CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

HOWTO
Secure and Audit Oracle 10g and 11g

Ron Ben Natan

Foreword by Pete Finnigan

Chapter 2

Hardening the Database

System hardening is the process by which you securely configure a system to protect it from unauthorized access. System hardening is necessary in any system that has a range of configuration options and is viable in any system that has enough security measures to make them suitable for usage in security-oriented environments. Oracle database falls into both of these categories.

The purpose of system hardening is to eliminate as many security risks as possible. This is done by removing all nonessential elements from the system and by selecting configuration options that limit access and reduce risk. As Oracle has evolved, more and more options have become available and these options offer new ways to access data—sometimes by unauthorized users if used inappropriately. The larger the footprint and capabilities of a system, the harder it is to harden and the more security risks may be present. Therefore, as the Oracle database grows in size and functionality, hardening becomes even more important. Luckily, as Oracle evolves, there are also more and more security options available that you can use to secure the data—Chapters 3 through the end of the book outline these capabilities and how you should use them. But first, you need to harden the database.

Oracle hardening covers a wide range of activities and involves many types of configuration options. The most important guideline is that if there is a feature that you do not use, remove it. The fact that you don't use something does not mean that an attacker won't. The smaller the surface area of a system, the more secure it is. Examples of this include:

- Remove or lock predefined accounts that you do not use and change the password for accounts you do use that have a predefined password (Oracle 11g already comes configured that way).
- Remove predefined roles that you do not use.
- Remove components in the database software that you do not use.
- Remove options that you do not use—for example, remove EXTPROC from your listener if you do not use external procedures.
- Remove privileges from PUBLIC that you do not require.

Because Oracle has so many capabilities and configuration options, hardening is usually an exercise that involves hundreds of activities. Coming up with a list of these required activities is a monumental task. Luckily, you don't have to come up with this list. Lists have been created and

entire books are dedicated to this topic—so all you have to do is pick a source—the topic of the first HOWTO of this chapter.

2.1 HOWTO Choose a Hardening Guideline

Hardening of any complex system involves many little details. The more a system can be configured, the lengthier the list of tasks you need to perform (or validate) to create a hardened configuration. The list for Oracle is long, but openly available and free.

There are two documents that provide very mature guidelines for implementing a secure Oracle configuration and that you should look at when forming your standard hardening process. One is the Database Security Technical Implementation Guide (STIG) developed by Defense Information Systems Agency (DISA) for the Department of Defense (DOD) and the second is the Center for Internet Security (CIS) Benchmark for Oracle developed by the CIS. Both are excellent and very comprehensive; use one of them (or both) rather than develop your own hardening checklist.

Database STIG

STIGs are documents published by the DISA to assist in improvement of the security of DOD information systems. There are numerous STIG documents—all of them are accessible at http://iase.disa.mil/stigs/stig/index.html. The checklists can be downloaded from http://iase.disa.mil/stigs/checklist/index.html. The Database STIG focuses on relational databases. The Database STIG has a generic section which outlines guidelines relevant to any database management system (DBMS) and has an Oracle-specific section which adds steps relevant for Oracle only. The sections within the general document address:

1. Integrity
   a. Software integrity
   b. Database software development
   c. Ad-hoc queries
   d. Multiple services host systems
   e. Data integrity—including file integrity, software baseline, and file backup and recovery
2. Discretionary access control
   a. Account control
   b. Authentication
   c. Database accounts
   d. Authorizations
   e. Protection of sensitive data
   f. Protection of stored applications
   g. Protection of database files
3. Database auditing
   a. Audit data requirements
   b. Audit data backups
   c. Audit data reviews
   d. Audit data access
   e. Database monitoring

4. Network access
   a. Protection of database identification parameters
   b. Network connections to the database
   c. Database replication
   d. Database links
5. Operating system (OS)
   a. File access
   b. Local database accounts
   c. Administrator accounts
   d. OS groups

The Oracle-Specific Policy and Implementation appendix specifically addresses:

6. Oracle access control
   a. Oracle identification and authentication
   b. Oracle connection pooling
   c. Secure distributed computing
   d. Oracle administrative connections
   e. Oracle administrative OS groups
   f. Default accounts
   g. Default passwords
   h. Oracle password management requirements
7. Oracle authorizations
   a. Predefined roles
   b. System privileges
   c. Object privileges
   d. Administration of privileges
8. Oracle replication
9. Network security
   a. Encrypting network logins
   b. Protecting network communications
   c. Listener security
   d. XML DB protocol server
10. Oracle Intelligent Agent/Oracle Enterprise Manager (OEM)
11. Oracle account protections
12. ARCHIVELOG
13. Securing SQL*Plus
14. Protecting stored procedures
15. Oracle trace utility
16. Auditing in Oracle—includes standard auditing, fine-grained auditing, mandatory auditing, and architectural discussions
17. File and directory permissions at the OS level
18. Critical file management—including control files, redo log files, and data files
19. Optimal Flexible Architecture (OFA)
20. Initialization parameters
21. Miscellaneous OS requirements—including Unix, Window, and z/OS

The Database STIG is published as an unclassified document and is made available to all. DISA also publishes a set of evaluation scripts and these can help you check the security strength of your database—download these from http://iase.disa.mil/stigs/SRR/index.html.

CIS Oracle Benchmark

The CIS (www.cisecurity.org) publishes the CIS Benchmark for Oracle as part of a set of benchmarks, scoring tools, software, data, and other services that are made public as a service to all users worldwide. You can download the benchmark from http://www.cisecurity.org/bench_oracle.html. The recommendations contained in the Oracle benchmark result from a consensus-building process that involves the leading Oracle security experts. The CIS benchmark takes the form of a checklist partitioned into a number of sections. Within each section is a list of items that should be validated. Each such item includes a description of the item, the action or recommended setting for parameters, comments, which Oracle version it applies to, and whether it is relevant to Unix, Windows, or both. The main sections in the CIS Oracle benchmark are

1. OS-specific settings
2. Installation and patch
3. Oracle directory and file permissions
4. Oracle parameter settings
5. Encryption-specific settings
6. Startup and shutdown
7. Backup and disaster recovery
8. Oracle user profile setup settings
9. Oracle user profile access settings
10. Enterprise Manager/Grid Control/Agents
11. Items relevant to specific subsystems
12. General policy and procedures
13. Auditing policy and procedures
14. Appendix A—additional settings

Both documents take a broad approach to hardening. They do not have a narrow interpretation that hardening only involves certain configuration settings, removing default components, locking users, etc. They provide a full checklist that also includes what activities should be audited, where separation of duties is required, what activities need to be performed, etc. Of the two—the STIG puts even more focus on the general implementation, process, roles that need to be involved in securing an Oracle environment, etc.

Two Things to Remember about Choosing a Hardening Guideline

1. Don't build your own checklists—hardening an Oracle database is no longer an art; there are good mature guidelines for you to choose from such as the CIS Oracle benchmark or the Database STIG.
2. Use these documents not only as a hardening guide but also as the basis for putting together a complete Oracle security implementation. These documents outline configuration settings but also outline process, procedures, and what to focus on. In many ways, all the chapters in this book explain how to use Oracle tools to implement what these two documents suggest that you do.

2.2 HOWTO Use a Vulnerability Assessment Tool

Using hardening checklists is simple but tedious. From a pure hardening perspective, the checklists contain many checks and modifications that you need to perform and these can be automated. In fact, without automation this tasks quickly becomes unmanageable—especially if you have tens and hundreds of instances and they do not all conform to a single "gold build." Tools that you can use to automate this process are called vulnerability assessment (VA) tools or vulnerability scanners.

For almost any type of system there are VA tools and Oracle is no exception. VA tools will scan your database instances and come back with a report showing what changes you need to perform to make your database more secure. These results are presented in the form of a security report where each problem is classified and a recommendation provided for what changes you need to make (e.g., see Figure 2.1). These checks and recommendations usually cover the items specified in the various hardening checklists—meaning that a VA tool can save you most of the tedious work involved in reviewing your databases and their alignment with the checklists.

There are many VA tools for Oracle including AppDetective, AppSentry, Guardium, IPLocks, and NGS Squirrel. Some of these tools are stand-alone VA scanners and some tools are part of a larger suite of products that address multiple aspects of Oracle security. From a pure VA perspective all these tools scan your database to recommend changes you need to make to harden your instances. The tools that are integrated within a larger suite have an advantage—in the same way that STIG and CIS take the wider interpretation to securing the database environment and define practices for auditing, practices for review, etc. (see the previous section), so do these suite-based tools. This allows you to become secure and fully compliant within a single implementation thus saving a lot of time.

VA tools perform many types of checks. These checks can be classified into three main groups:

1. Checks for software vulnerabilities
2. Checks for misconfigurations
3. Checks for misuse of the database

All of these checks are necessary to check for vulnerabilities in your database. An attacker can gain unauthorized access to your database by using a code vulnerability that exists because you have not patched your server using the latest Critical Patch Update (CPU) (an example of type 1), through the use of a default account that has not been locked and still has a default password (an example of type 2), because you have REMOTE_OS_AUTHENT set to true (an example of type 2), or because everyone knows the password for SYSTEM and multiple people make use of this account constantly (an example of type 3).

Checking for vulnerabilities (of all types) is done using a multipronged approach. Some things can be checked from the outside-in and other checks are done from within the database. VA tools have multiple modes in which they work to provide you with the full picture. Many checks need to be performed from the inside. For example, to check for bad configurations the VA tool needs to be able to access V$PARAMETER. To check for bad privilege assignment (as an example—too many privileges assigned to PUBLIC can be a serious vulnerability), VA tools need certain SELECT privileges to the catalog. These tools come with scripts that grant these privileges to a user or a role (that is then assigned to a user you create for the tool to use). Review these scripts when you evaluate the tool to ensure that it does not assign itself too many privileges. The better VA tools also inspect and check files at the OS level. For example, lax file permissions or a wrong owner or group used for important Oracle files (such as data files, software files, configuration files, and log files) are a serious vulnerability. Contents of files such as sqlnet.ora can affect how your database behaves and checking a configuration must include checks on files, on registry values (on Windows), environment variables, etc. Finally, comprehensive checks will often include inspecting output from scripts and OS commands—e.g., inspecting the

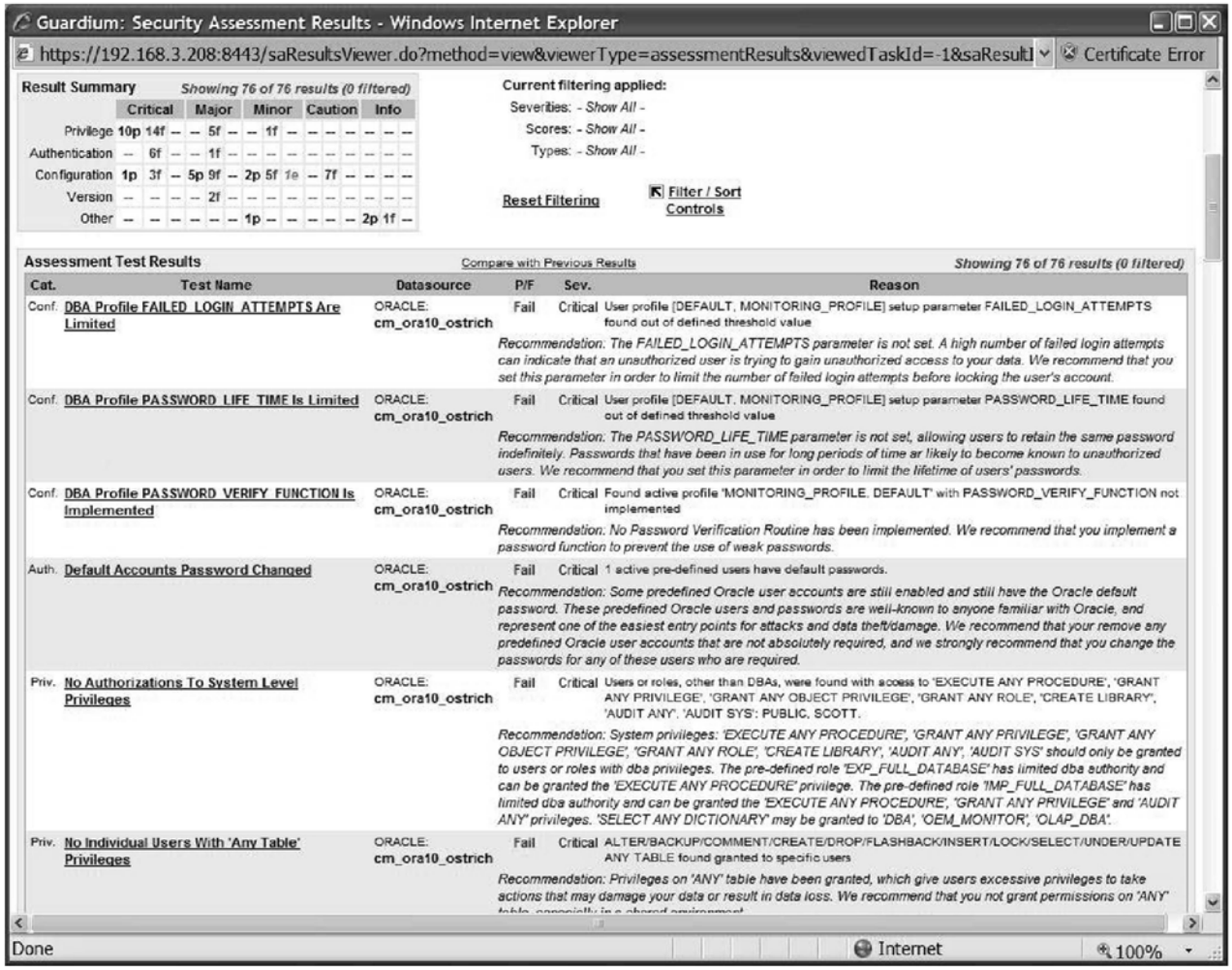

Figure 2.1 Sample recommendation report from an Oracle VA scanner.

output of orapatch to ensure that you have the latest CPUs installed to address known code vulnerabilities. At the end of the day, a VA tool is the only way to ensure that you have hardened your databases properly and that you remain compliant over time.

Three Things to Remember about Using an Assessment Tool

1. Some VA tools are stand-alone scanners and others are part of a larger database security suite. If you're implementing the full set of recommendations presented by within the Database STIG or within the CIS checklist you should consider the suite products that can also support your auditing implementation, intrusion detection implementation, separation of duties, etc.
2. VA scanners check both vulnerabilities and CPUs installed (or that should be installed) as well as the configurations of your database.
3. Some of the checks that you need to perform are at the OS level. Make sure the VA tool you choose can perform checks on file ownership, file permissions, etc.

2.3 HOWTO Create and Maintain a Secure Configuration Baseline

Once you have finished hardening your database, you have a tight configuration, but you need to ensure that it remains tight and does not degrade over time. There are two things you can do to ensure a sustained secure configuration—(1) run assessments on a scheduled basis to find new vulnerabilities as they are created, and (2) create a baseline for the configuration once you are happy with it and track any changes from this configuration using an alert that needs to be reviewed and approved. The best practice suggests that you do both of these because they complement each other.

If you have a VA tool, then running an assessment periodically is easy. All VA tools have a scheduler that allows you to test your databases every month or every week. Most VA tools also have a "diff report" that shows you changes to the assessment results between one run and another—so once you are happy with the results of a scan you can monitor only the differences in future scans.

The second set of tools is called change tracking tools. These tools create a baseline of your configurations and alert you on any change that is made. Baselines are created by computing a digest on configuration elements and then periodically recomputing them to see if there are any changes. Digests (or hash values) can be computed on files (to ensure for example that no one modifies the software files themselves), on the output of a script (e.g., the output of a script that greps certain values from sqlnet.ora), on Structured Query Language (SQL) result sets (e.g., the output of a query that checks assignment of system privileges), etc. A change tracking tool tells you when there is a deviation from the hardened configuration.

Change tracking tools will always be part of a database security implementation. These tools are required to comply with the broad hardening checklists because this is the only way you can ensure that no one replaces your files with Trojan versions. They are also the most effective way to ensure that scripts run periodically are not used as a point-of-compromise; tracking changes made to scripts is much simpler and more effective than reviewing audit trails that show what a script did.

Because you'll have access to these tools if you're responsible for database security, use them in the context of VA change tracking tools—once you've completed your hardening process use them to

ensure that you don't deviate from this standard. If there are changes over time (and there always will be changes) you can reauthorize and update the baseline and track changes from that point on. Look for a VA tool that includes a change tracking tool to ensure sustained and continuous compliance.

Three Things to Remember about Creating and Maintaining a Secure Configuration Baseline

1. Change tracking tools have multiple uses within an Oracle security implementation. One of these is to create and track a secure baseline following the hardening phase. VA tools that incorporate a change tracking tool offer you more options in terms of continued compliance.
2. Baselines are generated by creating digests that uniquely identify a file or a result of a script. Any change to the underlying entity will cause the digest to change and a change report to be issued. Digests act as digital fingerprints of the underlying entity.
3. Baselines include digests for files that should not change, digests for the result of an OS script, digests for the result of a query, digests for values of environment variables or registry entries, and more.

2.4 HOWTO Understand Critical Patch Updates

Always install patches to security issues as soon as they are available from Oracle. When code vulnerabilities are discovered (and there always will be code vulnerabilities—any large piece of software has bugs), Oracle issues patches—you must install these patches to remove these vulnerabilities from your environment.

Security patches come in the form of Oracle CPUs. An Oracle CPU is a bundle of patches that are released on a quarterly basis to fix security issues. CPUs have been around since 2005 (before that there were called Security Alerts) and they come out at 1 p.m. Pacific time on the Tuesday closest to the 15th of January, April, July, and October. The fact the CPUs are released at a known date is important—it allows you to plan ahead and define change management windows accordingly. Before CPUs were used, security alerts were issued when issues were discovered and fixed. This made installing these security patches very difficult and sometimes as people were in the processing of testing one fix, another one would be announced. Knowing when the patches are published makes it easier for you to build a process around applying them.

The Oracle CPU includes fixes for all Oracle software components. One patch is released per version of the database, application server, Enterprise Manager, etc. This has changed as of 2007 with the introduction of the n-apply process (more on that later). Patches are also released for Oracle E-Business Suite, PeopleSoft, Siebel, and other applications. Patches for the database are cumulative so that the latest CPU includes fixes for all earlier CPUs unless stated otherwise.

Each CPU includes a set of patches, an advisory, preinstallation notes, and release notes. The CPU advisory contains information which helps you evaluate the impact of the fixed vulnerabilities so that you may assess the criticality of issues and how quickly you need to install patches on production systems. The advisory includes a set of risk matrices—one per software system—in which Oracle reports on the risks of the discovered issues.

Reading a CPU Advisory Risk Matrix

A CPU advisory contains a database risk matrix. The risk matrix helps you understand the risk of discovered issues in terms of loss of confidentiality, loss of integrity, and loss of availability—the three dimensions that can be affected by security vulnerabilities.

Figure 2.2 shows a sample from the database risk matrix of the April 2008 CPU. The matrix summarizes the list of vulnerabilities fixed within the CPU and for each one provides information about risk. Each vulnerability is given a vulnerability number composed of four characters—the first two characters represent the system and the last two are an incremental numeric code starting from 01. Database vulnerabilities are tagged as DB##. Each CPU has its own numbering scheme so the vulnerability number is unique within a CPU. Sometimes a vulnerability in another system affects users of the Oracle database—in this case that vulnerability will be included in the database risk matrix so that the risk matrix is self-contained and you do not need to read the entire CPU if you only care about the database. As an example, in Figure 2.2 EM01 is listed within the database risk matrix even though the vulnerability is in Enterprise Manager. In such a case the vulnerability number appears in italics.

The risk matrix contains information about which component the vulnerability is in, what protocol is required to exploit the vulnerability, what package or privilege is required to exploit the vulnerability and whether an attacker can exploit the vulnerability from a remote node without

Oracle Database Risk Matrix

Vuln#	Component	Protocol	Package and/or Privilege Required	Remote Exploit without Auth.?	CVSS VERSION 2.0 RISK (see Risk Matrix Definitions)							Last Affected Patch set (per Supported Release)	Notes
					Base Score	Access Vector	Access Complexity	Authentication	Confidentiality	Integrity	Availability		
EM01	Oracle Enterprise Manager	Local	None	No	6.6	Local	Medium	Single	Complete	Complete	Complete	9.0.1.5+	
DB01	Advanced Queuing	Oracle Net	Execute on SYS.DBMS_AQ	No	5.5	Network	Low	Single	Partial	Partial	None	9.0.1.5+, **9.2.0.8**, 9.2.0.8DV, 10.1.0.5, **10**.2.0.3	See Note 1
DB02	Change Data Capture	Oracle Net	Execute on DBMS_CDC_UTILITY	No	5.5	Network	Low	Single	Partial+	Partial+	None	10.1.0.5, 10.2.0.3, **11**.1.0.6	
DB03	Core RDBMS	Oracle Net	Create Session	No	5.5	Network	Low	Single	Partial+	Partial+	None	9.0.1.5+, 9.2.0.8, 9.2.0.8DV, 10.1.0.5, 10.2.0.3	See Note 1
DB04	Oracle Secure Enterprise Search or Ultrasearch	Oracle Net	Execute on WKSYS.WK_QRY or WKSYS.WK_QUERYAPI	No	5.5	Network	Low	Single	Partial+	Partial+	None	9.0.1.5+, **9.2.0.8**, 9.2.0.8DV, 10.1.0.5, 10.2.0.3	See Note 1
DB05	Oracle Spatial	Oracle Net	Execute on SDO_UTIL	No	5.5	Network	Low	Single	Partial	Partial	None	10.1.0.5, 10.2.0.3	
DB06	Oracle Spatial	Oracle Net	Execute on SDO_GEOM	No	5.5	Network	Low	Single	Partial	Partial	None	9.0.1.5+, **9.2.0.8**, 9.2.0.8DV, 10.1.0.5, 10.2.0.3	See Note 1
DB07	Oracle Spatial	Oracle Net	Execute on SDO_IDX	No	5.5	Network	Low	Single	Partial	Partial	None	9.0.1.5+, **9.2.0.8**, 9.2.0.8DV, 10.1.0.5, 10.2.0.3, **11**.1.0.6	See Note 1
DB08	Authentication	Oracle Net	None	Yes	5.0	Network	Low	None	Partial	None	None	**11**.1.0.6	
DB09	Oracle Net Services	Local	None	No	4.6	Local	Low	None	Partial+	Partial+	Partial+	9.2.0.8, 10.1.0.5, 10.2.0.3	See Note 2

Figure 2.2 Sample from a database risk matrix in a CPU.

first being authenticated. The next thing provided per vulnerability is a Common Vulnerability Scoring System (CVSS) score. CVSS is a standardized method for assessing security vulnerabilities in all systems. CVSS has been used by Oracle since October 2006—before then vulnerabilities were scored using a proprietary scoring system.

CVSS scores range between 0.0 and 10.0 with 10.0 being the worst possible score (implying the worst possible vulnerability, and the highest risk). Oracle currently uses CVSS version 2.0 to derive a base score and this score is included in the risk matrix. Scores are computed using a calculator available at nvd.nist.gov and shown in Figure 2.3. The score is computed after you enter a selection for each of the entries. When Oracle scores vulnerabilities they key in the answers to the base score

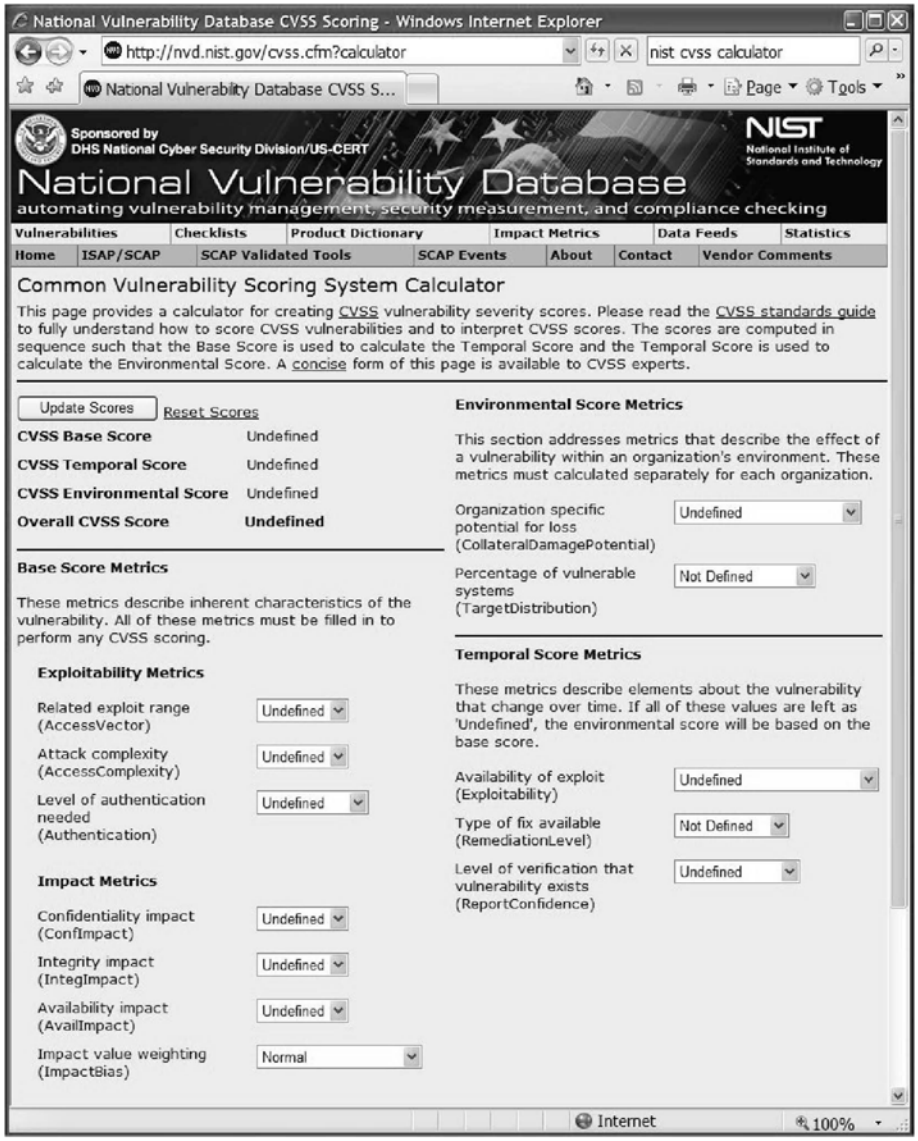

Figure 2.3 NIST CVSS calculator.

metrics and get a base score which then goes into the CPU risk matrix. Note that although the CVSS scores desire to be a single number through which you can tell right away whether it is critical or not, CVSS ratings depend on Oracle's interpretation of the problem. To fully understand how Oracle fills in the entries for computing CVSS scores see Metalink Note 394487.1.

It is especially important to understand how Oracle interprets the effect on confidentiality, availability, and integrity. The CVSS calculator allows you to enter one of three values—None, Partial, or Complete. Complete is defined to mean that the impact is to the "whole system." The question is what exactly this means. One interpretation of Complete can be that the impact is to all Oracle software running on a system and the other can be that the impact is to all software running on the system—OS and all. Oracle chooses to use the latter interpretation. This means for example that if the vulnerability affects all data within the database—all tables in all schemas—the CVSS score is still going to be based on a "Partial" selection in the calculator. Having chosen the first interpretation would have created higher CVSS scores. This is not uncommon and is the way most vendors interpret CVSS levels but you should understand this when you determine what your threshold for risk is.

To distinguish between cases in which the impact can be limited versus wide (which were the terms used before adoption of CVSS), Oracle introduces another level called Partial + (see Figure 2.2). A vulnerability that affects a limited set of resources (e.g., a specific database table) will have Partial in the risk matrix. If the vulnerability affects a wide range of resources (e.g., all tables in the database) then the impact is logged as Partial +. Note that this does not impact the CVSS score—the score in both cases is computed using Partial! It is therefore important to look at the entries in the risk matrix and not only the CVSS score.

Database n-Apply CPUs

Until 2007 CPUs included one monolithic patch for the database every quarter. It was impossible to install a subset of the fixes. This changed with the CPU of July 2007 when the n-apply patch format was introduced to CPUs. n-apply CPUs have the following benefits:

1. Customized patch conflict resolution.
2. Elimination of rollbacks and reinstallation of CPU patches that are already installed to limit downtime. CPUs are still cumulative but the installation process has been improved.
3. Ability to install only parts of the CPU fixes rather than have to install the whole CPU.

An n-apply CPU is a zip file that contains molecules and are installed using opatch. Each molecule is a group of security fixes. A molecule is an independent patch that does not conflict with any of the other molecules within the CPU.

Five Things to Remember about CPUs

1. CPUs are released every three months at specific dates so that you can plan ahead for testing and deploying fixes.
2. CPUs include security fixes to discovered vulnerabilities. It is very important to apply security fixes because this is the best way to protect yourself from attacks that exploit such vulnerabilities. Note that once a CPU has been released it is easier for an attacker to find the vulnerability because there is some information (e.g., the component) listed per vulnerability—therefore, apply CPUs as soon as possible while going through a standard testing and change management process.

(*continued*)

(continued)

3. CPUs include a risk matrix that allows you to determine how critical these fixes are for your environments. Risk matrices list which components are affected and how severe the issue is—all are there to help you decide whether or not you can tolerate the risk if you can't immediately apply the CPU.
4. CPUs are cumulative—if you apply the latest CPU you have included all fixes for all previous vulnerabilities too.
5. The new n-apply CPU packaging allows you to deploy fixes to only some vulnerabilities versus the old format which was delivered as a single patch.

2.5 HOWTO Sanitize Data for Test

As part of the section on Database Software Development the Database STIG states:

> (DG0076: CAT II) The DBA will ensure that export data from a production database used to populate a development database has all sensitive data such as payroll data or personal information, etc., removed or modified prior to import to the development database.

Production databases usually have better access controls and are monitored with higher scrutiny than development databases. This only makes sense if you assume that the sensitivity levels of data in development and test databases are lower than those of production servers. Developers have access to development and test databases but usually not to production servers (outside a "break glass," or emergency maintenance event). Therefore, you must sanitize data before you can load it to development servers.

Sanitizing data is hard. Not only do you need to know where all your sensitive data resides, you also need to change a lot of data in a way that does not invalidate your application and your tests. You can't change data randomly. If you have foreign keys (whether they are constraints in the database or managed by the application) you need to make sure that keys are preserved and that all references stay intact. Some fields encode application logic. For example, many account numbers follow a certain scheme in which there is a checksum or some algorithm for generating numbers—not any random set of digits is a valid account number. Another example is credit card numbers. When you sanitize data you need to preserve this property or your application will break. And finally, you need to make sure that after you sanitize the data, columns used for indexes maintain a statistical distribution which is close to that of the production data. Otherwise, performance tests on the sanitized data may not be indicative of the performance you will have on the production data. All in all, sanitizing test data is a hard problem to solve—that is why many development organizations don't do it and simply use the production data as-is. This is in violation of any security guideline, and tools do exist to help you accomplish this task.

The first tool available to you is the Data Masking option in Enterprise Manager. Follow these steps once you have enabled the data masking option in Enterprise Management Grid Control:

Step 1: Log onto EM
Step 2: Click on the Targets tab and the Databases subtab.
Step 3: Select the database where you want to mask sensitive data.

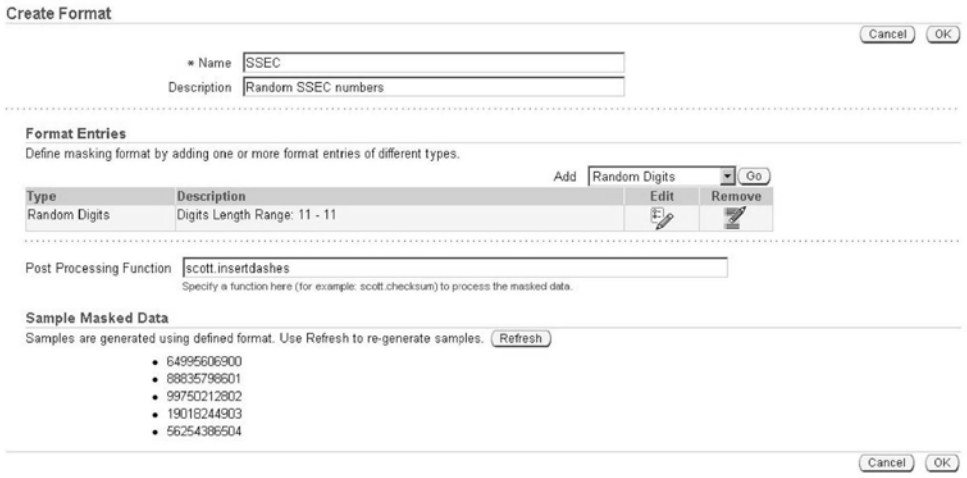

Figure 2.4 Defining a masking format.

Step 4: Click on the Administration link. At the bottom right is a Data Masking section. The Definitions link lets you set the masking rules. The Format Library link lets you build up a library of data masking formats. A data masking format defines how you want to mask sensitive data. For example, you can use random numbers or random characters. For more advanced functions (and usually you need more advanced obfuscation techniques for real applications) you need to build PL/SQL routines. For example, if you want to test an application you probably will want data that may have indexes with the same statistical distribution as the real data—using random characters will certainly not preserve cardinalities.

Step 5: Click on the Format Library link. This takes you to a page with a list of formats available to you. As this product matures, there will be many predefined formats for the most common identity patterns but for now click on Create.

Step 6: Figure 2.4 shows a format for masking social security numbers. These numbers have a pattern of [0–9]{3}-[0–9]{2}-[0–9]{4}. In this case you can choose Random digits from the drop down and click on Go. Enter 1 as the start and 11 as the end to ask Oracle to create 11 random digits for you. Click on OK. Then, you'll have to call a PL/SQL procedure to put in the dashes in locations 4 and 7, so enter the name of your procedure and click OK.

Step 7: Click on the Masking Definitions breadcrumb at the top to take you back to the masking definitions screen. Click on Mask to create a masking job. Give the job a name and select the database where the sensitive data resides.

Step 8: Click on Add to define which column to mask and how to mask it. Put in the schema name and click on the search icon. Select the sensitive column from the list (or multiple columns). Click on Define Format and Add.

Step 9: Click on Import From Library because you have already created the masking format. Select your format and click Import. You've now selected where the sensitive data is and how to mask it as shown in Figure 2.5. Click on Next.

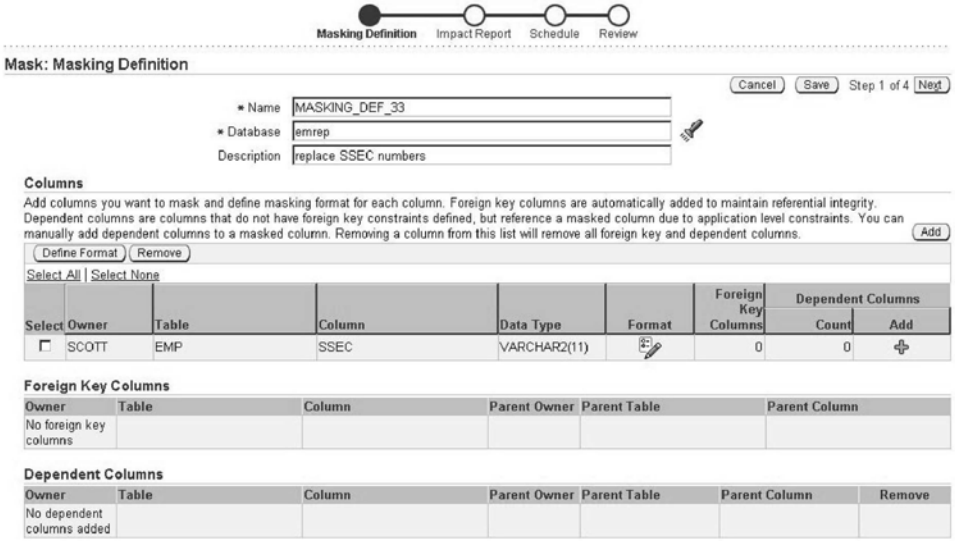

Figure 2.5 Defining which sensitive data to mask.

Step 10: The script is generated. Review the generation information and click Next. Enter the host credentials where the script will be stored. Enter a schedule if the job should be scheduled or select Immediately. Click on Next.

Step 11: Oracle produces the script and you can review it as shown in Figure 2.6. Submit the masking job. You can then review the status (and possible errors) of masking jobs as shown in Figure 2.7.

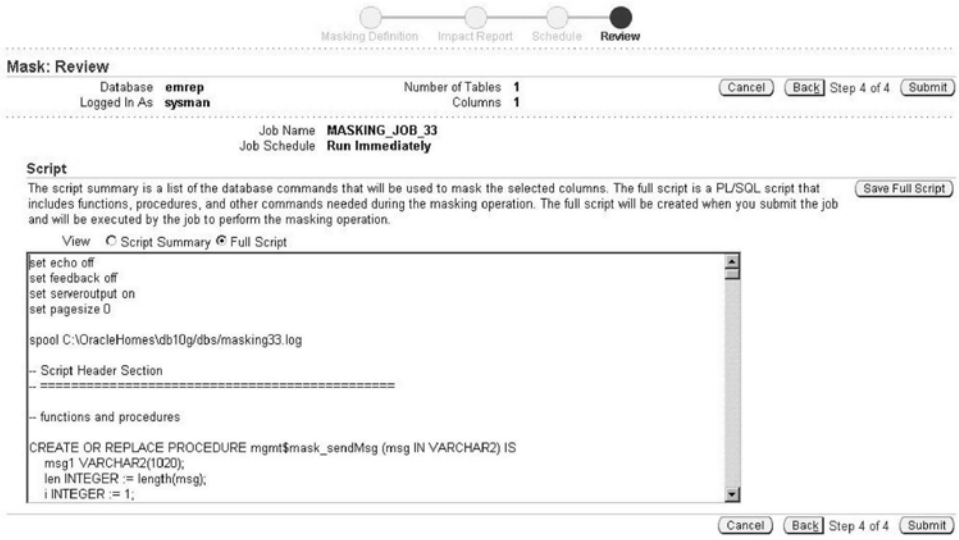

Figure 2.6 Reviewing the masking script and submitting the masking job.

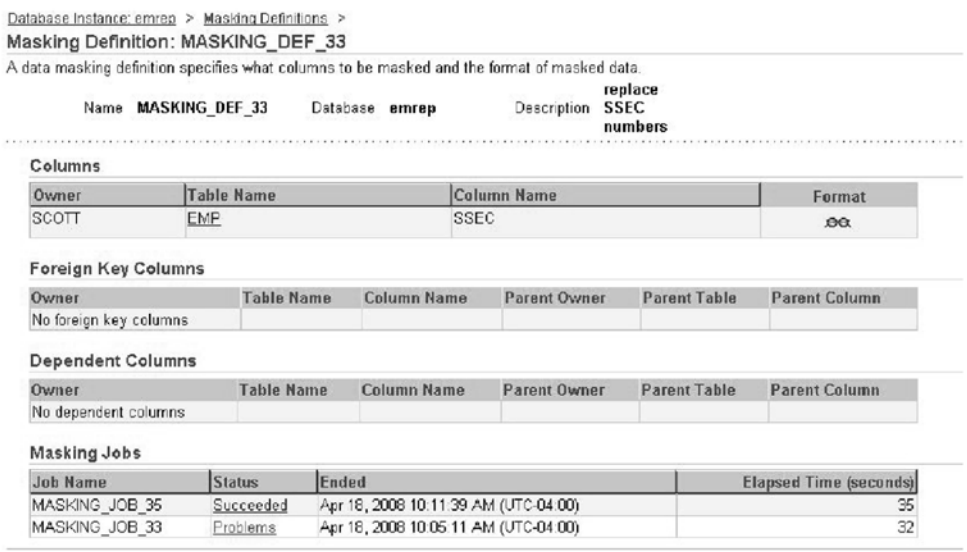

Figure 2.7 Reviewing the status of masking jobs.

As an example, if you use random digits and a PL/SQL procedure that adds the dashes, and if the data looks like:

```
SQL> select * from emp;

     EMPNO ENAME   JOB              MGR HIREDATE       SAL      COMM DEPTNO SSEC
---------- ------- ---------  -------- --------- --------- --------- ------ -----------
      7499 ALLEN   SALESMAN       7698 20-FEB-81       100       300     30 111-11-1111
      7876 ADAMS   CLERK          7788 23-MAY-87       101                20 222-22-2222
      7844 TURNER  SALESMAN       7698 08-SEP-81       102         0      30 333-33-3333
      7782 CLARK   MANAGER        7839 09-JUN-81       103                10 444-44-4444
      7566 JONES   MANAGER        7839 02-APR-81       104                20 000-00-0000
      7839 KING    PRESIDENT           17-NOV-81       105                10 888-88-8888
      7900 JAMES   CLERK          7698 03-DEC-81       106                30 777-77-7777
      7788 SCOTT   ANALYST        7566 19-APR-87       108                20 555-55-5555
      7369 SMITH   CLERK          7902 17-DEC-80       110                20 999-99-9999
      7934 MILLER  CLERK          7782 23-JAN-82       111                10 666-66-6666
```

Then, your masked data will look like:

```
SQL> select * from emp;

     EMPNO ENAME   JOB              MGR HIREDATE       SAL      COMM DEPTNO  SSEC
---------- ------- ---------  -------- --------- --------- --------- ------  -----------
      7844 TURNER  SALESMAN       7698 08-SEP-81       102         0      30  745-80-4600
      7782 CLARK   MANAGER        7839 09-JUN-81       103                10  857-87-7301
      7934 MILLER  CLERK          7782 23-JAN-82       111                10  522-91-5302
      7566 JONES   MANAGER        7839 02-APR-81       104                20  658-07-6603
      7839 KING    PRESIDENT           17-NOV-81       105                10  886-14-8204
      7788 SCOTT   ANALYST        7566 19-APR-87       108                20  661-97-3505
```

```
7369 SMITH   CLERK          7902 17-DEC-80      110                20  995-32-6006
7900 JAMES   CLERK          7698 03-DEC-81      106                30  821-79-0807
7876 ADAMS   CLERK          7788 23-MAY-87      101                20  420-95-5508
7499 ALLEN   SALESMAN       7698 20-FEB-81      100        300     30  375-11-5609
```

The Data Masking option is a new product and therefore has only rudimentary formats. With time the masking format library will grow and you will get logic-preserving and statistically preserving formats. However, there is a large set of third-party tools that perform this function and have a mature set of operators and formats. Examples include Princeton Softech (now IBM Optim), Application, Solix, and HP/Outerbay.

Three Things to Remember about Sanitizing Test Data

1. You must sanitize sensitive information if you generate test data by copying real data from production systems.
2. Sanitizing data is far from trivial—you cannot simply replace data with random strings or numbers. You have to preserve application logic which is often coded into data and you must preserve statistical distribution for performance tests to be valid.
3. You should use tools to sanitize data—use either the data making pack that is now part of Enterprise Management Grid Control or use third-party tools.

2.6 Discussion: Defense in Depth

All modern information security is founded on a concept called defense in depth. Defense in depth involves multiple layers of defense that increase the cost of an attack and places multiple barriers between an attacker and your computing resources. Multiple techniques and systems help mitigate the impact when one component of the defense is compromised or circumvented. The deeper an attacker tries to go the harder it gets. In addition to protecting your resources better, by the mere fact that there are multiple layers, defense in depth naturally provides areas in which you can put systems that can monitor and identify intrusions. This can often buy time to detect and respond to a breach and reduce its impact.

The term "defense in depth" is derived from a military strategy called defense in depth (also known as deep defense or elastic defense). This military strategy seeks to delay the advance of an attacker rather that prevent the advance. This buys time and causes enemy casualties—so rather than defeating an attacker with a single line, defense in depth relies on the tendency of an attack to lose momentum over a period of time or when it has to cover a larger area. The defender can yield some lines and territories causing the attacker to spread. This allows the defender to identify and mount counterattacks on the attacker's weak points.

Defense in depth is considered the only viable strategy for information systems. The reason is that there is no such thing as a perfect security layer. Anything can be cracked and every system has bugs. Moreover—the quality of both security and attacks are highly correlated with their cost. The goal of a good IT security implementation is to create an environment in which an attack will cost too much to be worth it—and do it all within a reasonable budget. Therefore, relying

on a single super-duper security system just does not work. Instead, build multiple good (but perhaps not perfect) security layers. This has been described in an excellent paper called "Defense in Depth—A practical strategy for achieving information assurance in today's highly networked environments" published by the National Security Agency (NSA)—http://www.nsa.gov/snac/support/defenseindepth.pdf.

Securing Oracle environments must follow the same strategy. The hardening checklists clearly discuss multiple types of activities that, if done in concert, implement a best practice in Oracle security. Remember that the security techniques available within the database can (and should) be augmented with security systems sitting outside the database—be they network security systems, database activity monitoring systems or host security. Always think of the main thesis of defense in depth—don't rely on one layer only.

Database security

HOWTO Secure and Audit Oracle 10g and 11g

Ron Ben Natan

Foreword by Pete Finnigan

Oracle is the number one database engine in use today. The fact that it is the choice of military organizations and agencies around the world is part of the company's legacy and is evident in the product. Oracle has more security-related functions, products, and tools than almost any other database engine. Unfortunately, the fact that these capabilities exist does not mean that they are used correctly or even used at all. In fact, most users are familiar with less than 20 percent of the security mechanisms within Oracle.

Written by Ron Ben Natan, one of the most respected and knowledgeable database security experts in the world, ***HOWTO Secure and Audit Oracle 10g and 11g*** shows readers how to navigate the options, select the right tools and avoid common pitfalls. The text is structured as *HOWTOs* — addressing each security function in the context of Oracle 11g and Oracle 10g.

Among a long list of *HOWTOs*, readers will learn to —

- Choose configuration settings that make it harder to gain unauthorized access
- Understand when and how to encrypt data-at-rest and data-in-transit and how to implement strong authentication
- Use and manage audit trails, and advanced techniques for auditing
- Assess risks that may exist and determine how to address them
- Make use of advanced tools and options such as Advanced Security Options, Virtual Private Database, Audit Vault, and Database Vault

The text also provides an overview of cryptography, covering encryption and digital signatures and shows readers how Oracle Wallet Manager and orapki can be used to generate and manage certificates and other secrets.

While the book's 17 chapters follow a logical order of implementation, each *HOWTO* can be referenced independently to meet a user's immediate needs. Providing authoritative and succinct instructions highlighted by examples, this ultimate guide to security best practices for Oracle bridges the gap between those who install and configure security features and those who secure and audit them.

CRC Press
Taylor & Francis Group
an **informa** business
www.taylorandfrancisgroup.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
270 Madison Avenue
New York, NY 10016
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

AU4127
ISBN: 978-1-4200-8412-2
90000
9 781420 084122

www.auerbach-publications.com