your own encryption scheme. The things needed to encrypt data, a key and an algorithm, are illustrated. I had not previously heard that longer encryption keys may or may not make the encryption stronger. On television (I watch way too much TV!) they are always impressing us with how many bits are involved in the encryption key, as if that means anything to the viewers. The difference between symmetric and asymmetric encryption is explained, and for both, the issue is getting the keys to the right parties. This leads to a discussion of public key encryption, which does away with the key distribution issue but is slower. Public key algorithms are slower, so they are used to transmit the key needed for faster symmetric encryption that is then used to move the data. I found it interesting how the compromise between security and performance is handled.

A detailed example of encrypting credit card data in the SALES_HISTORY table shows how we could see the sensitive data using the strings command without ever needing to access the database, as well as what the sensitive data looks like after encryption. A great deal more detail about how all this works is covered. Note that TDE is only for data that is "at rest," which means data that is in the database. As data is selected or modified, it is decrypted and manipulated, and only when the data is returned to the database is it again encrypted. We are referred to the "standard network encryption feature" to deal with encrypting data that is moving through the environment. I found this to be one of the best chapters for me from a DBA perspective, because TDE is something I could see being used by many people since it doesn't require application development and can be set up entirely in the database.

Chapter 12: Audit for Accountability

We start with a statement that auditing is not exciting and that we all should do it but rarely do. I agree; I can only think of one workplace where I saw database auditing happening, and the only thing we did with the database auditing was delete it whenever it got close to filling up the assigned tablespace. I also agree with the next statement, that many times we don't know what to do with the audit records. This chapter explains why auditing is necessary and not that hard to do.

We start with the security cycle, which is described as a process that begins with prevention that includes access control; moves on to detection—detecting that however good your access control may be, it isn't perfect; and ends with response—what you do when you detect a security issue. I agree with the authors when they state that many times the focus is all on the prevention, on the thoroughness of the access control, as if that is all that is needed. Assuming no one will ever break in is not a good plan, especially when your own DBA could decide it was time for some off-the-books activity.

Auditing provides the detection phase of the security cycle. You can't have a response unless you detect the event. It is vital that you audit the right data, processes, and users, and you must review the audit records regularly. We are told that we must not look at auditing as overhead, something that negatively impacts performance without any benefit.

Several audit methods are reviewed, starting with two that are outside the database: infrastructure and application server logs, which come from network switches; firewalls and application servers; and application auditing, which is dependent on the auditing that the application developers decided to code into the application. Next are two methods that are inside the database: trigger auditing and database auditing. Trigger auditing is transparent to existing applications, which is one way to add auditing when it wasn't built into the application. Four kinds of database auditing are described: mandatory SYS auditing (MSA), traditional auditing (TA), fine-grained auditing (FGA), and Oracle unified auditing (OUA). Even more acronyms!

The advantages and drawbacks of each of these four options are explained.

The details of setting up database auditing are shown, and like some other 12*c* features we have seen so far, this has to be configured in the root container (CDB), and all the PDBs share the auditing setup. I'm not sure this will really work in an environment with many different databases serving widely different application, user group, and political agendas. The one database that needs the most auditing will force that level of detail on all the other databases, and there could be way too much auditing going on. The chapter also covers the options for setting up separate users to control and conduct auditing, further details on how to determine what to audit, and storage issues that arise from auditing.

Chapter 13: An Applied Approach to Multitenancy and Cloud Security

This final chapter offers to tie together everything that has been discussed in all the previous chapters, to provide a pragmatic approach to setting up a secure multitenant database system. This can also be applied to a traditional, non-multitenant database as well. We are reminded to apply only as much security as needed to meet compliance or regulation standards, and that we need to have multiple layers of security, each one making it less likely that anyone can get all the way through.

This pragmatic approach is broken into four sections: system baseline and configuration, installing and securing the application, data encryption, and locking down your system. Each of these sections has many detailed steps to follow. For example, under the system baseline section, we find steps covering personnel security, configuration management, equipment, virtualization, operating system, users, groups, and several others. This is a very detailed list of what should be done. I would have liked to see some discussion of how this process could be applied to an existing system; I think many of the detailed steps would be much more difficult to implement when many of the configuration decisions have already been made.

Under the section covering system baseline and configuration there is a list of steps to consider for installing 12*c* software. Here we find a good discussion of how to choose how many Oracle Homes you need. I had not thought about this. You can have a single Oracle Home for the 12*c* software for multiple CDBs and all the PDBs that will be contained in them, but you may want multiple Oracle Homes so that not all of the databases are affected by an outage. Patching comes to mind. If you have hundreds of databases, can you have all of them down at once to patch the single shared Oracle Home?

The level of detail included in these sections is impressive. I would recommend that you look at this last chapter if you are in any stage of planning a new installation or a security review for an existing system. I had not worried about the chain of custody for the patches applied to the database, but it is on the list of things to consider.

Another interesting section points out that availability should be part of your security plan because many attacks will simply try to put your online business offline. I was puzzled by the specific advice to have a backup site in Lake Tahoe, California. Seriously? Have you looked at real estate prices there? I don't know of many data centers close to the lake, but maybe that is a sign of just how secure they really are.

Conclusion

I enjoyed reading this book and I learned a lot about 12*c*. I do think this book adds to my feeling that the enterprise software we are all working with is too complex to be effectively supported. The Oracle 12*c* database has many cool features, but who has time to understand, configure, and manage all of them? Who has the personnel resources to set up all the different roles needed for true separation of duty, for example? This is why we still do what is quick and easy. I still access databases using "/ as sysdba," and I see many database users with more privileges than they absolutely need. We do these things because we are pressed for time, and customers do not want their next rollout delayed because an application user doesn't have the needed privileges on a table. I think this is the real security issue for all of our software systems, but no one really wants to talk about it. We don't know how to fix the real issue—that no one has the time and resources to really address security—so we focus on new features and hope for the best.

"We assume that our infrastructure will prevent bad actors from getting into our databases. This is one of the worst mistakes we can make. No matter what we do to prevent it, someone can still get in, and bad things can still happen. Furthermore, the insider threat is always present."

I'm surprised, although maybe I shouldn't be, at how much of this book talks about security features that can only be set up when designing and coding the applications that run against the database. This points out just how hard it is for a DBA to do much about security in real time. If the applications are sending data to anyone that asks for it, what can be done to the existing applications?

I want to be clear that none of what I'm saying is a criticism of the authors in any way. Their job was to communicate the security features of 12*c*, and they have done a great job of that. I'm also pleased that the performance impact of the various security features was explicitly brought up and discussed. Too often, new features are presented as magic cure-alls that have no impact. ▲

Brian Hitchcock works for Oracle Corporation where he has been supporting Fusion Middleware since 2013. Before that, he supported Fusion Applications and the Federal OnDemand group. He was with Sun Microsystems for 15 years (before it was acquired by Oracle Corporation) where he supported Oracle databases and Oracle Applications. His contact information and all his book reviews and presentations are available at **www.brianhitchcock.net/oracle-dbafmw/**. *The statements and opinions expressed here are the author's and do not necessarily represent those of Oracle Corporation.*

Check your member ID at nocoug.wildapricot.org. If it is 6431187, you have won a copy of Oracle Database 12*c* Security. Collect your prize at the fall conference on November 20 at PayPal Town Hall.