Global Information Assurance Certification Paper

Margaret Reedy
Security Essentials, Version 1.4b

ORACLE'S VIRTUAL PRIVATE DATABASE

ABSTRACT

Databases are often well behind the organization's security perimeter, and while flaws in the database software provide avenues of attack to outsiders, the more likely source of threats against the confidentiality and integrity of data stored in the database are from insiders. Some can be unintentional, as when end-users access the database using software that the application developers did not anticipate and account for. This is the 'application security problem'. If weaknesses exist, there is always the possibility that knowledgeable users will exploit them for malicious purposes. Furthermore, recent incidents like the Breeders' Cup scandal illustrate that not all super users are trustworthy.

Oracle's fine-grain access features help protect data from accidental or intentional misuse by legitimate users inside the security perimeter. This paper takes a look at the components that make up Oracle's 'Virtual Private Database' and how they give developers and database administrators tools to protect the data that is often of crucial importance to the organization. Finally, it looks at some of the new features available in Oracle 9i and as additional cost options.

THE "APPLICATION SECURITY PROBLEM"

"When access control is embedded in an application (instead of being enforced directly on the data), users who have access to ad-hoc query or reporting tools can bypass the security mechanisms."[1]

Why would access control be embedded in an application? In Oracle, privileges are granted at the table level. Someone who has the 'update' privilege against a given table can update any or all rows in that table. In practice, most applications need to put further restrictions on the update privilege. For example, in a Human Resources application, it might be desirable for an employee to be able change his / her home address, but we probably would not want one employee to be able to change another's home address, unless that employee worked in Human Resources. Many applications build these business rules into the user-interface programs.

"Oracle 9i builds upon over two decades of development and deployment ".[2] When Oracle's relational database was introduced, the paradigm was to separate data from business logic on the assumption that data structures were less likely to change than the more volatile business rules that governed operations on that data. The client-server model pushed business logic to the client to reduce network traffic and improve response time by doing data validation locally. The application server model took this a step further by having users connect to the application rather than the database, and

having the application connect to the database as a single user with all necessary privileges.

The application security problem occurs when a user accesses the database with software other than the application's user-interface programs. This might be an ad-hoc reporting tool or a software package with ODBC or ADO connection to the database. Because the privileges are granted within the database, the user who connects to the database with software other than the application interface bypasses the restrictions built into the application interface. In the above example, the user would have sufficient privileges to update all home addresses, not just his / her own address.

When business rules are not enforced, the integrity and confidentiality of the database is threatened. A user who knows enough to be dangerous might offload an entire table through an ODBC connection, modify a few records, and then replace the entire table, overlaying changes made in the interim by other users. Older applications may use code built into the interface to perform more complicated validations, along with restricting access. For example, the business rule may require that a unit supervisor enter both the closing date and reason to close a matter. Via a third party connection, any person with update privilege could enter the closing date for any matter in the table, thereby introducing a data anomaly (a closed matter without a reason) while circumventing the organization's division of responsibilities (supervisors may only close matters assigned to their own units). While database triggers can help address the validation issue by moving the validation from the application to the database level, database triggers don't check if the user is a supervisor in the unit the matter belongs to.

The 'select' privilege does not carry the same possibility of damaging data, but confidentiality is at risk. Business rules might dictate that a person should only be able to see the accounts assigned to him / her, but a reporting tool won't be aware of that.

In these circumstances, even well-intentioned users can violate confidentiality and cause inaccurate data to be committed to the database. Motive may help in identifying sources of possible attacks, but lack of motive doesn't guarantee security.

Before Oracle introduced the concept of Virtual Private Database, developers came up with several ways (mostly relying on security through obscurity) to protect against the application security problem. "Hack Proofing Oracle" [3] describes these methods and explains why each falls short.

THE VIRTUAL PRIVATE DATABASE

At first glance, the terms 'row level security', 'fine grain access control' and 'virtual private database' seem to be interchangeable.

Row level security is the goal of limiting access to the individual rows in a table to appropriate users. Fine grain access control is a method of achieving row level

security. Oracle's Virtual Private Database is a feature that combines fine grained access control with secure application context to provide a row level security that presents to each database user only the data that the user should see, based on known characteristics of the user.

Davison describes the Virtual Private Database as "users - from different companies, divisions, or groups, and with different roles and a variety of access privileges - have secure access to their own slice of data, as if it were physically partitioned".[4]

In a nutshell, the VPD is a combination of packages that work together to modify the 'where' clause of an SQL statement as it is executed based on characteristics of the user executing the statement. The modified 'where' clause usually returns fewer rows of the table than the original 'where' clause would. In other words, the FGAC packages append conditions to the 'where' clause to limit the rows returned.

The main components of the VPD are the **application context** and the **security policy**. Before delving into details, we'll review some terms and sketch how these components work with each other and other Oracle features to control access to tables.
The VPD is used in conjunction with roles, which are groupings of privileges. Roles are still necessary, because they grant the user privileges to the tables that are named in the 'from' clause of the SQL statement. The role determines which tables the user has access to. The security policy determines which rows within a table the user has access to.

Both the application context and the security policy are associated with PL/SQL packages. Packages are collections of procedures and / or functions. Functions are executable statements which accept input parameters, process them and return values to the procedure which called the function. Packages are stored in the database. So, packages are collections of executable code stored in the database.

The application context captures pertinent characteristics of the user. The security policy contains instructions for writing 'where' clauses for specific tables. The security policy refers to the application context to build the 'where' clauses based on the characteristics of the user.

APPLICATION CONTEXT

The application context is the component that specifies the characteristics of the user that are used in determining what rows the user should be allowed to access. There are two database object types involved: the 'context' and a 'package' which implements the context. Let's deconstruct some more formal definitions.

"Application contexts are secure namespaces that identify the current values for the application-specific attributes you designate."[5]

"A namespace is an area in which no two objects can have the same name."[6]

What this means is that "context names are unique across an entire database, to ensure that contexts can't be duplicated or spoofed by individual users, either inadvertently or maliciously".[7] Database object names, such as table names, are usually qualified by the name of the user that creates the object, called the 'owner' or 'schema'. When a user tries to reference an object name that is not qualified with an owner name, Oracle assumes that the user owns the object. If contexts followed this convention, a user with the 'create any context' system privilege would be able to substitute a private context to elevate his / her privileges.

Instead, all application contexts are owned by the user 'SYS', regardless of which user executes the 'create any context' statement.

The 'create any context' statement simply associates the context name with a package. It is the package that specifies the which attributes will be used to determine whether the user should have access to a given row, and populates the variables for the user's session.

The package is a PL/SQL program which defines and initializes variables which describe the user. For the earlier example of the business rule stating that the a unit supervisor may close matters for his / her own unit, the variables needed would include the user's job title and unit. This information would be selected from the application's tables. The package would include a 'dbms_session.set_context' statement that essentially commits the variable values to memory.

Information could also be selected from the data that the system collects about the user session, including the name of the user logged in and the IP address. USERENV is "a special context namespace, the user environment, which is automatically created by Oracle"[8]. (Just to confuse matters, USERENV is also a function.) When USERENV is used as a context namespace, the function SYS_CONTEXT is used to retrieve information from it.

"An application context functions as a cache for repeatedly used application-oriented information." [9] That is, once the context is initialized for the user session, the variables describing the user remain in memory for reference by the security policy throughout the session, or until the context is explicitly refreshed.

Recapping, the package which implements the application context gathers information about the user from either application tables or from the system's session data and holds that information in memory.

Most articles on this topic recommend that the application context be set when the user connects to the database by means of a database trigger. If the context is set after connection by the application code, it may be possible to bypass the security policy restrictions, depending on how the policy is written.

SECURITY POLICY

The security policy is the component that builds the dynamic 'where' clauses when a SQL or PL/SQL statement is executed against a table. There are two parts involved: the 'policy' and a 'package' which implements the policy. The security policy package refers to the application context when building the 'where' clauses. Again, Let's start with some more formal definitions.

The policy is a named association between a schema and object and the function that builds the 'where' clause for that object. The policy also specifies which types of statement (SELECT, INSERT, UPDATE and DELETE) are governed by the policy.[10] If a policy exists for a given table, the database knows that it has to use the function to modify the 'where' clause of any statement executed against the table. The policy is technically not a database object, but policies are tracked in the DBA_POLICIES table in the data dictionary.

The security policy package is a PL/SQL program that contains the procedures and functions that build the modified 'where' clauses. This is called "dynamic query modification".[11] "... (A) user directly or indirectly accessing a table ... with an associated security policy causes the server to dynamically modify the statement based on a "WHERE" condition (known as a *predicate*) returned by a function which implements the security policy".[12]

The security policy package often begins with a procedure that initializes variables used in determining what the user should have access to. This procedure may use the 'SYS_CONTEXT' function to obtain values from the user's application context.

But the heart of the security policy package are the functions that build the predicate. Suppose we want to restrict employees to viewing their own personal data for verification purposes, but need to allow employees in Human Resources to view all personal data. The generic SQL statement would read:

```
select home_address
from emp_personal_data_table
where [predicate ]
```

For most employees, the predicate might read 'emp_userid = userid'. This would allow an employee to view his / her own home address.

For an employee working in the Human Resources bureau, the predicate might read '1 = 1'. This would allow the HR employee to see all addresses because the condition is always true.

Obviously, real world functions would contain much more complex 'if' statements, depending on the complexity of the business rules. Good programming practices dictate that the function should in some way account for users who do not meet any of

the conditions in the 'if' statement. Assumptions have always been dangerous in computer code, and the security policies are no exception. The more explicit the conditions in the policy, the more legible and robust the policy will be.

STEP-BY-STEP ARTICLES

The following articles provide step-by-step guides along with sample code to illustrate the process of setting up a VPD. All are helpful.

- "Virtual Private Databases (VPD)"[13] provides a very simple, succinct illustration of how the various pieces work together, including steps for testing the policy. This article recommends granting the execute privilege on the security package to PUBLIC.

  This may conflict with the advice from "Hack Proofing Oracle" : "Revoke as many packages from PUBLIC as possible. Revoke as many privileges and roles from PUBLIC as possible. Ensure end users cannot execute packages that might compromise underlying system security".[14] An alternative is to grant the privilege to execute the security policy package to one of the application's roles, rather than to PUBLIC. The users who are given access to the application's tables and views are also given permission to execute the security policy package; others are not.

- Mary Ann Davison's "Creating Virtual Private Databases with Oracle 8i"[15] describes a four-step process, using an order entry system as the example. This article includes background on why you would want to use VPD and suggestions for enhancing security, and its blend of introductory theory and practical example is an excellent starting point.

- Roby Sherman' "Internet Security With Oracle Row-Level Security"[16] walks through a manager / employee example in four steps with lots of explanatory notes in the code samples.

- Michael R. Ault's "Managing Row Level Security in Oracle 8i"[17] describes a five-step process for a graphics application. The example itself is not as intuitive as others, but the article includes a summary of usage guidelines and an explanation of the syntax of the DBMS_RLS package that is used to create and maintain policies.

  •Steven Feuerstein in Guide to Oracle8i Features[18] includes a chapter called "Deploying Fine Grained Access Control". The example, a health care system used by doctors, patients and regulators, follows a explanation of the statements, procedures, functions and packages used to build the security policy. This is more detailed than the articles mentioned above, and includes debugging tips.

- Douglas Scherer et al in Oracle8i Tips & Techniques[19] also devote a chapter

called “Security” to describing the virtual private database features. The example is a human resources application. This also contains lots of detail and suggestions for ease of management.

TIPS, TRICKS, CONSIDERATIONS AND CAVEATS

The step-by-step articles all try to demonstrate the process in simple terms. Most allude to possible complications in the world beyond the textbook examples. The following is a compilation of suggestions, some explicit and some suggested between the lines.

- “...the ability to create a security context is a separate system privilege; only suitably-privileged users are able to create a context.”[20] As always, good security practice requires that some thought be given to which groups of users (developers, DBAs, schema owners) should be given the privileges needed to create a security policy.

- “...it's a good idea to have the policy function owned by a system security officer, to prevent a developer or user from inadvertently or maliciously dropping a policy from a table.”[21] That is, the security policy should be created by a schema / user other than the schema which owns the tables and views protected by the policy. There should be restricted access.

- Security policies need to take into account parent-child relationships between tables. Applications may force users to retrieve information from the parent or master table first before selecting from the child or detail tables.[22] Again, the policy needs to be written on the assumption that the user is bypassing the restrictions built into the application.

- “If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.”[23] The principle of least privilege would suggest that security functions should be written to provide explicit access to users meeting certain conditions, and to provide no access to the users who fail to meet any of those conditions. Accordingly, a ‘none of the above' option should build a predicate that will deliberately fail to retrieve any rows from the table. The code examples in the step-by-step articles show several ways of accomplishing this.

- “SYS user is not restricted by any security policy.”[24] The implication here is that other users, including SYSTEM, are. This may affect exports performed by SYSTEM. The schema which owns the tables and views may also be restricted by the security policy. This may require some adjustment in the assumptions made by developers who are used to the schema owner automatically having full rights to every row of every table. There is an EXEMPT ACCESS POLICY system privilege, but as with any system privilege, it should be used judiciously.

- Security policies need to account for super users’ activities. Many applications

require batch processing. Some user, frequently the application owner, must be authorized by the security policy to perform the necessary updates on all rows of the table.

- It is possible to construct a security policy function that causes an infinite loop.[25] Suppose information from the employee table is used in determining which rows the user can access, and that the security policy function which protects the employee table needs to query the employee table to find that information. The suggested remedy is grant access to the table only to the security manager user which creates the policy, and to have all other users access a view of the table. The security policy functions for the view would select the attributes which determine access from the original table.

- Errors in the application context invoked by a database trigger on logon could prevent all users from connecting to the database.[26] As security policies do not apply to the user SYS, SYS is able to come to the rescue.

- It is possible to create a context using a schema and package that does not exist.[27] This can cause confusion and name conflicts, particularly in the development environment, unless there is a periodic procedure to find and remove the contexts that refer to phantom schemas and packages.

ORACLE 9I ENHANCEMENTS

The basic VPD available in Oracle 8i applies a policy for a given user, regardless of which application the user is using to access the database. Oracle9i offers some enhancements, particularly for the n-tier application model.

Partitioned FGAC allows for different policies to be applied to the same tables and views for the same user, depending on which application the user is using. This allows applications to use 3rd party software tools, such as reporting tools. It simplifies development because the developers of different applications using the same data don't have to coordinate their efforts to define a single policy that addresses all possibilities. The server detects which software is accessing the data during a given session, and activates the appropriate policy. Better yet, when in doubt, the server concatenates all policies so that when no specific policy is in force, all policies are in force.[28]

The **global application context** allows an application server program to identify an end-user who is connecting to the application as if that end-user were connecting directly to the database. This is especially good for auditing activity at the database level. Otherwise, the application has to duplicate functionality that is available in the database, which increases cost of development and maintenance.[29]

Oracle9i Label Security is an additional cost option that is useful when the characteristics of the data to be protected are not sufficient to describe the data's sensitivity. "Oracle Label Security uses the application context functionality of the VPD

product"[30] , but where FGAC uses a dynamic predicate, OLS uses labels applied to both the data and to the users to control access.[31]

"Oracle Label Security is based on data element labeling concepts used by government and defense organizations to protect sensitive information and provide data separation."[32] One of the elements of the label is the sensitivity of the data. The other elements of the labeling scheme are 'compartment' and 'group'. These are likely to be the attributes similar to ones a FGAC security policy would use. When individual rows in a given compartment and group have different sensitivity levels, these compartment and group characteristics are not enough to identify which users should have access to the rows.

Spy movies have made us all familiar with ratings like 'classified' and 'top secret', and we are also comfortable with the idea that a person with 'top secret' clearance would be able to access information at all sensitivity ratings up to and including 'top secret'. The sensitivity hierarchy, in combination with policies defined by the application context, provides an extra layer of protection.

OLS includes an EXEMPT ACCESS POLICY system privilege, which provides a way to exclude a user from the security restrictions.[33]

Oracle-Base provides a simple step by step guide to setting up an Oracle Label Security policy.[34]

FINE GRAIN AUDITING

One of the insider problems mentioned at the start of this paper was the super user who uses his / her privileges inappropriately. The principle of least privilege and separation of duties help prevent problems in this area, but there also needs to be a way to detect whether there have been violations of policy. Auditing is a tool that can help, but it often generates so much data that it is more useful for proving a suspected problem than flagging one for investigation. Loney recommends "enabling auditing for certain key events, reporting on those events frequently, and truncating the SYS.AUD$ table regularly" as one of his six tips for protecting an Oracle database.[35]

Fine grained auditing is a tool that can help identify key events, particularly when there are concerns about the privacy of the data. "Oracle FGA provides facility to identify abuse of legitimate user privileges and possible intruders while reducing the volume of unnecessarily audit data that would normally be generated by turning on full-blown auditing." [36]

The concept is similar to fine grain access control. The fine grain auditing policy defines the schema and object to be protected, a condition that will trigger an audit entry, and the information to be captured for the audit entry. FGA identifies conditions that would classify the row as sensitive, and then reports specific actions against the sensitive rows. For example, a law enforcement agency might want to be alerted whenever

someone attempts to look up the license plate of one of its undercover vehicles.

SUMMARY

"The compromise between enabling appropriate database access and maintaining tight security against unauthorized access is the enduring dilemma of the database administrator". [37]

Oracle's Virtual Private Database features and related products give developers and database administrators powerful tools to protect the confidentiality and integrity of the organization's most valuable data.

REFERENCES

Ault, Michael R., TUSC. "Managing Row Level Security in Oracle 8i". June 2001.
URL: http://www.quest-pipelines.com/newsletter-v2/rls.htm

Browder, Kristy and Mary Ann Davison. "The Virtual Private Database in Oracle9i R2 - Understanding Oracle 9i Security for Service Providers - An Oracle White Paper". January 2002.
URL: http://otn.oracle.com/deploy/security/oracle9iR2/pdf/VPD9ir2twp.pdf

Davison, Mary Ann. "Creating Virtual Private Databases with Oracle8i". Oracle Magazine. July 1999
URL: http://www.oracle.com/oramag/oracle/99-Jul/49sec.html

Davison, Mary Ann. "The Need for Granular Access Control". Secure Business Quarterly. Volume Two, Issue Two, Second Quarter 2002.
URL: http://www.sbq.com/sbq/app_security/sbq_app_granular_access.pdf

Feuerstein, Steven. Guide to Oracle8i Features. O'Reilly & Associates, Inc, 1999. p.153-170

Loney, Kevin. "Protecting Your Database", Oracle Magazine, May 2000.
http://www.oracle.com/oramag/oracle/00-May/index.html?o30sec.html

Oracle-Base, "Oracle Label Security (OLS)",
URL: http://www.oracle-base.com/Articles/9i/OracleLabelSecurity9i.asp

Oracle-Base, "Virtual Private Databases (VPD)", URL: http://www.oracle-base.com/Articles/8i/VirtualPrivateDatabases(VPD).asp#SecurityPolicies

Scherer, Douglas, William Gaynor, Jr., Arlene Valentinsen, Xerxes Cursetjee. Oracle8i Tips & Techniques. Osborne / McGraw-Hill, 2000. p.193-217

Sherman, Roby “Internet Security With Oracle Row-Level Security”
URL: http://www.interealm.com/roby/technotes/8i-rls.html

Sherman, Roby “Implementing Data-Level Monitoring With Oracle Fine-Grained Auditing” .
URL: http://www.interealm.com/technotes/roby/fga.html

Smith, Howard. “Hack Proofing Oracle”.
URL: http://otn.oracle.com/deploy/security/pdf/oow00/orahack.pdf
(Note: Oracle Technology Network (otn) requires a user account and password)

Ziola, Brad. “Label Based Access Control vs. Fine-Grained Access Control for Implementing a Virtual Private Database”. March 2002.
URL: http://www.managedventures.com/images/OLSFGAC.pdf

ENDNOTES

1. Davison, “The Need for Granular Access Control”, p. 2
2. Browder, p. 1
3. Smith, p. 6-7
4. Davison, “Creating Virtual Private Databases with Oracle8i”, p. 2
5. Davison, “Creating Virtual Private Databases with Oracle 8i” , p. 2
6. Scherer, p. 197
7. Browder, p. 6
8. Scherer, p. 197
9. Scherer, p. 210
10. Davison, “Creating Virtual Private Databases with Oracle 8i” (Part 2)
11. Browder, p. 5
12. Browder, p. 5
13. Oracle-Base, “Virtual Private Databases (VPD)”
14. Smith, p. 5
15. Davison, Mary Ann “Creating Virtual Private Databases with Oracle 8i”
16. Sherman, Roby “Internet Security With Oracle Row-Level Security”

17. Ault

18. Feuerstein, p. 153-170

19. Scherer, p. 193-211

20. Browder, p. 7

21. Davison, “Creating Virtual Private Databases with Oracle 8i”

22. Davison, “Creating Virtual Private Databases with Oracle 8i”

23. Ault, p. 6

24. Ault, p. 6

25. Scherer, p. 203

26. Feuerstein, p. 169

27. Scherer, p. 212

28. Browder, p. 8

29. Browder, p. 9

30. Ziola, p. 1

31. Ziola, p. 3

32. Ziola, p. 1

33. Ziola, p. 2

34. Oracle-Base, “Oracle Label Security (OLS)”

35. Loney

36. Sherman, “Implementing Data-Level Monitoring With Oracle Fine-Grained Auditing”

37. Ziola, p. 1