SYNGRESS®

4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP

4 FREE E-BOOKLETS

Practical Oracle Security

YOUR UNAUTHORIZED GUIDE TO RELATIONAL DATABASE SECURITY

Your Hands-On Guide for Securing Your Oracle Database

- Bonus Companion Web Site Contains Dozens of Scripts to Help You Automate Security Tasks
- Step-by-Step Instructions for Securely Integrating Your Database into the Enterprise
- Understand How to Implementy Security Measures Using Virtual Private Databases

Josh Shaul
Aaron Ingram

SYNGRESS®

Practical Oracle Security

YOUR UNAUTHORIZED GUIDE TO RELATIONAL DATABASE SECURITY

Josh Shaul
Aaron Ingram

PUBLISHED BY
Syngress Publishing, Inc.
800 Hingham Street
Rockland, MA 02370

Practical Oracle Security

Printed in the United States of America
1 2 3 4 5 6 7 8 9 0
ISBN 13: 978-1-59749-198-3

Publisher: Amorette Pedersen
Acquisitions Editor: Andrew Williams
Technical Editor: Mike Petruzzi
Cover Designer: Michael Kavish
Page Layout and Art: Patricia Lupien
Copy Editor: Judy Eby
Indexer: Odessa&Cie

For information on rights, translations, and bulk sales, contact Matt Pedersen, Director of Sales and Rights, at Syngress Publishing; email matt@syngress.com or fax to 781-681-3585.

Author Acknowledgments

From Aaron
To Irving Pector, Dennis Shasha, his co-author Josh and his parents for their endless encouragement.
To his mentors past and present for their wisdom: Bill Bond, Duane Schwartz, Hans Delly, Bob Fitterman, and Candace Martin

From Josh
To my wife Jill and my daughter Marlee Rose who endured many long nights with none of my attention.
To my mentors in the security world Tim Ober and Aaron Newman who taught me much of what I know.
And to my parents, for making me believe that success is possible in any endeavor.

From Both
To the crew at AppSecInc who made this all possible: Esteban, Cesar, Sean, Eric, Aaron, John, Tom, and Team SHATTER.

Authors

Aaron Ingram has fifteen years experience developing enterprise software, focusing on database systems and security applications. After graduating with a Bachelor's degree in computer science from Columbia University, he worked at Accenture as a consultant for Fortune 500 financial and telecommunication companies and for various government agencies. He then worked for ShieldIP creating Digital Rights Protection technology. Most recently, he merged his extensive database background with his security skills to manage the development of Application Security's real-time database intrusion detection and security auditing solution, AppRadar.

Josh Shaul got started in the security industry with SafeNet, Inc. in 1997, working on the industry's first complete IPsec accelerator chip. During a five year tenure as a SafeNet developer, Josh spent time designing, developing and enhancing SafeNet's embedded security solutions for a wide range of applications. For the last four years Josh has focused primarily on field engineering, helping companies deploy security software and hardware into various Networking Devices, Systems-on-a-chip (SoCs), and Processing Platforms. He is an expert on security protocols and standards, trusted computing, and application level security. Recently, Josh has focused primarily on database security, working to assist large organizations in developing the proper defense-in-depth strategy to secure sensitive data at its source. Josh is currently responsible for Worldwide Systems Engineering at Application Security, Inc.

Technical Editor

Mike Petruzzi is a senior penetration tester in the Washington, D.C. area. Mike has performed a variety of tasks and assumed multiple responsibilities in the information systems arena. He has been responsible for performing the role of Program Manager and InfoSec Engineer, System Administrator and Help Desk Technician and Technical Lead for companies such as IKON and SAIC. Mike also has extensive experience performing risk assessments, vulnerability assessments and certification and accreditation. Mike's background includes positions as a brewery representative, liquor salesman, and cook at a greasy spoon diner.

Contents

Chapter 1

Oracle Security: The Big Picture

Solutions in this chapter:

- A Brief History of Security Features in Oracle
- The Regulatory Environment Driving Database Security
- Major Data Theft Incidents
- A Step-by-step Approach to Securing Oracle

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

A senior database manager at one of the world's largest banks once told me that the best way to secure Oracle is to unplug it from the wall…and he is probably right. In fact, this holds true for nearly every networked application. Unfortunately, for many of us turning off the database is not an option; we must find another way to secure our systems. New technologies and services drive revenue for businesses, particularly those that provide a tailored experience to customers. These systems frequently require storing, processing, and offering access to personal information. No different are the systems that store corporate secrets or financial information. Much of the data they store is extremely sensitive, but it all needs to be readily and easily accessible nonetheless. This creates a significant data security challenge, one we will address in detail throughout this volume.

This book is designed to help you establish a practical security program for Oracle databases. We will create a means for measuring and assessing the security of your databases, and give you tools to create a security scorecard for each of your Oracle databases. This is not a Database Administrator (DBA) handbook—far from it. Instead, we are writing to the entire database security community, which includes DBAs, but also includes Information Technology (IT) security staff, auditors, and even the Chief Information Security Officer (CISO).

Oracle is by far the most widely deployed database in the world. It is a central component of critical systems across nearly every major industry that thrives today. In the financial, medical, telecom, infrastructure, government, and even the military, the databases and the data within them are the business. Over the last several years, databases have become a frequent target of cyber attacks. At first, these attacks were primarily intended to cause disruptions in business and gain notoriety among the hacker community. This has changed dramatically with database attacks increasingly focused on extracting the sensitive and valuable information from systems in an attempt at monetary gains by the attacker. Stealing personal information for use in identity theft, stealing credit card numbers to make purchases at will, stealing corporate secrets to take back the competitor's edge, these are today's drivers behind attacks on databases. Because of Oracle's dominance in the marketplace, many of these attacks have been focused on Oracle systems. Oracle has been an unwilling conspirator in these attacks, having built a system riddled with vulnerabilities for the attackers to go after. At the same time, Oracle has built the most complete suite of security features in any commercial database. This book will show you how to properly use these features to ensure your databases remain available and secure in a diverse and rapidly changing environment.

A Brief History of Security Features in Oracle

The Oracle database was born out of a U.S. intelligence community project, called Project Oracle, run by the Central Intelligence Agency. Given the initial customer, security was a serious concern from day one. If you go back to the initial release of Oracle from 1979 (actually dubbed version 2) the beginnings of today's security system were already present. Those were the days when databases were housed in physically protected secure rooms, with no outside or network access. At the time, there was no means to access Oracle without being on the server itself. The concern about an outside attacker breaching the system was hardly even considered; however, the threat from an insider was a present concern. It is quite likely that this threat brought about some components of the often confusing and misleading error message system that remains in use today. For example, try selecting from a table or view that exists in Oracle that you do not have permissions to access. Oracle will respond with the following:

```
C:\>sqlplus scott/tiger

SQL*Plus: Release 10.2.0.1.0 - Production on Sat Sep 15 13:18:52 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select * from sys.user$;
select * from sys.user$;
                  *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Why tell the user that the object they requested does not exist, rather then provide a message telling them they do not have the proper permissions? It's simple. If a user does not have rights on an object, that user has no reason to know that the object exists. Handing out any more information than is absolutely necessary has never been good for security, regardless of what is being secured.

Passwords were present in version 2 as well, but the password management system was extremely basic. Overall, there was a minimal need for security when Oracle first got started, but underpinnings were put in place. As the technology world began to change, and the protections offered by physically securing the servers were no longer sufficient, Oracle changed as well and began to implement a complex array of increasingly sophisticated security features.

Privilege Controls

At first, controls on user rights and privileges were nearly nonexistent. Within a few years, Oracle introduced the concept of roles. Three roles were rigidly defined: *CONNECT*, *RESOURCE*, and *DBA*. Users could be assigned roles, but these roles could not be modified, nor could custom roles be created. All privileges were assigned directly to the roles, never to users. This continued until version 6 of the database was released, when Oracle introduced the capability to grant privileges to users. This was a significant improvement in the security system, allowing administrators to have much more granular control in giving out access to data on an as-needed basis. The CONNECT, RESOURCE, and DBA roles remained statically defined until version 7 of Oracle was released and the concept of user-defined roles was introduced. User-defined roles revolutionized the privilege control system in Oracle, allowing for efficient and flexible management of permissions by allowing them to be grouped together and assigned or removed in bulk. The Oracle7 role system remains in use today and will likely not be changed with the future release of Oracle11g. We'll cover using roles and privileges in detail in Chapter 5.

One more major development in the privilege control system came in Oracle8i: *invoker rights* procedures. Stored procedures have been around for quite some time, and were commonly used as they are today, to group complex functionality into a single procedure and then offer easy access through a simple interface. For a long time, all procedures were run with *definer rights*. This means that when a procedure is called, it runs at the privilege level of the definer (the user who owns the procedure), often the DBA. *Invoker rights* procedures are different. They run at the privilege level of the invoker (the user who ran the procedure). This offers flexibility to database developers who can now create procedures that access data and database functions without the worry of a user getting access to data they should never see.

Networking

Until version 5 of the database, Oracle offered no networking features. Database access was only permitted by directly connecting from the host operating system (OS). This required users to be able to access the server on which Oracle ran, making it impossible to implement any kind of distributed computing system. In version 5, SQL*Net 1.0 was introduced and with it came a dramatic change in how Oracle was used and how it was secured. It was no longer necessary to access the host OS to log in to the database; all that was needed was a network connection and some client software and the database could be accessed remotely. The introduction of SQL*Net had an important side effect. Users could now interact directly and exclusively with Oracle's immature user authentication system and could completely bypass the mature authentication features offered by the database host OS.

Oracle Advanced Networking Option

The Oracle Advanced Networking Option (ANO) was released with version 7.3 of the database in 1996. ANO was the first Oracle product to offer strong security for a networked database; its two primary features were network security and single sign-on.

Network Security

In Oracle's terms, network security means both confidentiality and integrity protection. The confidentiality part ensures that nobody can read the data as it crosses the network, while the integrity protection part ensures that nobody can change the data as it moves. Confidentiality was implemented using the symmetric key encryption algorithms Rivest Cipher 4 (RC4) and Data Encryption Standard (DES). The integrity protection was implemented with a cryptographic hash or message digest function called Message Digest 5 (MD5). Oracle chose strong algorithms and made attacks against the data as it traverses the network very difficult to execute.

NOTE

ANO was released in 1996 during the dark days when the US government held tightly to export controls on encryption products. This forced Oracle to offer two versions of ANO, one for US domestic use only and another for export. The export version was limited to the use of 40-bit encryption keys for both RC4 and DES, significantly watering down the strength of each encryption algorithm. Domestic versions used 56-bit DES and allowed for up to 128-bit RC4. In the years since, these export controls have been largely removed and Oracle now offers only one version of the network security product, now called the Advanced Security Option.

Single Sign-on

In an attempt to simplify password management for organizations, Oracle began to integrate with third-party providers of single sign-on (SSO) authentication systems. The intention was to integrate Oracle databases into enterprise SSO systems, allowing users to set a single strong password in one place, and then use the same account to access all systems. This allowed users to remember only one password, and it allowed administrators to force the use of strong passwords on their users. Furthermore, SSO makes user provisioning and password management simple, as all credentials are managed centrally. In the initial release, Oracle offered support for a number of SSO systems, including Kerberos, CyberSAFE, SecurID, Biometric, and DCE GSSAPI.

The i in Oracle8i

Oracle ushered in the Internet era with the release of Oracle8i in early 1999. Targeted at the eCommerce marketplace in the heat of the .com boom, Oracle had the right product at the right time to claim an enormous share of the online retail market. Touted as a platform for developing Internet applications, Oracle8i was built to face the Internet and store sensitive data. This represented a shift in how and where databases were used. Hackers started finding databases directly accessible from the Internet. Oracle hacking went mainstream.

Network security continued to be a strong suit for Oracle. They renamed SQL*Net as Net8 and renamed the Advanced Networking Option to Oracle Advanced Security (OAS). Major enhancements were made to both the network security and single sign-on components of OAS. Oracle added support for Secure Sockets Layer (SSL) (network authentication, encryption, and integrity protection) and Remote Authentication Dial-in User Service (RADIUS) (centralized user authentication). Both are standards-based systems that had been publicly reviewed, offering assurance to companies who didn't want to rely on Oracle alone to attest to the effectiveness of their security protocols. In the releases since 8i, Oracle has continued to offer enhancements and upgrades to the OAS product.

Auditing

Oracle has offered auditing features in the database since very early on. Capabilities were provided to audit log-ins, object access, and database actions (defined as anything that makes a change to a database object, such as CREATE TABLE or ALTER DATABASE). Each event would be captured and labeled as successful or failed. Early versions of Oracle auditing were somewhat limited. Database actions could only be audited by role (CONNECT, RESOURCE, and DBA), not by individual user. SYSDBA activities could not be audited at all. Audit data was stored inside the database in the SYS.AUD$ table, and needed to be periodically truncated by the DBA. However, Oracle has supported auditing of access to SYS.AUD$ from the beginning.

With the Oracle7 release, database triggers were introduced. This new functionality was useful in many areas, audit in particular. The built-in Oracle auditing system records only those events that have occurred; it does not record before and after values for data changes. Triggers could be used to do just that, triggering on an update or delete to capture the old value before replacing or removing it. While a trigger-based audit system had to be implemented manually, it was a useful and powerful addition to the native auditing capabilities of the database.

Oracle7 also brought about the capability to write audit data directly to a flat file instead of into the database. This provided administrators with needed flexibility and allowed for tighter access controls on the audit data. The restrictions on database action auditing were lifted, allowing auditing by an individual user instead of by role.

Oracle8 came with its own set of improvements to the audit system. Things got much more granular with the capability to enable auditing at the object, schema, and system level. Oracle also added several views to allow for simpler review and analysis of the stored audit data.

Fine Grained Auditing

Fine Grained Auditing (FGA) was first introduced with Oracle9i. A major improvement in Oracle's auditing capability, the first release of FGA focused on auditing of SELECT statements. Previously, Oracle auditing could record that a user had read from a table or view with a SELECT statement. While it was possible to know that a SELECT statement was issued and who issued it, it was not possible to determine what data was selected. FGA was designed to collect this information. Each event logged by FGA detailed the user, date/time, schema, table, columns accessed, the exact SQL statement issued including bind variables, and a system change number. This was a level of detail never before seen in a database auditing system, allowing for complete recreation of each audited event. By logging the exact SQL statement executed and the system change number, it was possible to determine exactly what data had been returned by the query. A DBA could use a flashback query to effectively restore the database to the point in time when the query was first executed and then run it again. The results would be the same, proving what data had been read from the system.

Oracle took this a step further by allowing administrators to audit access to individual rows by evaluating the row against a set of conditions supplied when auditing was configured. This allowed for auditing access to sensitive data only. The DBA could set up a column to indicate sensitivity, and then configure the audit system to capture access only to the rows labeled sensitive. This powerful feature had only one major shortcoming; it worked for SELECT only. There was no granular auditing of the Data Manipulation Language (DML) commands INSERT, UPDATE, or DELETE.

With the release of Oracle10g came another round of major improvements to the built-in auditing systems. FGA was enhanced with the capability to audit DML statements, providing the same level of detail for INSERT, UPDATE, and DELETE statements as had previously been supplied only for SELECT. The improvements in 10g were not limited to FGA, in fact, the entire audit system was overhauled to make all audit capabilities more granular and FGA-like. By setting the *audit_trail* parameter to *db_extended*, standard Oracle audit will capture both the exact SQL text executed, along with the values for any bind variables used for any query run against the database. Oracle DBAs now have a powerful mechanism to track exactly what their users are doing in the database.

Password Management

Oracle has included its own authentication and password management system since the very beginning. At first, the system was barebones. Each user got a password; the password was assigned and set by the DBA. Users had no means to change their own password, and Oracle had no automated controls for password management. Passwords never expired, never needed to be changed, and could be as simple or as complex as the DBA chose. Initially, the problem with this system was password distribution. Since the DBA had to set each user's password, they would also have to distribute the password to each user. This could be a challenge in a large organization with dozens or even hundreds of database users. Hand written notes, phone calls, and personal visits were commonly used to distribute the passwords, but that took time and users had to wait their turn to get their new password before being allowed into the database. At the time, it was not entirely uncommon for a DBA to simply give each user the same password. This made it easy to distribute the passwords, but created new security nightmares. Anyone with access could essentially log in with anybody else's account and privilege level. Things needed to change.

Before long, Oracle gave users the ability to alter their own password. This was a big improvement. The distribution problem was almost solved. DBAs still needed to set each user's initial password, and the same problems apply to distributing those initial passwords. However, the scope of the issue was drastically reduced. DBAs would instruct each user to change their initial password the first time they logged in to the database. They could even use the auditing system to ensure that a password change had been made. However, Oracle's password controls were still well behind those offered by the popular OSes of the time, which started to become a legitimate concern when Oracle databases began to accept connections from across the network.

Profiles

True password management features were first offered in the database in Oracle8, with a system called "Profiles." Profiles provide a means for setting controls on passwords, and then applying those controls to users in groups (in a manner very similar to how roles allow permissions to be managed in groups). DBAs could create custom profiles for each group of users, with controls tailored to each group's needs. A DEFAULT profile was provided as a catchall. Any users not explicitly assigned to a profile would be assigned the DEFAULT profile, ensuring that the password management features apply to everyone. This profile system remains in use today and has been largely unchanged since its initial release. Oracle now had password management on par with most operating systems.

- **FAILED_LOGIN_ATTEMPTS** This feature, often referred to as "account lockout," is designed to effectively thwart any password guessing attempts against the database. Without this control in place, an attacker can literally spend eternity attempting to break into Oracle by repeatedly guessing passwords and attempting to log in. No matter how strong or complex the passwords, given enough time, an attacker could "brute-force" the system and gain access. The account lockout feature prevents this attack by enforcing a threshold of failed login attempts before an account is disabled or locked, meaning it is no longer permitted access to the database, even if the correct password is supplied. By setting this parameter to a reasonable value, 5 for example, DBAs can ensure that brute-force password-guessing attempts will almost always fail, while giving users a few opportunities to make a mistake typing in their password before their account is locked. Once an account has been locked, a DBA must manually unlock it, unless the database is configured to do so automatically.

- **PASSWORD_LIFE_TIME** Even the strongest passwords need to be changed periodically, and relying on the DBA to remember when that time comes for each user leaves a big opportunity for forgetfulness. The PASSWORD_LIFE_TIME setting enforces password changes automatically after the lifetime, set in number of days, has expired. Once a user's password life time has passed, the database forces the user to change their password on the next login, denying access to the database until the password has been changed. Alternately, a DBA can set a grace period after a user's password has expired, during which the password will still work to gain access to the database. However, a warning message will be displayed, informing the user that their password has expired and must be changed soon.
- **PASSWORD_REUSE_TIME** In order to prevent users from trying to trick the password management system into letting them keep their existing password once it has expired, Oracle tracks password history and can enforce a minimum length of time before a password can be reused. Without this feature, when a user's password expires, they could easily change the password to some new value, allowing the database to log them in, then change the password right back to the old value. The password reuse time setting enforces a number of days before a password can be reused. This value can be set to UNLIMITED to ensure that a user can never use the same password twice.
- **PASSWORD_REUSE_MAX** Similar to PASSWORD_REUSE_TIME, the reuse max value controls how many times a user can reuse the same password before it is permanently banned.
- **PASSWORD_LOCK_TIME** As mentioned earlier, Oracle can be configured to automatically unlock accounts that were locked by the FAILED_LOGIN_ATTEMPTS control. The password lock time controls if

and when those accounts are automatically unlocked. This parameter is configured with a number of days for automatic unlock, or set to UNLIMITED to force the DBA to unlock accounts manually.

- **PASSWORD_GRACE_TIME**: Once a user's password life time has expired, they may be given a grace period during which they are asked, but not forced, to change their password. The duration of this grace period is controlled by the PASSWORD_GRACE_TIME parameter, set in days. Once the grace period has expired, a user must change their password in order to successfully log in to the database.
- **PASSWORD_VERIFY_FUNCTION** Probably the most powerful and flexible of all password management features is the PASSWORD_VERIFY_FUNCTION. This setting points Oracle to a user-defined function, typically written in C, that can enforce any complexity rules desired on a new password. Want to enforce a minimum length? Require users to include a digit? Special character? The password verify function can be as simple or complex as desired, the only limitation is your programming ability. Oracle comes with a default password verify function which enforces several controls. We will cover the password verify function in detail in Chapter 7.

Data Compartmentalization

In the database world, compartmentalization of data is something that is uniquely offered by Oracle. The concept involves classifying data elements, and then controlling access to those elements based on the classification and a user's access or security level. By assigning a security level and compartment to each row of data in the database, access can be tightly controlled on a row-by-row basis, even when permissions have been granted on the entire table. When queries are issued, the system compares the security level and compartments on the data being accessed with the security authorizations of the user executing the query. Only the rows that match the user's authorizations are accessible, enforcing mandatory access control.

Trusted Oracle7

Data compartmentalization features first appeared in an add-on product to Oracle7, called Trusted Oracle7, primarily driven by Oracle's clientele in the US military. Based on the Bell-LaPadula security model, Trusted Oracle7 came pre-configured with three security levels: *Confidential*, *Secret*, and *Top Secret*. By combining these levels with a set of compartments, say one for each project that uses the database, it was possible to create a hierarchical set of controls that limited each user to accessing only the data from their project(s) at their security level. At the top of the hierarchy, users could see data from any compartment with any security level. At the bottom, a user could be restricted to seeing only Confidential data (not Secret or Top Secret) for their one compartment (or project).

Trusted Oracle7 offered some significant benefits, but came with a significant amount of baggage, as the complexity of configuring and implementing the system could be quite daunting, particularly in a system hosting a dozen or more projects with millions of rows of data stored in the database. The system was deployed in some military and even a few commercial applications, but it was widely viewed as too burdensome and complex for broad market acceptance. Even after enhancing the product to allow for the utilization of user-defined roles to define compartments, the commercial world continued not to accept the product, and eventually it was redesigned and renamed.

TIP

The Bell-LaPadula model was invented by David Elliot Bell and Len LaPadula in 1973, in an effort to define a multi-level security policy for the US Department of Defense. The model defines a set of security labels, ranging from Top Secret down to Unclassified (or Public) that can be used to enforce controls on access to data. Bell-LaPadula is defined as a *state machine* with a clearly defined set of states and functions to transition from one state to the next. When implemented properly, a system can be proven to satisfy its security design requirements.

Beyond basic access controls, Bell-LaPadula enforces two main rules, called the *simple security property* and the **-property*. The simple security property ensures that a user cannot read data that is classified above their security level (no read up). This means that a user assigned the Secret classification can read both Secret and Confidential data, but cannot read anything labeled Top Secret. The *-property (read as star-policy) ensures that a user cannot write data that is classified below their security level (no write down). A user with the Secret classification can write Secret and Top Secret data, but cannot input any data that is classified as Confidential. A modification to the *-property, called the *strong *-property* restricts users to writing data at their own level only, never above or below.

Virtual Private Database

While Trusted Oracle7 proved too rigid and difficult to implement, it provided a feature that the market clearly desired: a mechanism to allow multiple users within the same schema to see only the data that applied to them. Consider an online retail system, where customers can log in and view the status of their pending orders. It's likely that the orders for all customers are stored in the same table and it's important that each user cannot view the orders that belong to others. Some means of access control is required. Before the release of Virtual Private Database (VPD), organizations generally implemented this access control in the application. A simple approach was taken. Construct a query that includes a WHERE clause that ensures only the current user's data is returned. This works great until the user finds a way to connect to the database directly, then all bets are off. Once connected directly, there is nothing to limit the data that a user can see. If they have rights on the Orders table, they have rights on all of the data in that table. VPD was introduced to eliminate this concern and enforce security within the database, so that no matter what application is used to connect, each user can only see their data.

VPD uses a simple mechanism to enforce this access control. By transparently appending a WHERE clause to every query a user runs, VPD can effectively limit access to data by matching each user to a set of labels stored with the data. Users are granted access to data with specified labels, the VPD is configured, and then Oracle does the rest.

Oracle Label Security

With the release of Oracle8i came the new version of Trusted Oracle7, now dubbed "Oracle Label Security." Based on the same classification levels as were used in Trusted Oracle7, Oracle Label Security was essentially a pre-configured VPD for military applications. Oracle Label Security came with several innovations, particularly around the capability to create custom configurations with user-defined labels and compartments. The tool also came with an intuitive graphical user interface (GUI) for configuration called "Oracle Policy Manager." The Policy Manager product allowed DBAs to set up policies, define labels and their functions, and control user authorization. Once the set up is complete, Oracle will create a VPD designed to enforce the desired policies and authorizations.

Label Security can be applied at either the schema or individual table level, offering complete flexibility. Most applications require only a small percentage of the data they store to be secured by Label. By allowing Label Security to be implemented on the few tables that require it, configuration and management of the database is vastly simplified over what was offered in Trusted Oracle7. Organizations now had a set of powerful tools to create a highly compartmentalized database, with effective access controls in place to ensure that users can only access the data they need to get their job done.

Oracle10g and Beyond

Oracle10g represents the state-of-the-art in database security. With more effective security features packed into the product than ever before, 10g and the newly released Oracle11g offer an unprecedented level of control over who can get into your database and what they can access while they are there, while ensuring that an audit trail is kept that can log everything that goes on. It is likely that Oracle databases offer more security features than any other piece of software that has ever been created.

While this all sounds great, with all these features comes tremendous complexity. Therein lies the problem. Complexity is bad for security. The more features and options you have, the more potential for misconfigurations. Even worse, the more complex the code, the more opportunities there are for making mistakes, the kind of mistakes that can void all of the fancy security features. Oracle is not immune to making coding mistakes. In fact, so many vulnerabilities have been found that Oracle has been forced to implement a quarterly patch release schedule, solely for fixing security holes in their products. Each quarter, more devastating vulnerabilities are announced and fixed, and with each release, more researchers and hackers jump into the fray, finding more and more vulnerabilities for the software giant to fix. Several of the vulnerabilities found thus far have been extremely disastrous. In more than 10 cases, vulnerabilities have been discovered that allow an unauthenticated user to connect to the database and assume the role of SYSDBA, taking complete control over the database and everything in it, regardless of the security features that are enabled at the time. This is a fascinating dichotomy, as Oracle is likely both the most secure and the most vulnerable database in existence today.

The Regulatory Environment Driving Database Security

Over the last several years, things have changed dramatically in the IT Security world. Data security has become a major focus area for both government and industry regulations, real regulations with real consequences for non-compliance. At the heart of any data security program must be a database security program, as most of the world's sensitive data spends 95+ percent of its time in a database, most commonly an Oracle database. We have all heard of Sarbanes-Oxley (SOX) , the U.S. Federal regulation enforcing strict control financial reporting practices for publicly traded companies, but there are a host of other regulations that govern data security in a similar way. Financial institutions must comply with the Gramm-Leach-Bliley Financial Services Modernization Act (GLBA), requiring protection of personally identifiable information. Health care institutions must comply with the Health Insurance Portability and Accountability Act (HIPAA), requiring protection of patient health information. Retailers and credit card processors must comply with the Payment Card Industry Data Security Standard (PCI-DSS), requiring strong protection of cardholder information. U.S. Federal government departments must comply with the Federal Information Security Management Act (FISMA), requiring proper safeguards to protect all sensitive data stored in Federal systems. The list goes on and on, with a large backlog of pending legislation dealing with data security currently working its way through both the state and Federal legislation process.

The world of the DBA has permanently changed. While security in the database was often ignored, or more commonly was left for the firewall and network team to handle, today's regulatory environment has changed all that. Inadequate security controls at the database level can now lead to fines, penalties, loss of business, and in extreme cases even jail time. Organizations are no longer left to their own devices to ensure the security of their systems. Today, third-party audit firms police data security under the auspices of auditing regulatory compliance. There is no longer a choice but to draft and enforce an effective security program around protecting the confidentiality and integrity of sensitive data. Database security has entered the limelight.

Let's examine some of the regulations that you are likely to run into which mandate that you secure your databases.

The Sarbanes-Oxley Act

Sarbanes-Oxley (SOX) is probably the most widely known regulation governing the protection of corporate data. Also known as the Public Company Accounting Reform and Investor Protection Act of 2002, SOX requires that all public companies implement effective internal controls around financial reporting, and mandates review of those controls by independent auditors. SOX was passed amidst a storm of corporate disclosure of illegal and irresponsible accounting practices led by Enron, WorldCom, and Tyco. The fury over re-establishing investor confidence was overwhelming, and when put to a vote the bill passed in the Senate 99 to 0 and in the House 423 to 3.

SOX includes several requirements that directly relate to data security, primarily focused around ensuring the integrity of financial information that will be reported to the public. SOX makes corporate Chief Executive Officers (CEOs) and Chief Financial Officers (CFOs) accountable for the accuracy of financial reports, requiring them to provide personal certification of each report released. Jail time is stipulated for those executives who purposefully misstate financial performance. Computer systems that store, process, and manage financial data are recognized as tightly coupled with the overall financial reporting process, and are therefore required to be secured. Typically, organizations implement strong access controls, auditing of access to financial reporting systems, strict segregation of duties, and a thorough vulnerability management process in order to comply with SOX and eliminate the potential for a mistake or attack sending an executive off to the big house.

The Gramm-Leach-Bliley Act

The Gramm-Leach-Bliley Financial Services Modernization Act (GLBA) passed in November 1999 in an effort to reform rules governing the financial institutions. The bill repeals the Glass-Steagall Act, allowing banks to offer investment, commercial banking, and insurance services. GLBA paved the way for mega-mergers in the financial services industry, including the combination of Citibank and Travelers Group, forming Citigroup, the largest financial institution in America.

GLBA includes two key rules which govern the collection, storage, protection, and disclosure of customer's personal financial information by financial institutions: the *Financial Privacy Rule* and the *Safeguards Rule*. The Safeguards Rule mandates financial institutions to develop and document an information security plan to protect client's personal data stored within their systems. The plan must include a process for performing risk analysis on existing systems and controls, a process to monitor access to personal information, and a program to test the effectiveness of the security controls in place. Since nearly all personal data stored by financial institutions is kept within a database, GLBA has direct implications on database security.

California Senate Bill 1386

Leading what has become a national charge, in 2003, California passed a bill requiring companies to disclose any incident where the unencrypted personal information was, or is, reasonably believed to have been acquired by an unauthorized person. Since the bill passed, several other states have enacted similar legislation, and it is only a matter of time before the Federal government passes a breach disclose bill as well (at the time of this writing, more than a dozen such Federal bills have been proposed).

The motivations behind California Senate Bill 1386 are clearly stipulated in the text of the law; privacy and financial security are at risk because of a significant increase in the incidences of identity theft. The bill notes a 108 percent year over year increase in identify theft cases in Los Angeles County in 2000. Before the passage of this bill, it was commonplace for organizations that experienced some kind of breach to keep it a secret, not even reporting the theft to law enforcement. Senate Bill 1386 changed all that, leading to what have become regular disclosures of major data breaches that have made headlines, embarrassing companies and devastating consumer confidence. The threat of disclosure alone has been enough to force many companies into establishing real programs for data security, often grounded within the database infrastructure.

The Health Insurance Portability and Accountability Act

Passed in 1996, the Health Insurance Portability and Accountability Act (HIPAA) is designed to protect workers and their families from losing their health insurance when they change or lose their jobs. HIPAA also establishes a set of Administrative Simplification provisions which serve several functions including creating national standards for electronic health care transactions and ensuring the security and privacy of Protected Health Information (PHI). PHI is interpreted as any data about medical records or health care payment history that can be linked to an individual.

HIPAA compliance requires that organizations implement administrative, physical, and technical safeguards to ensure the protection of PHI. Administrative safeguards are a documented set of procedures that demonstrate the mechanisms by which an organization will comply with the act. Physical safeguards are a set of controls designed to protect against an unauthorized person gaining physical access to protected data (for example, by taking a server or hard disk). Technical safeguards are access controls intended to ensure that only authorized individuals can gain logical access to view, modify, or delete protected data. This includes protecting data at rest while stored in a database, as well as data in transit while traversing the network.

The Payment Card Industry Data Security Standard

Before the Payment Card Industry issued their first Data Security Standard (PCI-DSS) in January 2005, each one of the major credit card companies had created their own set of standards for how their merchants, issuers, and acquirers should protect cardholder information. Visa CISP, MasterCard SDP, Discover DISC, Amex DSOP—it was an alphabet soup of standards that were similar to one another but never the same. For large merchants that accept each type of card, compliance to the letter of each standard was nearly impossible. In an effort to simplify compliance and achieve broad acceptance of a single, well-considered set of standards, the PCI Security Standards Council was founded by American Express, Discover, JCB, MasterCard, and Visa. This group has worked together to produce two revisions of the PCI-DSS. The latest version, 1.1, was approved in September 2006.

The PCI standard is significantly different than the government standards we have covered so far. PCI-DSS provides specific details on what steps must be taken in order to properly secure cardholder data. At the top level there are six categories of controls that must be implemented:

- Build and Maintain a Secure Network
- Protect Cardholder Data
- Maintain a Vulnerability Management Program
- Implement Strong Access Control Measures
- Regularly Monitor and Test Networks
- Maintain an Information Security Policy

The PCI DSS is a multifaceted security standard that includes requirements for security management, policies, procedures, network architecture, software design, and other critical protective measures. This comprehensive standard is intended to help organizations proactively protect customer account data.

The Federal Information Security Management Act

The Federal Information Security Management Act (FISMA) was enacted in 2002 as part of the E-Government Act, designed to modernize the inner workings of the US Federal government. Before FISMA came along, information security was largely neglected in the government, particularly by the civilian agencies. The situation was clear; there was little motivation or budget allocated to cyber security, so Congress intervened in an attempt to make implementing security controls a mandatory responsibility of government IT shops.

FISMA requires that any information system used or operated by a US Federal agency, including those run by contractors and others on behalf of the government, follow a set of prescribed security processes. These processes are not defined within the FISMA regulations, but rather FISMA makes reference to other pertinent standards and legislation, including Federal Information Processing Standards (FIPS) documents, National Institute of Standard and Technology (NIST) special publications, HIPAA, and the Privacy Act of 1974.

FISMA mandates that all Federal information systems be reviewed to determine the types of data contained within the system, and then categorized based on the damage that could be caused if the system's confidentiality, integrity, or availability were to become compromised. There is significant debate as to the effectiveness of FISMA; however, few will argue the fact that FISMA and its web of related standards is extremely complex. Minimum security requirements for Federal agencies are outlined in FIPS 200, which refers to security controls described in NIST SP 800-53 (Recommended Security Controls for Federal Information Systems). NIST 800-53 is further broken down into categories for various types of information systems, and describes both operations and technical safeguards that must be implemented for each. It should be no surprise that NIST has created documents in the 800-53 series that directly address databases and database security.

Compliance with FISMA is generally evaluated on a departmental level by the Office of the Inspector General (OIG). This process is referred to as certification and accreditation (C&A) and includes a review of the controls and processes in place, and then signoff that the controls and processes meet Federal standards. Typically, each system must pass the C&A process at least once every three years or whenever a major change is made to the system, whichever comes first.

Major Data Theft Incidents

Despite the myriad of regulations governing data security, and a clear increase in focus on security issues in general, there have been a deluge of successful thefts of data on a vast scale. There have been so many incidents made public that the Privacy Rights Clearinghouse was established as a publicly available chronology of data breaches, and a count of the number of personal records that have been disclosed since February 2005. At the time of this writing in 2007, over 167,000,000 records containing sensitive personal information have been compromised in several hundred incidents. Check out the latest chronology at www.privacyrights.org/ar/ChronDataBreaches.htm.

The problem is not entirely limited to theft. There have been a significant number of cases where information was disclosed inadvertently, often because of an innocent or silly mistake that seemingly lies outside the realm of data security. There have been far too many cases to cover them all here, too many even to cover just the really interesting ones, so we have chosen four incidents to describe in some detail. These examples are intended to paint a picture of the various ways in which sensitive data has become compromised in the recent past.

CardSystems Solutions—June 2005

In the spring of 2005, a hacker was able to exploit vulnerabilities at CardSystems Solutions in order to gain access to their internal network and retrieve detailed data on approximately 40 million credit card transactions that the company had stored in a database. CardSystems was an Atlanta-based credit card processing company, responsible for handling $15 billion dollars in annual transactions on behalf of more then 100,000 small- to medium-sized businesses. The attack was detected first by a MasterCard fraud detection system when several likely fraudulent transactions were detected. MasterCard was able to trace the cardholder information used back to CardSystems, whom they immediately notified of a possible breach. A short investigation by CardSystems confirmed that their systems had been hacked. While the exact details of the attack remain the secret of the credit card companies, several clues have been disclosed that allow us to piece together the most likely scenario for how the attack worked.

During September 2004, a hacker found vulnerabilities in an Internet-facing application that CardSystems customers used to access data. The attacker was able to gain access to the Web application, likely through an easily guessed password, and then begin the process of looking for ways to access the underlying servers and databases directly. The most common method for using a Web application to gain access to internal databases and other systems is via Structured Query Language (SQL) injection. The attacker was able to locate points in the application where weak input validation allowed him to inject SQL into some forms, and interact directly with the database that housed the application data. From there, the attacker gained access to the database server's operating system, likely using database functions to do so, such as *xp_cmdshell* on MS SQL Server and Sybase. Since databases generally run with full administrative access on their host servers, once the attacker could access the OS, they assumed full control. From there, a script was created on a server in the CardSystems internal network. The script was designed to search the network for a particular type of file that contains Credit Card Track Data (the data on the magnetic strip on the back of your credit cards), and then send that data to the attacker via File Transfer Protocol (FTP).

In clear violation of the credit card companies' data security rules, CardSystems had been storing files that contained complete track data for failed transactions. These files were the ones targeted and stolen by the script. The files contained a complete set of information on each transaction, including the cardholder's name, card number, expiration date, and CVV code. With this data, the attacker was able to initiate the fraudulent transactions that led to the attack being detected. Within 60 days of discovering the hack, Visa declared that they would revoke CardSystems' authority to process their transactions. American Express quickly followed. CardSystems was to pay the ultimate price for their negligence; they were out of business.

ChoicePoint—February 2005

The attack against ChoicePoint was more of a brilliantly executed con than what most of us think of as a hack. ChoicePoint is a data aggregator and information broker, collecting, mining, and selling information on the backgrounds and spending patterns of, well, of everyone. This data is then sold to insurance companies, loan officers, media companies, law enforcement, or really any business looking for background checks on potential employees or customers. The attackers took advantage of ChoicePoint's business model and poor customer screening processes to essentially convince them to hand over the data.

Attackers set up approximately 50 fake companies, giving them legitimate names and phone numbers, dreaming up business models, and even establishing false Tax IDs. They used these companies to open accounts with ChoicePoint, who were only too happy to sign up a few more customers and start feeding them data. Over a period of months, these companies requested and received data on 145,000 adults from ChoicePoint, acting and operating in much the same way as any of their legit customers. But these folks were not legit and not interested in targeted marketing campaigns. They were interested in identity theft, in destroying the finances of others for their own personal gains. The breach was eventually uncovered, but the exact data collected could not be determined. This is when things got ugly for ChoicePoint.

Making the mistake of allowing these accounts to be set up was shameful, and punitive action was certainly unavoidable, but when the prosecuting attorneys realized that ChoicePoint had no way to determine what records had been accessed, they went out for blood. ChoicePoint executives were subpoenaed to testify before Congress, where many difficult questions were asked about the lack of tracking and user auditing within the sensitive information systems that the company maintained. The incident cost ChoicePoint a fortune, kicked off a flurry of legislation governing data mining companies that collect and sell personal information, and made crystal clear to the business world that if you are going to store personal information, you'd better take steps to protect it and ensure that you properly track access to it, and retain the audit logs.

TJX—January 2007

Currently on record as the largest single incident of data theft of all time, information on nearly 50 million credit card transactions was disclosed to what appears to be a group of professional computer criminals. The attack was technically complex, devastatingly effective, and was in place providing data to the attackers for nearly two years before it was noticed. In a 2006 research study by the Ponemon Institute, 31 companies that had experienced a data breach were analyzed, and the cost to the company of each compromised record was estimated at $182.00. Using that number as a guideline, TJX could be forced to pay out nearly $9,000,000,000. This staggering figure does not include the less tangible costs such as loss of business, loss of productivity, brand damage, or loss of market capitalization. With risks on this scale, it is hard to believe that companies don't do more to secure their data. Hopefully, by following the steps in this book, you can help your organization avoid this ultimate nightmare scenario.

Details of the attack remain somewhat sketchy, but we have a general timeline of events and a sound theory of what went on. TJX first found evidence of unauthorized software on their systems on December 18, 2006. They immediately hired two major consulting firms to perform a detailed analysis of the incident and provide guidance on how to respond. Within a few days, it was becoming clear that the software was indeed malicious and that the intruder responsible continued to have access to critical financial systems within TJX's network, access that dated back to July 2005. The attack was multi-faceted, including elements of stealing files, intercepting communications, and breaking encryption on protected data.

Files stolen contained historical information about payment card transactions. TJX has not been able to completely identify which files were stolen, primarily because they periodically delete these files during the normal course of business. Some of the files that were likely accessed dated back to 2003, when security rules about protecting cardholder information were far more lax than they are today. These older files generally contained clear-text (unencrypted) data, giving the attacker easy access to the credit card details.

Taking the attack to the next level, the attacker was able to gain access to the network used during the payment card transaction approval process. During this process, cardholder information, including names, card numbers, expirations dates, and CVV2 numbers are transmitted to the payment card issuer without encryption. The intruder was able to watch this approval process in action, collecting the credit card data as it traversed the network.

The final blow to TJX was the discovery that the attacker had likely gained access to the software and decryption keys needed to decipher the card data that had been stored, properly encrypted, in their databases. Even when the company believed that it had strong security in place to protect their sensitive data, the attacker was able to find holes in the system and gain full access.

The methods used to initially penetrate the TJX network have not been disclosed, however many experts agree that it likely began with the attacker gaining access to an unprotected wireless network at one of TJX's retail stores. Sitting in a parked car in the parking lot, the attacker could have connected to the wireless network and attacked the system used to manage the store's cash registers. Once that system was breached (probably using a freeware hacker tool downloaded from the Internet), it became a simple matter to connect to the central processing systems in TJX headquarters. Corporate security is often compared to a Tootsie-Pop; hard and crunchy on the outside, soft and chewy in the middle. The analogy refers to strong security at the network perimeter, but little to no security inside the network. Once the hacker found his or her way past TJX's network perimeter, it was game over; weak internal security controls allowed them to steal everything but the kitchen sink.

Department of Veterans Affairs—May 2006

The “breach” at Veterans Affairs (VA) was an interesting one, as it demonstrates how a seemingly innocent (but poorly thought through) act can lead to the biggest inadvertent disclosure of sensitive information in the history of the US government. This case involved a break-in and a theft, but not in the way that you would expect. On May 3, 2006, a VA data analyst who had legitimate access to the VA information systems took an extract of Veteran’s names, dates of birth, and social security numbers. Approximately 27 million records were written to an external hard disk drive, and left attached to the analyst’s laptop. At the end of the day, the laptop left the building, going home with its rightful owner, with the external disk still attached. This was a clear violation of VA policies, but no technical controls had been implemented to either detect or stop this type of behavior.

That evening, the analyst’s house was broken into by a petty burglar looking to steal electronics and other small household valuables. Included among the thief’s booty was a laptop, the very same laptop that came home from work that day, along with that very important hard disk. The next morning, the break-in was noticed and the missing laptop reported to the VA. The data on the laptop was never an issue, but the hard disk was a big deal and while it seemed clear that the data was never the target of the theft, there was no way to know what had happened with the data once it was in criminal hands. A mad FBI search for the laptop began, lasting for about eight weeks before the missing machine and the all important external disk was found. But it was too late; the data was out there and there was no way to prove it had not been accessed or transferred to others (although the FBI did perform a forensic analysis and found no evidence that the data had been touched).

Everyone affected needed to be informed, credit monitoring services were contracted to watch for potential identity theft, and the government took action first requiring the tracking of all data extracts from systems containing sensitive data, and soon thereafter mandating hard drive encryption on all mobile PCs. VA also implemented more training and education programs on data handling policies, but apparently not enough. It was not even three months later before the VA experienced a similar breach, this time involving 18,000 records stolen with a laptop that a contractor for Unisys had taken home for the night.

A Step-by-step Approach to Securing Oracle

This book focuses on a practical, step-by-step approach to securing Oracle databases. We'll define three levels of Oracle security and provide you guidance on how to determine what level is appropriate for each of your systems, then explain exactly how to get yourself from where you are today to the right level for you. We'll take things a step or two beyond just helping you secure your Oracle databases. We will build a mechanism for tracking your progress, establishing a plan and then demonstrating continuous improvement. We will also create a system to measure and prove the security of your systems to both internal and external compliance officers and auditors, giving you the option of using a checklist and a set of scripts, or automating the process with commercially available tools. Your journey to secure Oracle databases begins here.

Tools & Traps…

One Size Fits All Doesn't Fit Most

We've seen quite a few companies try to start a comprehensive database security program only to end up nowhere but frustrated. The most common reason for their failure: establishing a "one size fits all" approach to securing their systems. The database environment within a typical enterprise is extremely diverse, and different systems have different needs. This makes it next to impossible to create one overarching set of standards to which every system must comply. The one size fits all approach can fail in several different ways, but the end result is always the same: no real database security program, and an easy target for any hacker who cares to attack.

The first step in the security program is to establish a plan and a set of guidelines, which is a common point of failure. Some companies will establish a working group in order to build a set of security and configuration guidelines. Problems often arise when the members of the working group, who often represent different functional business areas, are unable to come to agreement on common standards. In the most severe cases, this leads to the database security effort being abandoned before it really begins, leaving each administrator to decide on their own if and how they will protect their data. More commonly, working groups establish a watered down set of standards,

Continued

leaving plenty of room for the exceptions required by each business area, but also leaving a knowledgeable attacker with open doors into the systems.

Once a set of standards have been established, they are often passed to the database administration teams with little or no guidance on how to implement the requirements. The issue is not that the DBAs don't know how to configure the security settings; they know the databases better then anyone else. The problem lies in prioritization and workload. In organizations with dozens, hundreds, or thousands of databases, the task of implementing a security standard enterprise-wide can easily be overwhelming. When open questions are left about which systems to secure first, or how far to go with each system before moving on, efforts often fizzle out after tightly locking down a small handful of randomly chosen databases. Hackers will use automated tools to search for weaknesses across any database they can access; securing only a few is almost like securing none at all.

With a set of security configuration standards that explain how to secure each database based on business value and risk, and a clear plan that sets out a priority order in which to secure the databases, an enterprise can be extremely successful in implementing database security across the organization. This book will help you build that plan for your Oracle databases, but the lessons learned can easily be translated to secure any database platform.

Appropriate Security For Each Class of Database System

To get started, you will need to assign each of the databases you are responsible for a priority rating based primarily on its business value. This activity should not be performed in a vacuum. Rather you should get together with other stakeholders within your organization, folks such as the database administrators, IT security engineers, application owners, business line managers, and internal auditors to review and assign each system a business value rating. It is often easiest to use a set of categories rather than come up with an absolute priority list. Try using three categories, such as:

- Business Critical
    - Databases that must be running in order for a business to run. For example, databases running online retail shops, stock trading systems, or critical infrastructure systems.
    - Databases that contain data that if stolen could cause irreparable harm to the business. For example, databases containing credit card transaction

    data, corporate secrets, or sensitive personal information on customers or employees.
    - Databases that require protection in order to achieve regulatory compliance. For example, databases containing financial reporting data for SOX, personal health information for HIPAA, or credit card numbers for PCI-DSS.
- Business Impact
    - Databases supporting business operations. For example, databases hosting HR systems, inventory management systems, or customer support systems.
    - Databases supporting business intelligence and long-term decision-making. For example, databases containing historical financial data or marketing data.
- Everything Else
    - Databases that do not contain sensitive data.
    - Databases that are not required to support day-to-day business operations.
    - Development, QA, and test databases (not backup or disaster recovery systems; those belong in the same category as the primary systems they safeguard).

Once your databases have been assigned to categories, you can make smart decisions about which to secure first and how far to go in securing each one. Your most critical systems will get the highest level of security, while the least critical systems get the lowest level. This way, each database gets only the security features that it requires, eliminating the excess workload caused by forcing every database to be secured to the same standard of protection. Like a system of building blocks, each category takes over where the last leaves off, building a foundation of security and then expanding it to the needs of the system it is protecting. This allows you to quickly establish a baseline level of security on every database, preventing the vast majority of simple attacks and lowering business risk, then moving on to conquer more complex issues with more rigorous security standards on only the most sensitive databases.

Demonstrating Compliance

It is not enough to secure your databases; you have to prove it. Achieving compliance with anything from government regulations to internal security standards means providing evidence that you have taken the proper steps to secure your system. To enable you to demonstrate and communicate your Oracle security to whomever you must report to, each level includes a systematic approach to maintaining and assessing compliance. This includes references to checklists, scripts, and tools you can use to automate the assessment and monitoring process.

Summary

With a solid understanding of the history and evolution of security features in Oracle, we hope that you can understand how the many different security controls came to be and why they were added to the system. Government and Industry regulations have been established to govern the handling of sensitive personal and financial information, forcing companies to comply or face significant punitive actions. Even with increasing pressure to secure systems, major organizations have experienced high profile breaches, with costs as high as going out of business. Hacking is a type of theft that companies must target with real investments in people, process, and technology.

This book gives you a prescription to secure a single database, or to build an enterprise-wide strategy to lock down your critical Oracle database systems and tightly integrate your database security strategy with your corporate security infrastructure. By establishing a strong Oracle security program, you can play a lead role in ensuring your business meets regulatory requirements, properly protects corporate secrets, and acts as a good custodian of the sensitive personal and financial information that you store.

Solutions Fast Track

A Brief History of Security Features in Oracle

☑ At first, Oracle had only the most basic security features. However, security was a consideration from day one.

☑ Over time, Oracle evolved to add a slew of basic protections. These included authentication and password management, authorization and access controls, networking, and network security.

☑ Recently, Oracle has added a number of security products to apply to the database to achieve levels of security previously impossible.

The Regulatory Environment Driving Database Security

☑ After several incidents of misuse of personal information and of companies misleading investors with falsified financial information, governments have taken a stand and enacted legislation requiring companies to protect sensitive data that they collect and store.

- ☑ The commercial marketplace has taken steps on its own to regulate and protect sensitive data. Leading the charge is the PCI-DSS, demanding that merchants and banks take seriously the responsibility to protect credit card information
- ☑ Organizations that do not comply with the regulations that govern them are severely penalized. The risk of major fines and even jail time is enough to force organizations into compliance.

Major Data Theft Incidents

- ☑ Data theft has become a serious problem, with professional criminal rings turning to hacking as the next front for their enterprises.
- ☑ Large and prestigious institutions have been hacked, leaking tremendous amounts of data about customers, patients, employees, students, and everyone else. Mandatory disclosure of these incidents has led to great embarrassment and loss of business.
- ☑ CardSystems, ChoicePoint, TJX, and Veterans Affairs all experienced major data theft incidents. Each one was different, from sophisticated computer crime at TJX to a dumb mistake and a house burglar at VA. The methods are less important than the results. Companies that don't protect their data can and do lose their data.

A Step-by-step Approach to Securing Oracle

- ☑ Oracle security can be defined in various levels, with appropriate levels of security defined for each class of systems. Databases are classified based on business value and then security controls are mapped on to suit the business need.
- ☑ Build a security plan from the ground up, applying a base set of security controls to every database, and then expanding on those controls for the more sensitive and business critical systems.

☑ A system of measurements provides proof that each system has been secured and meets the requirements put forth by anything from government regulations to internal security configuration guidelines and standards.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: What is the top Oracle security issue you see out there today?

A: By far, it is default accounts with default passwords enabled in the database. Read more about that in Chapter 4.

Q: I've been a DBA for 20 years. Why am I only now being asked to secure my databases?

A: The world has changed. The bad guys have pulled off some pretty significant crimes, so the good guys have put controls in place to try and stop the attackers. Today, your business is faced with increasing regulatory pressure to secure your critical data, while at the same time there are more hackers than ever trying to get at it.

Q: My database is behind a firewall in a secure network, am I protected?

A: Not a chance. Today, most of the bad guys operate from inside your network. More then 70 percent of attacks involve an insider. Also, that firewall is full of holes, designed to allow applications to function. Attackers have found ways to exploit vulnerable applications to gain direct access to the databases.

Q: How much work is involved in securing an Oracle database?

A: Every database is different. Depending on the sensitivity of the data, the environment it operates in, and several other factors, the answer can be anything from an hour to a couple of weeks. The important thing is to take the task step-by-step, eliminating the most critical issues first and making real progress from the beginning. Security is never complete and no system is unbreakable, however, we can build significant defenses so that the effort to break in far outweighs any potential payoff to the attacker.

Chapter 2

File System

Solutions in this chapter:

- Getting to Know Your Files
- Reviewing Recommended Permissions
- Managing Change

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Oracle shares its host with other processes and users. Together they share the operating system and all of its resources, such as the file system. The file system contains your Oracle installation, including its data and other files. This means any user potentially has access to it, wittingly or unwittingly. Even a seemingly innocuous service may execute as a user having rights to read or write database files. If this service is hacked, even though it has nothing to do with Oracle, your files instantly are vulnerable. Because of this, it is critically important that you ensure access is restricted to only those user accounts that require it. Data files are particularly important to protect, but as you will learn in this chapter, there are several other types of files that you also must secure.

Attacks are increasingly initiated by sources internal to an organization, so the computer security arms race is moving inside the network. This is especially true given that intrusion detection technologies are constantly improving. Firewalls are not enough to lock down your systems. To be optimally secure, you must implement a layered approach. Every level of every application should be reviewed for security vulnerabilities, then locked down, monitored, and reviewed again. For databases, this starts with the host and in particular, the file system.

Getting to Know Your Files

Have you ever reviewed the list of files that come in an Oracle installation? There are over 15,000 of them by default, and Oracle creates more as you use your database. If the wrong one has privileges that are too permissive, you risk your database being knocked out by a Denial of Service (DoS) attack, data being stolen, data being lost forever, or worse, all three. It may seem like a daunting task to lock down all of them, so to make this easier, let's group them into classes, and discuss each class in turn. We'll review individual files that are of particular concern and some others that serve as good examples of each class.

NOTE

A DoS attack is one that reduces the capabilities of a computer service or completely prevents it from functioning. Users are unable to use the service in the same way as they normally can. For example, a DoS attack may send so much work to a service that it significantly slows response time for regular users, possibly to the point of being unusable. Another example is one that intentionally sends bad passwords until the service (e.g., Oracle, locks the account).

A DoS attack that can be executed by a user with write access to Oracle's *bin* directory is simply to delete the *oracle* executable thus preventing Oracle from running. A less trivial example is to replace *oracle* with a custom-made executable that can do anything an attacker wants.

The most critical files from a security perspective that make up an installation of Oracle are in four categories: *data*, *software*, *configuration*, and *logs*. Each plays a crucial role in your database and so each is important to protect. Data files contain all of your business-critical information; software files define how Oracle executes on the host; configuration files tell Oracle how to behave; and log files tell the reviewer how Oracle is behaving. They each have their uses to a hacker and all are important.

Data

Data is the essence of your database. Your database's entire job is to manage this information. The data files contain business secrets, private information about your customers, and a host of other knowledge hackers would love to get their hands on. In this section, we'll discuss the three main types of files in this category: *tablespaces*, *redo logs*, and *backups*. These files can be placed anywhere on the file system and very often they are placed in several places. If you are the database administrator, you will know their locations off-hand. If not, let's review a few queries that will help you find them.

TIP

Database security is all about protecting your data. Every permission you revoke, user account you remove, security patch you apply, and service you disable is all in a grand effort to protect your data. Protecting against significant, irrecoverable data loss is your primary concern. Protecting against data theft is second, a close second, but still second. It is one thing for someone else to get your data, but if you no longer have it yourself, your situation is much worse.

The files we discuss in this section are at the heart of the matter for both concerns, and are of the utmost importance to protect. When developing a security lockdown plan, the steps included in this chapter should be among the first on your list.

Tablespaces

Tablespaces are logical groups of *datafiles* where you place, for example, tables. A tablespace may contain one datafile or it may contain hundreds. This is up to the database administrator. Your role in securing Oracle is to know that they exist, to know their locations, and to ensure that they are properly protected. They contain different kinds of data from the perspective of the Oracle architecture. In particular, tablespaces contain tables, indexes, undo information, and temporary data.

Datafiles are the primary storage containers for your live data. They are the containers of your tablespaces (i.e., the files holding your tables, indexes, rollback information and temporary data). Every single bit of these files is valuable because all of the information that defines your database is in them. These are gold to a hacker. If he can read and write these, he's won. To see a full list of the ones in your system, run the query on the *v$datafile* system view shown in Figure 2.1.

Figure 2.1 Listing Tablespace Files

```
SQL> select name from v$datafile;

NAME
--------------------------------------------

/u01/oradata/ORDB/system01.dbf
/u01/oradata/ORDB/undotbs01.dbf
/u01/oradata/ORDB/cwmlite01.dbf
/u01/oradata/ORDB/drsys01.dbf
/u01/oradata/ORDB/example01.dbf
/u01/oradata/ORDB/indx01.dbf
/u01/oradata/ORDB/odm01.dbf
/u01/oradata/ORDB/tools01.dbf
/u01/oradata/ORDB/users01.dbf
/u01/oradata/ORDB/xdb01.dbf
/u02/oradata/ORDB/foo01.dbf
/u03/oradata/ORDB/bar01.dbf
```

Pay particular attention to the last two lines of Figure 2.1. Did you notice they are in different directories from each other and from the rest of the datafiles? The database administrator (DBA) in this case chose to place files in *u01*, in *u02* and in *u03*. A typical production system will distribute data across several disks to reduce contention. This means data files will reside under different mount points on the file system and thus in different directories. When listing files for your lockdown plan, be careful to note subtle directory differences such as *u01* versus *u02*.

WARNING

Oracle is only aware of the datafiles contained within the *v$datafile* view. It is possible, however, that unused datafiles exist on disk. The *DROP TABLESPACE* command does not drop datafiles unless the database administrator explicitly instructs it to do so by using the *INCLUDING CONTENTS AND DATAFILES* clause. The command

```
DROP TABLESPACE tblspace INCLUDING CONTENTS AND DATAFILES
```

drops all datafiles associated with the tablespace called, in this example, "tblspace." Be certain to check the alert log for errors after running this command just in case some file was not actually removed.

Datafiles have a known format and so it is trivial for a hacker with file system access to find them. To protect against this, ensure your DBA always removes or at least locks down datafiles that are not in use and develop a plan to verify that this has been done. In the "Managing Change" section of this chapter, we'll review some ways to accomplish this.

The reason for securing those datafiles that store tables is obvious, but it is also necessary to secure those that have indexes, undo information, and temporary space. Each serves a different purpose, but they all derive their data from tables. Technically speaking, not every one of these will have data that you absolutely need to lock down—this is a business decision—but you will find it easier and less risky to secure all of them rather than to spend time prioritizing individual files to secure.

Indexes

Indexes are data structures that dramatically improve performance of many Data Manipulation Language (DML) commands. The files that host them are important to protect, because they will almost always contain a copy of at least some data from many of the table columns that you might find in a *join*, a *where*, or an *order by* clause of a DML statement (e.g., a *select*, *update*, or *delete* statement). Let's say that you have a table such as the one below called *People* having columns for social security number (*SSN*), last name (*last_name*), and first name (*first_name*).

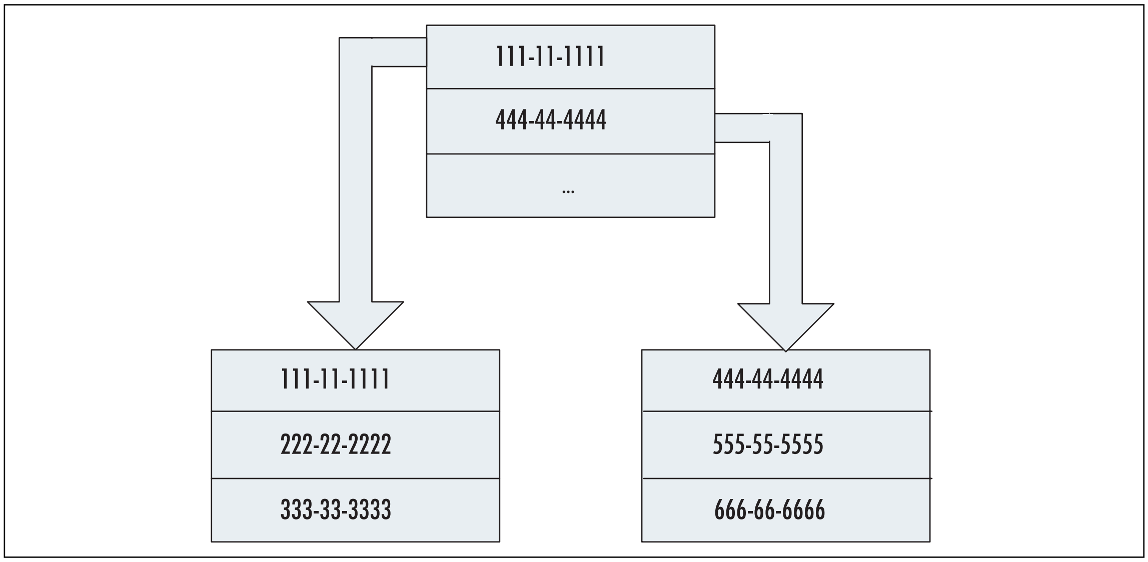
```
SSN         LAST_NAME  FIRST_NAME
----------- ---------- ----------
111-11-1111 Bond       William
222-22-2222 Buermann   Scott
333-33-3333 Ingram     Aaron
444-44-4444 Kulkarni   Sachin
555-55-5555 Pector     Irving
666-66-6666 Shaul      Joshua
```

For optimal performance for querying by SSN, your DBA would create an index on the SSN column. She or she may put that in a tablespace separate from the underlying table by using a command such as *create index*. For example, to create an index called *people_ssn* in the *idxtblspace* tablespace, use the following command:

```
CREATE INDEX people_ssn ON People (ssn) TABLESPACE idxtblspace;
```

Several different types of indexes are available, but the most common one is called a *b-tree*. B-trees store the values of the indexed column in the leaf pages of a tree, as depicted in Figure 2.2. In this example, the leaf pages of the index are at the bottom of the diagram and each of those pages contains up to three values. As you can see in the figure, an index on the SSN column causes Oracle to store social security numbers in an additional location separate from the underlying table. The table and index may or may not be in the same file, but either way, both should be represented in your lockdown plan.

Figure 2.2 B-tree Index



Undo Information

A Structured Query Language (SQL) script can undo a database transaction anytime up until it commits the transaction. This implies that Oracle must remember all data changes from the beginning of the transaction until either the script executes a *roll-back* to undo the changes, or until it executes a *commit* to save the changes permanently. Let's look at a batch insert of several records in the *People* table:

```
BEGIN TRANSACTION;
INSERT INTO People values ('777-77-7777', 'Ockun', 'Cameron');
INSERT INTO People values ('888-88-8888', 'Hamburger', 'Lindsay');
INSERT INTO People values ('999-99-9999', 'Werblow', 'Ashley');
COMMIT;
```

The above records will exist in the undo tablespace at least until the commit and probably well beyond. As you can see from this example, like indexes, undo space contains data that is repeated in the live table. In this case, it contains that data just before Oracle updates the live table with it.

Temporary Space

Oracle uses temporary space as working room to help it sort data. The most frequent sort operations come from table joins of DML statements and from *group by* and *order by* clauses of *select* statements. Let's go back to our *People* table for an illustration:

```
SELECT ssn, last_name, first_name
FROM People
ORDER BY last_name, first_name
```

Sorting on the above query happens on the *last_name* and *first_name* columns. If Oracle does not have enough room in memory for the sort operation, it will use one or more temporary tablespaces. Because of this behavior, the datafiles in such tablespaces can contain critically important information, in this case, a list of people's names.

Redo Logs

Redo logs contain a history of data changes. This includes data that may or may not have been written to the datafiles yet. A hacker can use them, for example, to restore the previous values on a record in an effort to cover his tracks. They contain raw data and should be protected in the same way as datafiles. Because of this, we include them here in the "Data" section rather than in "Logs" below. Figure 2.3 shows a query of the *v$logfile* system view that locates redo logs.

Figure 2.3 Listing Redo Logs

```
SQL> select member from v$logfile;

MEMBER
---------------------------------------
/data/oracle_home/oradata/ORDB/redo03.log
/data/oracle_home/oradata/ORDB/redo02.log
/data/oracle_home/oradata/ORDB/redo01.log
```

If you are an Oracle DBA, you will know that redo logs may be in one of several states (e.g., *online*). From an attacker's point of view, online ones will have the most recent information and are the ones to modify when erasing a history of illegal changes. From a defensive viewpoint, however, all of the redo logs need the same access restrictions and so they can be grouped as one line item on your lockdown plan.

Oracle archives redo logs to a location defined by the *LOG_ARCHIVE_DEST_1* parameter. Use the *show parameter* command to list parameters and their values. The files in the directory represented by that value should also be on your lockdown plan.

Backups

Backups of your data are a necessary part of an effective recovery strategy. They contain a complete copy of all of your data so one might think that they should be treated with roughly the same security consideration as live data. However, because they are not live there are two important differences: 1) their prioritization on your lockdown plan and 2) runtime operational requirements.

An unauthorized modification to a backup only becomes a problem if that backup is restored. Modifications to live data are higher risk from a security standpoint, because unauthorized modifications are more likely to have an adverse effect on your system. Files containing live data should be locked down before backups.

Data that will not be used actively does not have the same performance considerations as live data. For example, backups can be stored on slow media such as tapes. Considering security, backups are a better candidate for encryption than live data. Encryption and decryption of live data imposes performance overhead from the

CPU and possibly from other sources (e.g., out-of-process calls to user-defined functions). This typically requires faster CPUs and more of them, which turns the decision to encrypt live data into a business one. However, file-based encryption of backups is rarely a problem, because it can be done offline when time is not a factor. This is especially true for older backups that have a small chance of being restored. For recent backups in high-availability environments, the decision to encrypt backups should be made carefully, balancing security with recovery time. Lastly, in any situation where backups are being shipped from one location to another via an insecure channel (e.g., physical delivery from a data center to an off-site safe), encryption is a must. All too often tapes are "lost" in transit.

Control Files

Control files contain information about your data including the tablespaces, datafiles, and redo logs mentioned above. They don't technically contain data, but they are so critical to the proper operation of your database that you should protect them as data. To see a list of control files, view the *control_files* parameter as follows:

```
show parameter control_files
```

This will yield a result similar to the following:

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
control_files                        string      /export/home/oracle/oradata/OR
                                                 CL/control01.ctl, /export/home
                                                 /oracle/oradata/ORCL/control02
                                                 .ctl, /export/home/oracle/orad
                                                 ata/ORCL/control03.ctl
```

Logs

Log files contain messages related to the proper or improper functioning of Oracle. They describe what is happening in the database in such a way that is often interesting to someone preparing to mount an attack. The core Oracle process has an alert log file, which mostly contains startup/shutdown messages, errors, and other major events. It also has trace logs which indicate normal operation and detailed error event data. The listener has an analogous "listener.log" and its own traces. Oracle has one kind of trace and its listener has another, but both contain a log of what has happened. This is very useful to an attacker. With read access, the attacker can see what users are doing, and with write access, he or she can clear his or her own trace events. For example, trace logs can indicate failed login attempts. The attacker's ability to delete these impairs your ability to know that Oracle is under attack.

Oracle logs contain such basic information as the Oracle server version and operating system (OS) version. This may seem like information that is readily available and isn't that important to protect. However, the most secure environments will prevent access to any information that is unnecessary for an attacker to have. This abides by the security principle of "least privileges." In the case of Oracle, this includes banner information such as the version and patch level.

Trace files come from the background process, from user processes, and from the core process. Each type can be located in a different directory. These directories are nicknamed *bdump*, *udump*, and *cdump*, respectively. The commands to locate them and sample results from those commands are as follows:

```
show paramter background_dump_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
background_dump_dest                 string      /export/home/oracle/admin/ORCL
                                                 /bdump


show paramter user_dump_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
user_dump_dest                       string      /export/home/oracle/admin/ORCL
                                                 /udump


show paramter core_dump_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
core_dump_dest                       string      /export/home/oracle/admin/ORCL
                                                 /cdump
```

Notes from the Underground…

Reconnaissance Before an Attack

The first thing a hacker will do when attempting to attack a system is gather intelligence. Logs, for one, have more information than you think. Have you ever looked closely at what is in log files? From a hacker's perspective, they have a wealth of information. For example, the Oracle version and patch information reveals what vulnerabilities have been patched, or more to the point, have not been patched. If auditing is enabled, they have a history of what a user was doing and this is especially true if SQL auditing is on ("10046" traces). They include table names, column names, and most everything the user needs to start planning an attack.

Imagine that some service on the same host with Oracle runs as a user in Oracle's group. This happens a lot when, for example, a Web server needs access to the database and the Web site administrator doesn't want to store a username and password in a plaintext file. If that service has a vulnerability such that an attacker has access to the file system, he or she can browse around Oracle files such as datafiles and trace logs. Stealing data becomes easy, but there is enough information stored in files to develop a more permanent backdoor into the database.

The attacker can also look at redo logs to see the rate of incoming data changes to attempt to determine if the system is production or just a test machine. Don't forget that security is still important on test machines. Very often there is authentication information there that is reused in production.

Once the attacker has acquired enough information to start trying various attack vectors, he or she will probably attempt to use the *ksdwrt* procedure to try to hide his or her tracks if auditing is on. Imagine if an attacker were to create millions of lines in the trace that all have random data. It would be next to impossible to find the few lines indicating an attack. Be wary of traces that become too large. The attacker need only feed random data to a script containing the following command to hide his tracks:

```
execute sys.dbms_system.ksdwrt(1,'random data');Configuration
```

Oracle software uses several different kinds of files for configuration. They have information such as memory management settings. The wrong user having the ability to write to these is a DoS attack waiting to happen. Among these are parameter files (PFILEs and SPFILEs), "tab" files such as *oratab*, and "ora" files such as *listener.ora*. Generally speaking, look for files like **.ora*, **tab*, *spfile*.ora*, and *init*.ora*.

Software

Software files are the executables that define what Oracle does. If a user is able to write these, he or she can change Oracle's behavior in any way he or she wishes. To find these files after a clean install, review all of the files looking for execute permission as follows:

```
find /opt/oracle -perm -u=x
```

This will generate a large list of files, which happens to be a superset of the executables in an Oracle installation. For a more detailed list or if Oracle has already been installed, you'll need to run a tool to find files that match the executable format for your OS. Also, look for shell script files. On UNIX, these begin with the characters "#!"

Tools and Traps…

Finding ELF Files

On Linux and Solaris, the executable file format is the Executable and Linkable Format (ELF). A tool called *readelf* can help to identify such files so that you can add them to your lockdown plan. For example, here are the results of running the main Oracle executable through *readelf* on a 32-bit Linux box:

```
$ readelf -h oracle
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
```

Continued

```
  Version:                          0x1
  Entry point address:              0x82b4e28
  Start of program headers:         52 (bytes into file)
  Start of section headers:         88362132 (bytes into file)
  Flags:                            0x0
  Size of this header:              52 (bytes)
  Size of program headers:          32 (bytes)
  Number of program headers:        7
  Size of section headers:          40 (bytes)
  Number of section headers:        30
  Section header string table index: 27
```

If the file is not an executable, *readelf* generates the following output:

```
[ora10g@thor OPatch]$ readelf -h opatch
readelf: Error: Unable to read in 0x2323 bytes of section headers
readelf: Error: Not an ELF file - it has the wrong magic bytes at the start
```

To automate the process of finding ELF files, use the following script, which will print the input filename if it is a valid ELF file. First, save it as "isexec" and make it an executable file.

```
#!/bin/bash

readelf -h "$1" 2>&1 | grep EXEC > /dev/null
if [ $? -eq 0 ]
then
  echo $1
fi
```

Then execute it recursively over your Oracle installation by using the *find* command:

```
find . –exec isexec {} \;
```

The resulting list is all of the ELF files recursively in the directory in which you ran the command.

After generating a list of executables, your list of software should include *oracle*, *lsnrctl*, and a lot of other files from the *bin* directory and other subdirectories where Oracle was installed. You'll notice the list includes files ending in "O" such as *oracleO*. These are backup copies of executables, as they existed before an upgrade. We'll cover them in more depth in the next section. Also, add to your list three files from */usr/local/bin* if they exist: *dbhome*, *oraenv*, and *coraenv*. These are sometimes executed by users to set-up their environments and, if your DBA chose to keep them, you should ensure that they cannot be altered by a guest user.

Reviewing Recommended Permissions

An attacker's goal can be to extract information, to change it, or to escalate his privileges possibly to the point of completely controlling your database. Having full access to the files mentioned in this chapter allows him or her to do all of these things. There is nothing a hacker cannot do with read, write, and execute access to all directories, executables, data, and logs. If the attacker can log on to the local machine hosting your database, the only things stopping him are the permissions set on those files and the OSes enforcement of them. Executing a plan to revoke unnecessary permissions is a huge leap forward in locking down your system. While you are logged in as the user who owns Oracle files, an attacker's access to them is completely under your control.

Operating System Basics

The OS defines methods for securing access to files to only those users who have rights. It defines a concept of ownership by users and by groups of users. On UNIX systems (Linux, Solaris, and so forth), there are three ways to define ownership: the user who owns the file, a group of users, and "other" users (those users who are neither the owner nor in the group). On Microsoft Windows, there can be one or more users and groups. The OS also defines different kinds of privileges, each of which allow a user to perform a particular action with the file. Combining ownership with privileges yields a file's "permissions."

NOTE

OS administrators have complete access to the machine, including your Oracle files. There's nothing you can do about that. Someone has to administer the box. However, you can audit what he or she does. See the "Auditing" chapter for more details.

Each user and group (including "other" on UNIX) can have different privileges. The types of privileges they may have are "read," "write," and "execute." For example, the owner may be able to read and to execute a file, but no other user has access. In another example, the owner may be able to read and write to a file, but the group can only read it and no one else has access. Execute permission only applies to software and directories. For the latter, execute permission grants the ability to change the "current working directory" to be that directory.

The *ls* command on UNIX shows the permissions of one or more files and directories if given the "-l" parameter indicating long output. Those permissions are displayed in the form *xuuugggooo*, where *x* indicates a special file, "uuu" indicates the owning user's privileges, "ggg" indicates the owning group's privileges, and "ooo" indicates other users' privileges. For each, the three characters represent read, write, and execute privileges, respectively. Read is represented by an "r" if that privilege is enabled, write is represented by "w," and execute is represented by "x." Execute with "setuid" or "setgid" is indicated by "s" instead of "x" for the user and group, respectively. This means that no matter who runs the program, it will run as the user or group owning the file. It will have all of the access privileges of that user or group regardless of the user or group who executed it. If a particular privilege is not enabled, a dash "-" will appear in the *ls* output. Let's look at an example. The following is a long listing of the *oracle* binary and its backup file *oracleO*:

```
$ ls -l oracle oracleO
-rwx------ 1 orauser oragroup 69382160 May 31 10:14 oracle
---------- 1 orauser oragroup 69382160 May 31 10:14 oracleO
```

In the above example, *oracle* can be read, written, or executed by its owner. No other user has any permission. No user can do anything at all with *oracleO*; however, the owner *orauser* or a system administrator (the "root" user) can alter those permissions at any time.

Software Permissions

Executable files are the starting point of any application. When the operating system starts a process, it first loads the executable(s) files from disk into memory. Executable files include the "oracle" binary as well as any shared objects (*.*so*) on UNIX and Dynamic Link Libraries (DLL) (*.*dll*) on Windows. If these files have been hacked, they could do anything, for example, wipe the hard drive of anything owned by the "oracle" user (or whichever user started the process), copy credit card numbers to a rogue File Transfer Protocol (FTP) site, or grant Oracle *sysdba* privileges to a particular user account.

Your software files should not have any guest ("other") access if they do not use Oracle binaries locally. Be sure to check this, because some releases of 8i and 9i did not set this correctly by default. The combination of the setuid bit and "other" execute permission enabled a vulnerability which left Oracle log files open to attack: www.oracle.com/technology/deploy/security/pdf/oracle_race.pdf.

The owner should have full access to executables. Leave group access at read and execute for most of them, but for "oracle" and for any other programs, which your DBAs do not use directly, revoke all group access. You'll notice that some executables have the *setuid* or *setgid* bit set. Consider removing it, because it allows anyone with the execute privilege (e.g., a DBA) full access to Oracle, but only via that executable. If he or she is able to exploit a vulnerability such as the buffer overflow of ORACLE_HOME in some 8i and 9i releases of *otrcrep*, a hacker could cover his or her tracks by erasing log files, he or she could get access to files containing data to which he or she otherwise did not have access, and he or she could have access to Oracle's System Global Area (SGA) in shared memory. For even better security, do not allow DBAs to have local access to the OS except for tasks that absolutely require it, and be sure to audit such activity carefully. They can perform most tasks remotely via any Oracle client application.

WARNING

When a user runs Oracle software having the *setuid* bit set, it runs as the user who owns that software file, not the user who executed it. Oracle client software in this situation can directly access Oracle's SGA and thus communicate to the server via shared memory rather than via the network. Clients connecting via the listener on the other hand use the network, which is a bit slower. Removing the *setuid* and *setgid* bits requires that local connections be made via the listener rather than directly via shared memory, so consider performance and other implications before doing so.

The above permissions do not apply to backup copies of executables (i.e., those ending in "O"). They only exist in case the newer version has a problem and you need to revert to it. Because a backup is older than the live software, it may contain a vulnerability that was patched in the newer version. Subsequently, backups should have no permissions whatsoever. If you do need to revert, you can restore permissions at that time.

Non-software Permissions

Configuration files, data, logs, and everything else should have read and write permissions for the owning user and group. This enables a DBA to manage the system as necessary. Other than software, only directories should have execute permission. "Other" users generally should have no permissions unless they use Oracle client software locally on the Oracle server machine. If they do require local access, selectively grant read permission.

Managing Change

Verifying file permissions is an excellent first step in locking down the file system. However, this must be scheduled along with checking for new files, removing old ones, and verifying that existing ones have not changed. As you might imagine, in an active database system, many files are going to be added, removed, and changed on a regular basis. When creating your lockdown procedures for your file system, include not just a list of files and their permissions, but also a method that dynamically updates that list and a method to ensure that static components have not been modified. When you find new files or discover missing ones, verify that this is okay. For example, new datafiles and redo files may appear, or backups may have been copied to tape and then removed from disk.

Software and configuration files should change only when authorized, so this should be a relatively infrequent occurrence. A *cryptographic hash* is a large, unique number that acts as a fingerprint for a file. Calculate cryptographic hashes of these files during your first pass of locking down your database and then the next time you review your system, you can calculate them again to determine if anything has changed. Message Digest 5 (MD5) and Secure Hashing Algorithm 1 (SHA-1) are common hash algorithms, but in recent years flaws were found in both. SHA-256 and RIPEMD-160 are better options. Be sure to store your hashes offline in case the host is compromised.

It is important to assess file permissions on a regular basis to ensure they remain at their proper settings. This is especially true after an upgrade or even after applying a Critical Patch Update (CPU). These can alter your Oracle installation in any number of ways. It is possible for someone (e.g., an operating system administrator or a database administrator) to change permissions to be more permissive accidentally … or intentionally. In the section of your lockdown plan that deals with security tasks to be performed on a schedule, include these details.

Summary

The files that define an Oracle installation are an often overlooked target of attack. The threat of a non-database user gaining access to them is very real and all precautions should be taken to prevent this. Having read or write access to any one of these opens a host of opportunities to a hacker. Data files and transactions logs have your important data and their contents can be parsed with the right tools. Configuration files can tell Oracle to mask an attack, they can grant privileges not otherwise available, or they can completely disable your database. Properly altered binaries can tell Oracle to do anything an attacker wishes. Error and trace logs contain a wealth of knowledge and can be used as part of reconnaissance prior to an attack.

A proper lockdown strategy will include assessing your inventory of files, carefully granting and revoking permissions, and checking for changes. This includes taking snapshots of a complete listing of files, their permissions and checksums after they have been locked down, and then comparing them to the current runtime environment on a regular basis.

Solutions Fast Track

Getting to Know Your Files

☑ Oracle data is contained in datafiles, redo logs, and archives. All are of utmost importance to secure.

☑ While tablespaces that are used for indexes, undo information, and temporary space do not contain live tables, they do contain raw data and therefore must be secured.

☑ Log files contain information that exposes information about your environment to an attacker. They can be manipulated to mask activities.

☑ Restricting operating system privileges to modify executable files, including the Oracle server, its listener service, and others, helps to prevent hackers from modifying the behavior of the software.

☑ In any situation where backups are being shipped from one location to another via an insecure channel (e.g., physical delivery from a data center to an offsite safe), encryption is a must.

Reviewing Recommended Permissions

- ☑ No file should have guest access unless absolutely necessary. Never grant guess access to server executables.
- ☑ Revoke setuid bit on executables if local connections can be made via the listener.
- ☑ Revoke all access to files that are not in use. This include oracleO, tnslsnrO and lsnrctlO.

Managing Change

- ☑ Scan your executables for unauthorized changes by comparing cryptographic hashes periodically.
- ☑ Look for new and removed files and verify that it is valid for them to exist or to be missing. Update your lockdown plan accordingly.
- ☑ Look for permission changes on existing files and verify they were authorized

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: The Oracle installer sets the permissions I require for my environment. Why do I need to check them?

A: File permissions are not set in stone. A system administrator or DBA can accidentally change them. Even worse, an insider may set up a back door by giving himself additional permissions or by giving them to some server process having a known vulnerability. The insider need only exploit that vulnerability to gain access to the file system and because that process has access to Oracle files, so does the hacker.

Q: I do not have sufficient access to set permissions on files in */usr/local/bin*. How can I ensure the three files Oracle's installer places in that directory are secured?

A: On most UNIX systems, the administrative user "root" owns those files. Ask your local system administrator to change the permissions for you, or if you can login as root, you can do this on your own.

Q: Will any of the permission changes recommended in this chapter affect my run-time environment?

A: Yes, it's possible that revoking privileges on certain files will affect the way you use Oracle, the way Oracle performs, and the features available to you. Revoking all privileges on *extjob*, for example, will disable "external jobs," which are programs that run outside of Oracle directly on the OS. Please read this chapter carefully and consult an experienced DBA before moving forward with changes.

Q: I've implemented encryption as described in this chapter, but where do I store the private key?

A: Store the private key in a secure location separate from the encrypted data. Also, use a passphrase on the key so that using it without authorization is difficult.

Q: I've implemented encryption for particular data within datafiles, not encryption of the datafile itself. Do I still need to encrypt my backups?

A: That depends. Several different implementations exist to encrypt data on a row-by-row or column-by-column basis. If the keys to your data do not reside with your data and the unencrypted data is not important to secure, you may decide this is sufficient. If there is any important, unencrypted data or if the encryption keys are stored within the datafile such that a hacker might get to them, you will probably want to encrypt the file rather than individual pieces within it.

Q: You recommend that non-DBA users should not have access to execute Oracle software unless they run it locally. If I do have such users, for which software files should I enable execute permission?

A: The quick answer is to enable execute permission for non-DBA users only to those executables they require and no other. These should be client applications such as *sqlplus* and never applications that run as part of the server such as *oracle* and *extjob*.

Q: How often should I check for changes to my files?

A: The more frequent your checks are, the sooner you will be aware of security issues. However, if this is not automated, it will take a long time and the process will be prone to error. Also, consider performance issues related to the disk input/output (I/O) of scanning your file system. You will probably want to do this during off-hours.

Chapter 3

TNS Listener Security

Solutions in this chapter:

Introduction

The Transparent Network Substrate (TNS) Listener is an integral part of almost any Oracle database. TNS refers to the network protocol used by clients to connect to Oracle databases. The TNS Listener (referred to as "Listener" going forward) is essentially a proxy, listening for network connection attempts, checking to make sure the connection attempts match with the databases the Listener is handling, and then passing the communications on to the appropriate database or security identifier (SID). Listeners do more than just proxy database connections; they are also used to provide access to operating system (OS) executables via PL/Structured Query Language (SQL). This powerful feature allows the database to run OS commands and programs with the authority of the Listener, which generally runs as the Oracle account on *NIX systems or the local system on Windows. Because of its role in the process of connecting to the database and the capability to run programs and commands on the host OS, the Listener is a prime target for attackers. It's important that you take steps to secure every TNS Listener on your network.

Introduction to the TNS Listener

As long as Oracle has offered network connections to the database, they have done so via the Listener. Over time, as nearly all database interactions have become network based, the Listener has assumed a critical role in controlling access to Oracle databases. The Listener is actually a very simple piece of software, comprised of two executables and a set of configuration files. Despite its simplicity, the Listener possesses the power to control most access to the database, to run OS commands, launch programs, and if attacked and exploited, can grant the attacker complete control over the database and the server it runs on.

Listener security is all too often completely overlooked by Database Administrators (DBAs) and corporate security teams. The reasons for this are unclear. Perhaps it is because the Listener seems so simple, or because it is a component that runs outside of the database. Over the years, nearly two dozen vulnerabilities in the Listener have been discovered. These range from minor design flaws that allow anyone to obtain detailed information about the Listener and its associated databases, to Denial of Service (DoS) attacks that can stop the database from responding to connections, to buffer overflows that allow an attacker to run arbitrary code on the database server with the authority of the account that owns and runs the Listener, which unfortunately is almost always the same account that runs the Oracle database.

Oracle has long recognized the potential for devastating attacks against the Listener, and over time has taken steps to add security features that help protect it. Their efforts, however, were mostly unsuccessful, as the security features offered were disabled by default, and very few administrators took the time to learn about and enable them. To their credit, Oracle recognized the severity of the situation, and with the release of Oracle10g, the Listener was reconfigured to enable many critical security features by default. This has led to a dramatic improvement in Listener security for new installations, but the changes have not trickled down to the Oracle 9i and 8i Listeners that still make up the majority of Listeners at nearly all organizations.

Listener Components

The Oracle TNS Listener is made up of a pair of executables along with a set of three configuration files. The executables, *tnslsnr* and *lsnrctl* are generally located in the *$ORACLE_HOM/bin* directory, while the configuration files, *sqlnet.ora*, *listener.ora*, and *tnsnames.ora* are generally stored in the *$ORACLE_HOME/network/admin* folder. Together, these components provide the Listener with functionality, configuration information, and utilities to configure and control the Listener.

tnslsnr

This executable provides the core functionality of the Listener. It is essentially a server process that proxies network connections to Oracle databases and provides those databases with the capability to run external procedures via PL/SQL calls. This executable is the foundation component that allows incoming network connections to reach Oracle databases.

lsnrctl

This executable is the utility used to configure and control the Listener. It provides the capability to administer a Listener either on the local machine or remotely across the network. This Listener control program facilitates running Listener commands to configure logging and tracing, set passwords, show information about the Listener and the databases it is serving, and even shut down the Listener process.

sqlnet.ora

This configuration file is used to provide the Listener with various networking parameters. Most relevant to security are settings to enable encryption and integrity checking of network communications, configure authentication modes, and to provide a list of hosts that are (or are not) allowed access to the database. Most of the security features enabled in *sqlnet.ora* require the Advanced Security Option (ASO) in order to function. One exception to this is Valid Node Checking, which is a powerful security feature that allows an administrator to specify a list of either Invited or Excluded nodes, controlling which hostnames or Internet Protocol (IP) addresses are allowed to, or are restricted from connecting to the database. In some systems, this can be used to help ensure that only legitimate host systems connect to the database, thwarting unauthorized attempts to connect to the system even before authentication begins.

listener.ora

This configuration file identifies the Listener and the databases for which it is listening. Included in *listener.ora* is a unique name to identify the Listener, a list of services (or SIDs) it is listening for, and a list of IP addresses and ports that it should listen on and accept communications from. *listener.ora* also provides other important configuration information, such as the Listener password(s), logging and tracing levels, and log and trace file locations. Essentially all of the information about how the listener should run is stored in this file. Changes to the Listener configuration can be made by directly editing *listener.ora* or by running commands using *lsnrctl*. Remember to restart the Listener after making any changes, as it only reads its configuration file once, at startup.

WARNING

listener.ora is a critical file to protect, particularly from write access. If a user can write to this file, they can change the configuration of the Listener in any way they choose. Writing to *listener.ora* is the most powerful method of administering the Listener; there are no controls in place to limit the changes that can be made. Critical changes that can be made to the Listener via *listener.ora* include setting a new Listener password, disabling logging and tracing, or disabling it entirely.

Consider protecting *listener.ora* with owner only file permissions (0x700 on *nix systems). If this is not an option, allow the group and everyone else read permissions (0x711 on *nix systems). File permissions are discussed in detail in Chapter 2.

tnsnames.ora

This configuration file is used to map Oracle Connect Descriptors to Net Service Names. The idea here is to simplify connections to databases by allowing the use of a short, intuitive Net Service Name instead of the longer and more convoluted Connect Descriptors. There is little in *tnsnames.ora* that is relevant to security; however, in this file you can specify information about the Secure Sockets Layer (SSL) certificates you want to use for authentication (assuming you have SSL enabled on the database). SSL features require the ASO.

Listener Commands

Oracle TNS Listeners have their own set of commands used for querying and making configuration changes. The Listener Control utility (*lsnrctl*) is the tool Oracle provides for managing the Listener. *lsnrctl* is most often used to manage Listeners on the local host, however, it can easily be configured to remotely connect and administer any accessible Listeners on the network. Oracle provides detailed documentation on the Listener, including in-depth descriptions of each supported command; however, the protocol used to communicate these commands is Oracle proprietary and undocumented—at least it was undocumented before the hacker community reverse-engineered and documented it themselves in April 2001. With the posting of the command formats and protocol information, the ability to manipulate the command language and cause all sorts of havoc began.

Notes from the Underground…

Hacking the Listener using *tnscmd*

When jwa, a self described lazy, pale computer geek posted his *tnscmd* script and documentation on his Web site in April 2001, a Pandora's box was opened. While he posted only a small amount of security research and information about attacks, it did not take long for the hacker community to take the lead from his work, and find ways to manipulate Oracle TNS Listeners to do their bidding. Today, more than six years after jwa's posting, the same misconfigurations exist in most production Listeners that allow hackers to take control of Oracle databases and the servers they run on.

Continued

There are a few types of attacks that the *tnscmd* script and associated documentation helped make possible, including DoS, running SQL queries as a DBA, leaking sensitive information, and most disturbingly, granting rlogin access to the host OS. You can still find *tnscmd* and its documentation at: www.jammed.com/~jwa/hacks/security/tnscmd

The simplest attacks are the DoS attacks. These come in at least three flavors, including stopping the Listener which effectively blocks any new network connections to the database, setting the Listener trace level to "Support," which causes a whole lot of logging on a busy system and can severely degrade performance, and overwriting critical files that can cause a system to become unstable and crash or can delete important data.

The Listener offers three commands that are designed to provide information about the Listener itself and the databases it is listening for. These commands, *version*, *status*, and *services*, provide details on both the Listener and the database SIDs. Some of this information should be considered sensitive, as it gives insight into the configuration of the system. This is hacker gold. What can be easier than a system that identifies itself by version number, tells you what OS it is running, defines its important environment variables for you, then gives you the path to its log and configuration files?

The last two classes of attacks running SQL queries as a DBA and granting rlogin access, both work using the same principal, and both can be executed with devastating results. By instructing the Listener to change its log file location and name, an attacker can input arbitrary data into the log. This can be used to create a *.rhosts* file granting rlogin access to the OS. It can be used to create a *glogin.sql* file, to be run every time *sqlplus* is used to connect to the database locally. That may not happen every day, but when it does, it's almost always with a DBA account. This would allow an attacker to create their own DBA account in the database, unlock a powerful default account, or even corrupt or delete critical data. These attacks are simple. Change the log file name and location and then send bogus commands to the Listener. It will write the failed commands right into the log file specified. So, create a *glogin.sql* file and then send SQL statements to the Listener wrapped up as Listener commands (using *tnscmd*), and the Listener will write those SQL statements right into *glogin.sql* for you (along with some other lines of logging information, which will essentially be ignored by the database).

You can avoid all of these attacks with proper Listener configuration. This chapter will guide you through configuring your Listener securely.

Oracle 10g Listener Changes

Oracle made some important security-related changes to the Listener with database version 10g. Most importantly, Listener security is enabled by default but no password has been set. Instead, Oracle has added the capability to use Local OS authentication, meaning that only the owner of the Listener executable can authenticate and therefore run commands to administer the Listener. This means no more remote administration of the Listener whatsoever when using the default configuration, and also means that local OS users who don't own *lsnrctl* can't configure the Listener either, unless they have permissions to write to *listener.ora* directly, but even then, their changes will not take effect until the valid user restarts the Listener.

Presumably, for reasons of backwards compatibility, all of this good security can be disabled. So even with 10g Listeners, it is important to periodically review their configuration and make sure they are properly locked down. That being said, the default configuration of a 10g Listener is not all it could or should be. At a minimum, there are a few changes that should be made to the default configuration to enhance the protections provided to the Listener. Fortunately these steps are the same on 9i Listeners, so while in the end 9i and 10g Listeners will end up with slightly different behavior (because of the local OS authentication feature only offered in 10g), the methodology to secure the various Listeners is largely the same.

Listeners Can Be a Major Source of Vulnerability to Attacks

Over the years, security researchers and the black hat hacker community have uncovered an array of attack vectors by which an Oracle TNS Listener can be used to either block access to a database, steal data from a database, or give the attacker complete control over the database and the host OS it runs on. The types of vulnerabilities found can be broken down into two broad categories, those that result from coding mistakes made by Oracle developers, and those that result from insecure configurations of the Listener. The coding mistakes get fixed by applying patches. There are currently no publicly known code-related vulnerabilities in the Listener for which Oracle has not released a patch, so fixing this entire category of issues can be handled all at once by simply applying the latest set of patches from Oracle. We'll cover patching in detail in Chapter 7. The vulnerabilities that result from insecure Listener configurations are all avoidable, and are in your hands to correct. The majority of this chapter will focus on how to configure the Listener securely.

Listener Vulnerabilities "By Design"

Oracle TNS Listeners have a few major security flaws in their authentication system. None are coding mistakes or misconfigurations; they are there on purpose, by design. Every once in a while Oracle throws us a real curveball, and this is one of them.

No Account Lockout

Oracle has elected not to implement any type of account lockout features for the Listener, even though these features have existed for years in the database. Account lockout is a critical security feature, designed to prevent brute-force password guessing. After a defined number of failed attempts to login with a given account and an incorrect password, a system with account lockout enabled will disable the account. This prevents an attacker from guessing every word in the dictionary or every combination of type-able characters until they hit the right password and get logged into the system. The Listener has no such feature. This opens the door to brute-force password guessing attempts that if given enough time and run aggressively will guess any password regardless of complexity.

This behavior is by design, and there is really no way around it. The Listener can log failed login attempts, so reviewing those logs periodically can be helpful, although it becomes a significant task if you are responsible for more than just a few databases. Oracle10g Listeners have the option of using local OS authentication only, which eliminates the password guessing problem, but also forces you to do all Listener administration locally on the database server.

Passwords Transmitted in Cleartext

This one was a real shocker. Who in their right mind sends a password across a network in cleartext? Oracle has been protecting database credentials sent across the network since they first allowed network connections to the database, so they clearly have the technology to do this. The same algorithm used by the database could certainly have been implemented in the Listener, but for reasons unknown it has not been. This means that if you administer your Listeners remotely, you may want to change the way you work. Anyone with the capability to "listen" to your network communications (or those of the database) can easily steal the Listener password and then use it to wreak all sorts of havoc. One good way to mitigate the risk here is to force the Listener to only accept encrypted communications by setting the protocol to Transmission Control Protocol (TCP)/IP with SSL in your *listener.ora* file. Use the following format:

```
ADDRESS = (PROTOCOL=TCPS)(HOST=hostname)(PORT=port))
```

Authentication with Password or Password Hash

This is another case where, as a security professional, I am totally taken aback by Oracle's implementation. Up until Oracle10g, it was possible to authenticate to the Listener by providing either the correct Listener password or by providing the Listener password hash. This is the kind of thing that must have been done on purpose, but at least in this case Oracle has realized the error of their ways and made a change to eliminate the problem in 10g. Allowing authentication with a password hash instead of the true password is a real problem. Remember, password hashes are used to protect the real passwords, but if you accept the hash instead of the password, then any security provided by storing the password as a hash is defeated. Furthermore, the Listener password hash is stored in the *listener.ora* file. Anyone with read permissions on this file can get the password hash, use it to authenticate to the Listener, and then run commands to change its configuration.

Fixing Listener Vulnerabilities by Applying Oracle Patch Sets and CPUs

Oracle released their first Listener security fix way back in November 2000 in Security Alert #2, a patch to help block attackers from setting the log file and trace file to any arbitrary OS file. Since that time, Oracle has released fixes for at least 16 different Listener vulnerabilities, and you can expect that there will be more fixes released over time. The vast majority of the issues Oracle has patched have been DoS attack susceptibility; however, at least a few of the patches address more critical buffer overflows that could allow an unauthenticated attacker to run arbitrary code on the database server. Consider that a DoS attack against a Listener is in many ways a DoS attack against the database, as killing the Listener makes it impossible for new network connections to be established to the database. These attacks may not give over control of your system, nor will they leak your sensitive data; however, in many cases stopping business from operating is just as bad. Consider an Oracle database that is hosting an eBusiness application. If the database stops, so does the application and consequently people can't purchase products. That can't be good for business. How about a database hosting a stock trading system at a major financial institution? If the trading database goes down, trades cannot be processed and millions of dollars could easily be lost in an instant. DoS attacks are not to be taken lightly.

Listener DoS Attacks

There are some common trends among the types of DoS vulnerabilities that have been found in the Listener over time. Most are caused by mishandling of unexpected input, where the Oracle developer made the (foolish) assumption that the network protocol would always behave as specified, or that only legitimate commands would be sent to the Listener, or that only valid client version information would be sent to the Listener. This is the folly of many developers, trusting the users and systems to behave as expected, and then only thoroughly testing the expected input cases. Making software work is quite different from making it secure. Those differences begin at the design and development phases and continue to grow into the testing cycle. Oracle is starting to learn that functional testing is not enough, and that security testing is just as important. It is often more costly to resolve software security issues than it is to fix functional bugs, thus the "secure-by-design" concept is becoming popular in the software development world.

TIP

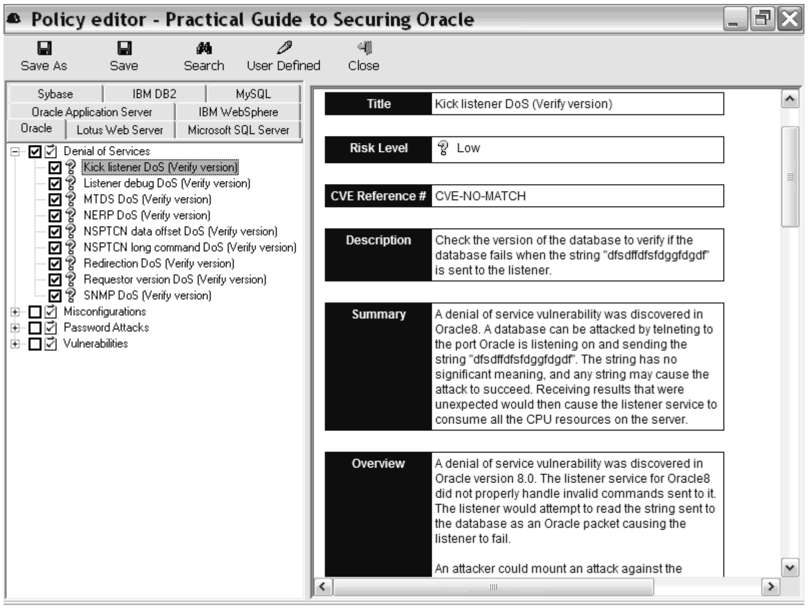
Not every test plan was created equal. There are many categories of testing that apply to software systems: functional testing, performance testing, stress testing, failure case testing, security/penetration testing, and more. A comprehensive test plan takes into account all of the types of tests that can be run, and include all those that are applicable to the application being tested. Some software is actually tested this way, particularly the critical software components that do such things as control airplanes or nuclear generators.

Most software applications, however, never get a comprehensive test plan. Often it is too time consuming, or too expensive, or never even considered. Instead, many applications are tested for success cases only, verifying that the software will behave as expected as long as both the user and environment behave as expected. What happens when the software expects a user to set a value to "TRUE" or "FALSE" and they set it to "FOOBAR" instead? What happens when the software expects to receive a password and gets SQL code instead? What happens when a user stops a transaction midway through? Will the system catch the error and fail gracefully? Or will it fall flat on its face? Without both strong success and failure case testing,

applications are likely to fail in unfortunate ways when exposed to unexpected or improper input. Incomplete testing of applications is one of the main reasons why hackers can remain in business.

Figure 3.1 below shows the Oracle DoS vulnerabilities that can be detected by Application Security, Inc.'s AppDetective database vulnerability scanner. Note the vulnerability that is selected, the "Kick listener DoS." Take a look at the description and summary to get an idea of how easy it is to exploit a vulnerability like this. At the same time, think about testing for failure cases and unexpected input. In this case, it is clear that Oracle never tested the scenario in which a Listener is sent a random string over a Telnet connection. This is a simple problem that should have been caught before release; however, it highlights the variety of vectors from which an attack can originate. Even those developers and Quality Assessment (QA) engineers that take steps to perform thorough testing of failure cases and unexpected input can miss something. There are simply too many situations to test to ever perform an exhaustive analysis. Staying on top of patches that are released by software vendors is the best way to ensure that your systems are protected against vulnerabilities that are discovered over time after the initial software release.

Figure 3.1 AppDetective Database Vulnerability Scanner Showing Checks for Listener DoS Vulnerabilities



- Denial of Services
  - Kick listener DoS (Verify version)
  - Listener debug DoS (Verify version)
  - MTDS DoS (Verify version)
  - NERP DoS (Verify version)
  - NSPTCN data offset DoS (Verify version)
  - NSPTCN long command DoS (Verify version)
  - Redirection DoS (Verify version)
  - Requestor version DoS (Verify version)
  - SNMP DoS (Verify version)
- Misconfigurations
- Password Attacks
- Vulnerabilities

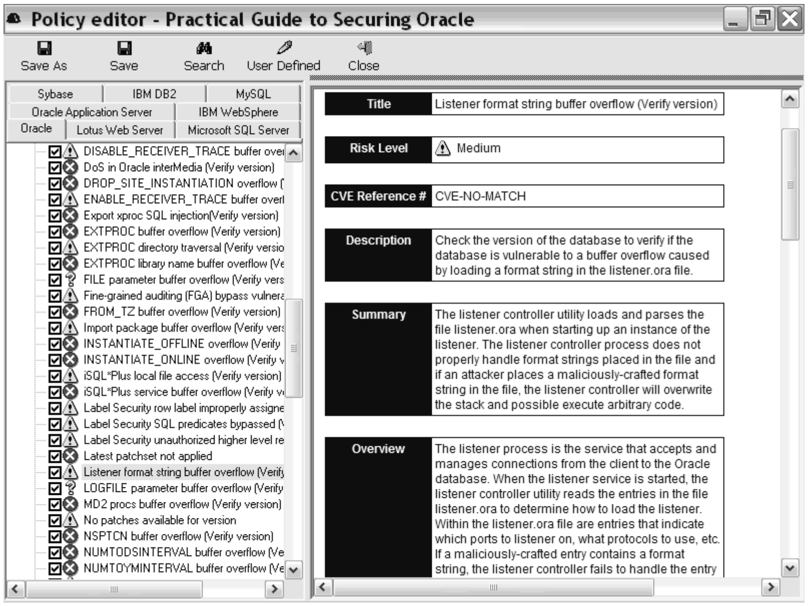
Title	Kick listener DoS (Verify version)
Risk Level	Low
CVE Reference #	CVE-NO-MATCH
Description	Check the version of the database to verify if the database fails when the string "dfsdffdfsfdggfdgdf" is sent to the listener.
Summary	A denial of service vulnerability was discovered in Oracle8. A database can be attacked by telneting to the port Oracle is listening on and sending the string "dfsdffdfsfdggfdgdf". The string has no significant meaning, and any string may cause the attack to succeed. Receiving results that were unexpected would then cause the listener service to consume all the CPU resources on the server.
Overview	A denial of service vulnerability was discovered in Oracle version 8.0. The listener service for Oracle8 did not properly handle invalid commands sent to it. The listener would attempt to read the string sent to the database as an Oracle packet causing the listener to fail. An attacker could mount an attack against the

Listener Buffer Overflow Attacks

There have been only a small number of Listener buffer overflows that Oracle has fixed, and of those, little is known about them. What we do know is that in some cases, an unauthenticated user can issue a buffer overflow attack that allows the attacker to run arbitrary code on the server. This is the most dangerous kind of buffer overflow, as anyone who can communicate with the Listener across the network can initiate the attack.

The Listener Format String Buffer Overflow is one that is fairly well known. The attack involves placing a malformed format string in the *listener.ora* file. When the Listener processes this file, it fails to deal properly with the format string, resulting in a stack-based buffer overflow. When crafted properly, this can be used to run arbitrary code on the system as the owner of the *tnslsnr* process, generally exposing all the files and data in the database to the attacker. This attack is limited in scope—the buffer overflow only occurs upon a restart of the Listener. In order to place the format string into the *listener.ora* file, the attacker needs local OS access with an account that can write directly to *listener.ora*, or the attacker needs to be able to remotely issue Listener commands. Figure 3.2 below shows some of the Oracle buffer overflow vulnerabilities that can be detected by AppDetective.

Figure 3.2 AppDetective Database Vulnerability Scanner Showing Checks for Various Oracle Buffer Overflow Vulnerabilities



- DISABLE_RECEIVER_TRACE buffer ove
- DoS in Oracle interMedia (Verify version)
- DROP_SITE_INSTANTIATION overflow (
- ENABLE_RECEIVER_TRACE buffer overl
- Export xproc SQL injection(Verify version)
- EXTPROC buffer overflow (Verify version)
- EXTPROC directory traversal (Verify versio
- EXTPROC library name buffer overflow (Ve
- FILE parameter buffer overflow (Verify vers
- Fine-grained auditing (FGA) bypass vulnera
- FROM_TZ buffer overflow (Verify version)
- Import package buffer overflow (Verify vers
- INSTANTIATE_OFFLINE overflow (Verify
- INSTANTIATE_ONLINE overflow (Verify v
- iSQL*Plus local file access (Verify version)
- iSQL*Plus service buffer overflow (Verify v
- Label Security row label improperly assigne
- Label Security SQL predicates bypassed (V
- Label Security unauthorized higher level re
- Latest patchset not applied
- Listener format string buffer overflow (Verify
- LOGFILE parameter buffer overflow (Verify
- MD2 procs buffer overflow (Verify version)
- No patches available for version
- NSPTCN buffer overflow (Verify version)
- NUMTODSINTERVAL buffer overflow (Ve
- NUMTOYMINTERVAL buffer overflow (Ve

Title	Listener format string buffer overflow (Verify version)
Risk Level	⚠ Medium
CVE Reference #	CVE-NO-MATCH
Description	Check the version of the database to verify if the database is vulnerable to a buffer overflow caused by loading a format string in the listener.ora file.
Summary	The listener controller utility loads and parses the file listener.ora when starting up an instance of the listener. The listener controller process does not properly handle format strings placed in the file and if an attacker places a maliciously-crafted format string in the file, the listener controller will overwrite the stack and possible execute arbitrary code.
Overview	The listener process is the service that accepts and manages connections from the client to the Oracle database. When the listener service is started, the listener controller utility reads the entries in the file listener.ora to determine how to load the listener. Within the listener.ora file are entries that indicate which ports to listener on, what protocols to use, etc. If a maliciously-crafted entry contains a format string, the listener controller fails to handle the entry

Securing the Listener Configuration

There are a few key configuration settings in the Oracle Listener that, when misconfigured, can introduce significant vulnerabilities. It is important to understand the meaning of each of these settings, and then configure them correctly for your environment. Luckily, things should be somewhat consistent for almost everyone and, in general, there is no reason not to implement a secure configuration. A locked-down Listener will look different in 10g than it will in previous versions, but the differences are minor. It is important to note that a 10g Listener can be implemented with a more secure configuration than any previous Oracle Listener, so upgrade your Listeners to 10g wherever possible, even if the database must remain at a previous version.

Listener Security/Listener Password

In versions earlier than 10g, enabling Listener Security meant setting a Listener Password. Listener Security sounds like more of a feature than it really is. When it is turned off, anyone can connect to the Listener and run commands without any type of authentication. When it is turned on, the majority of Listener commands cannot be run without first authenticating with the set password statement:

```
LSNRCTL> SET PASSWORD
Password: mypassword
```

Far too many Oracle Listeners have Listener Security turned off. To help eliminate this problem, Oracle made a change to the Listener in 10g, allowing Listener Security to be turned on without having to set a Listener Password. The idea is to use the local OS for authentication, allowing only the owner of the *tnslsnr* process to run Listener commands. This effectively disables remote administration of the Listener, which is quite good for security. This doesn't mean that you shouldn't configure a Listener Password for a 10g Listener; it means you have a choice. If you do decide to set a Listener Password in order to facilitate remote Listener queries, be sure to turn on ADMIN_RESTRICTIONS.
You can also set one or more Listener passwords by placing them directly in the *listener.ora* file as follows:

```
PASSWORD_[listener name] = password[,password2]
```

Substitute the actual name of the Listener from the *listener.ora* file for *[listener name]* above.

WARNING

The database security research community has been quietly hinting at vulnerabilities in the 10g Listener that may allow a remote attacker to bypass the local OS authentication and connect to the Listener to run commands. If they are able to bypass that authentication, there is nothing to stop them from administering the Listener remotely. In order to mitigate the risk, we strongly recommend that you always set a Listener Password and enable *ADMIN_RESTRICTIONS*.

ADMIN_RESTRICTIONS

The *ADMIN_RESTRICTIONS* flag was added to the Listener as a security patch in November 2000, intended as a means to block attackers from setting the *TRC_FILE* and *LOG_FILE* to any file location with any extension. Oracle was just learning about security patching back then; in fact, this issue was fixed in Oracle Security Alert #2. Because they were so new to security patching, it's no surprise that Oracle took a naïve approach to patching this issue. Instead of going into the Listener code and making changes to limit the locations and file extensions allowed for log and trace files, they added the *ADMIN_RESTRICTION* configuration flag. When enabled, *ADMIN_RESTRICTIONS* instructs the Listener not to accept any administrative commands from *lsnrctl*. Instead, an administrator must log in to the Listener's host OS and make configuration changes directly in *listener.ora*.

Specifically, the *ADMIN_RESTRICTIONS* flag tells the Listener to block any *SET* commands. These commands include *SET TRC_FILE*, *SET TRC_DIRECTORY*, *SET TRC_LEVEL*, *SET PASSWORD*, *SET LOG_FILE*, *SET LOG_DIRECTORY*, *SET LOG_STATUS*, *SET CURRENT_LISTENER*, and *SET STARTUP_WAITTIME*. By blocking these commands (along with the other SET commands), it becomes impossible to remotely make changes to the Listener configuration using *lsnrctl*. The problem with this approach is that administrators needed to not only download and apply the patch; they needed to reconfigure all of their Listeners, and learn how to manipulate the Listener by writing to *listener.ora*. This is where things basically fell apart. Many systems had the patch installed but never enabled *ADMIN_RESTRICTIONS*. Oracle's approach may have done as much harm as good, giving administrators a false sense of security because they applied the patch, while not making any real impact because the security feature was never enabled.

Enabling *ADMIN_RESTRICTIONS* is the right thing to do for every one of your Oracle Listeners. Doing so won't break your applications and won't change the way you work with the database. Making configuration changes in *listener.ora* is easy and well documented. This simple step will effectively shut the door on many Listener attacks, while maintaining your ability to remotely query the Listener for status, services, and version information.

Enable *ADMIN_RESTRICTIONS* by adding the following line to your *listener.ora* file:

```
ADMIN_RESTRICTIONS_[listener name] = ON
```

Remember to substitute the actual listener name in the *listener.ora* file for *[listener name]* above.

Listener Logging and Tracing

There are two types of data collection performed by the Listener: logging and tracing. During normal database operation, you always want logging enabled and tracing disabled. The two functions are related, but are very different. Listener logging provides an audit trail of the basic operation and usage of the Listener. It produces a fairly compact sized file and contains a listing of the Listener commands that have been run, the hosts that ran the commands, and the protocols they used to connect to the Listener. It also lists any failed commands, including invalid passwords and attempts to run commands that are blocked due to *ADMIN_RESTRICTIONS* being enabled. The Listener log is important; it contains information that could identify an attempt to break into your database (both failed and successful attempts). It is not enough to simply enable Listener logging and then forget about it. Take the time to periodically review the logs, or write a little code to help you review them. Arup Nanda wrote a great paper on Mining Data from the Listener Log. Download it at: www.dbazine.com/oracle/or-articles/nanda14/.

Enable Listener Logging with the following commands in *lsnrctl*:

```
LSNRCTL> SET LOG_DIRECTORY $ORACLE_HOME/network/log (*nix systems)
LSNRCTL> SET LOG_DIRECTORY %ORACLE_HOME%\network\log (windows systems)
Connecting to (ADDRESS=(PROTOCOL=TCP)(HOST=mothership)(PORT=1521))
LISTENER parameter "log_directory" set to $ORACLE_HOME/network/log
The command completed successfully
```

```
LSNRCTL> SET LOG_FILE listener.log
Connecting to (ADDRESS=(PROTOCOL=TCP)(HOST=mothership)(PORT=1521))
LISTENER parameter "log_file" set to listener.log
The command completed successfully

LSNRCTL> SET LOG_STATUS = on
Connecting to (ADDRESS=(PROTOCOL=TCP)(HOST=mothership)(PORT=1521))
LISTENER parameter "log_status" set to ON
The command completed successfully
```

If you have ADMIN_*RESTRICTIONS* enabled (and I hope you do), enable Listener logging by adding these lines to *listener.ora*:

```
LOG_DIRECTORY = $ORACLE_HOME/network/log (*nix systems)
LOG_DIRECTORY = %ORACLE_HOME%\network\log (windows systems)
LOG_FILE = listener.log
LOG_STATUS = ON
```

The *LOG_DIRECTORY* and *LOG_FILE* parameters are optional. You can set them to any directory or file name you like. If you omit the parameters, they will default to the values shown above for *LOG_DIRECTORY*. *LOG_FILE* defaults to *[listener name].log*, where *[listener name]* is the actual name of the Listener from *listener.ora*.

Listener tracing is designed to provide detailed step-by-step information on what the Listener is doing. Tracing produces a tremendous amount of data, and is intended for use in debugging database connection issues. There is little forensic value in the data collected by Listener tracing and the space requirements to store the trace would be enormous in busy systems. In fact, there is a DoS attack against the Listener that works by setting the tracing level to SUPPORT, which logs so much data that it can use up central processing unit (CPU) and HDD resources, causing the database to stop responding.

To ensure Listener tracing is disabled on your system, run the following command in *lsnrctl*:

```
LSNRCTL> SET TRC_LEVEL = off
```

which results in:

```
Connecting to (ADDRESS=(PROTOCOL=TCP)(HOST=mothership)(PORT=1521))
LISTENER parameter "trc_level" set to off
The command completed successfully
```

Alternately, turn tracing off in *listener.ora* as follows:

```
TRC_LEVEL = off
```

TIP

Listener tracing is a facility provided to debug network connection issues to databases and problems running ExtProc. The trace can be configured to run at varying levels of verbosity, depending on how much detail you want to see. Often, a single entry in a Listener log file can result in hundreds of lines of trace data, so it is not something you want on all the time, but it can give you the detailed information you need to fix various connection problems. When working with Oracle support to resolve a connection issue, it is not unusual for them to ask you to enable Listener Tracing at the "Support" level and send them the trace file for analysis. Just make sure to turn tracing off again once your problem has been resolved.

ExtProc

External Procedure Servers provide Oracle databases with a means of executing functions outside the database through a PL/SQL interface named the "call specification" (or call spec). TNS Listeners are the component used to implement an External Procedure Server, or ExtProc as it is commonly called. ExtProc allows the database to run functions written in C, Java, or any other programming language. This expands the functionality of the database tremendously and is a great feature when it's implemented properly.

When the database calls an external procedure, an alert is sent to the Listener, which then starts an external procedure agent. This agent is installed by default and named *extproc*. The database sends the Listener information about the external procedure call, then the Listener makes the call, collects the results, and ships them back to the database. The functions run by the Listener are run as the OS account that owns the Listener process (oracle software owner account by default).

ExtProc has significant security ramifications. It's a powerful feature, and in the wrong hands, can make it easy to assume control over the database and the host OS. For most databases, ExtProc is unused and therefore unnecessary. Review your system; if it is not using external procedures, remove ExtProc entirely. If you find that you are using ExtProc, there are a few steps you should take to secure your implementation.

First, you will need a plan. That means figuring out exactly how ExtProc is used in each of your databases, which libraries and dynamic link libraries (DLLs) are called, where they are located, and the OS permissions needed to run them. There is no easy way to figure this all out, but the Listener will help you by logging any ExtProc calls from the database. Check the Listener log for any ExtProc entries, and note the details of which libraries were called. Once you have gathered the necessary information, it's time to set up a second Listener, this one exclusively for running ExtProc. This second Listener should run as an OS user created, again, just for running ExtProc, with only the permissions needed (based on the information you collected for your plan). Configure the Listener to limit which libraries or DLLs can be called using an *ENVS* statement in the *listener.ora* file. Your new *listener.ora* file should start something like this (this Listener is named *LISTENEREXTPROC*):

```
 LISTENEREXTPROC=
 (DESCRIPTION=
  (ADDRESS=
     (PROTOCOL=ipc)(KEY=extproc)))
SID_LIST_LISTENEREXTPROC=
  (SID_LIST=
    (SID_DESC=
     (PROGRAM=extproc)
     (ENVS="EXTPROC_DLLS=ONLY:/home/xyz/mylib.so:/home/abc/urlib.so,
LD_LIBRARY_PATH=/private/xpm/lib:/private/mylibs,
MYPATH=/usr/ucb:/usr/local/packages,APL_ENV_FILE=/apl/conf/env.txt")
     (SID_NAME=extproc)
     (ORACLE_HOME=/oracle)))
```

The *ENVS* statement defines the *ONLY* files that the Listener will pass ExtProc calls. This limits the Listener to running only the external procedures used by your system. Remember, the filenames and paths above are just an example, you should replace them with the proper paths and files for your system. Note that when an *ONLY* statement is included, you must provide the full path to each file.

NOTE

It's critical that you never configure *listener.ora* such that *ENVS="EXTPROC_DLLS=ANY".* Doing so disables DLL checking and essentially tells the Listener to run any DLL that the database requests. This opens the door to an attacker assuming complete control over the OS. Even if you have properly configured the Listener to run as a low-privileged user, allowing any DLL to be called is a huge risk. Avoid it at all costs.

Once the new Listener is set up and running, you should remove the ExtProc entries from the Listener that handles connections to the database. To do this, you will need to make some changes to *listener.ora* to remove the ExtProc related entries. Find and remove any *ADDRESS_LIST* entry with *KEY = EXTPROC* and any *SID_DESC* entry where *SID_NAME = PLSExtProc* (or similar) or where *PROGRAM=extproc*. Make sure to save your changes and then restart the Listener.

Tools & Traps…

Default Listener Configuration

Far too many Oracle DBAs fall into the default configuration trap. It's an easy way to go, it just works, and it's all configured for you. The problem is, the default Listener configuration, particularly before the release of 10g is wide open, just asking to be attacked. Here are the primary areas of concern with respect to the default Listener configuration:

Oracle9i and older:

- Listener Security is off by default. There is no default Listener password. Anyone who can connect to the Listener across the network can administer it.
- Administrative Restrictions are off by default. This means that any *SET* command can be run remotely by sending commands to the Listener (either by *lsnrctl* or any other utility such as *tnscmd*).
- Port 1521 is used. This is the default listener port and is well known. Using the default port makes it easy for an attacker to scan the network and find your database. By switching to a non-default port, such as 15021, you can make it more difficult for someone to

Continued

scan your network and find the database. Avoid the port range 1520–1530, as these are all well known. Most commonly Oracle listens on port 1521 or 1526.

- ExtProc is enabled without any restrictions on which DLLs can be called. This means that anyone with database access, legitimate or not, can gain access to the *call spec* and run OS commands as the Oracle software owner account.
- The Listener process is owned by oracle on *nix systems and Local System on Windows. This account is more powerful than is needed to run the Listener. This becomes a significant issue if ExtProc is enabled. Even without ExtProc, the existence of buffer overflow vulnerabilities in the Listener could allow for arbitrary code execution. If the Listener is running with a powerful OS account, the damage that can be done by such an attack is vastly increased.

Oracle10g:

- Listener security is on by default, but there is no Listener password set. Instead, the configuration defaults to Local OS Authentication. This is actually good for security; however, there have been issues reported to Oracle that allow an attacker to bypass the Local OS Authentication. In order to mitigate the risk of such an attack, a strong Listener password should be set.
- Administrative Restrictions are off by default. This means the *SET* command can be used to change Listener configuration settings. Remote administration is supposed to be disabled because of the Local OS Authentication model, but as was previously mentioned, this model is not unbreakable. It is important to set *ADMIN_RESTRICTIONS = ON* for any Oracle Listener.
- Port 1521 is used. This is the default listener port and is well known. Using the default port makes it easy for an attacker to scan the network and find your database. By switching to a non-default port, such as 15021, you can make it more difficult for someone to scan your network and find the database. Avoid the port range 1520–1530, as these are all well known. Most commonly Oracle listens on port 1521 or 1526.
- ExtProc is enabled without any restrictions on which DLLs can be called. This means anyone with database access, legitimate or not, can gain access to the *call spec* and run OS commands as the Oracle software owner account.
- The Listener process is owned by Oracle on *nix systems and the Local System on Windows. This account is more powerful than is

Continued

needed to run the Listener. This becomes a significant issue if ExtProc is enabled. Even without ExtProc, the existence of buffer overflow vulnerabilities in the Listener could allow for arbitrary code execution. If the Listener is running with a powerful OS account, the damage that can be done by such an attack is vastly increased.

It should be no surprise that the default configuration of the Listener is not secure. Oracle has clearly taken the position (without saying it) that security is the responsibility of their customers. Oracle users need to learn the intricacies of the system, learn about the security features offered, and then enable them. You will find this to be true for many different areas of the database. While Oracle has taken steps in the right direction, a truly secure by default database platform looks to be a long way away.

Valid Node Checking

Valid Node Checking is one of the lesser known but more useful security features offered by the Oracle Listener. By configuring a set of either invited or excluded nodes, you can take control of which hosts are allowed to make connections to your database. This connection filtering is done at the IP address or hostname level, meaning that it is not foolproof, but it does add a layer of protection that takes some technical skill to get around. Valid Node Checking is simple to set up and can be implemented on almost any Oracle database.

In order to determine if this feature makes sense for each of your databases, you will need to get an understanding of which machines make connections to the database for legitimate purposes. In some cases, this is simple; connections are made only by the application whose data lives in the database and from the local host for administration. In other cases it can be more complex, particularly when a single database is hosting several applications. There will be some cases where a database must accept connections from a large number of hosts, when the list of hosts that connect is constantly changing, or even some organizations that change both IP addresses and hostnames for their machines on a somewhat random basis. For these systems, Valid Node Checking is not a viable feature. Use your judgment on whether it will work for you before you make yourself nuts trying to implement it in an impossible situation.

For the majority of systems though, there are a finite number of hosts with a business reason to connect to your database. You can get a list of hosts that connect by examining the Listener log file, a task that you can automate by writing some code to mine the log (see the earlier section on "Listener Logging and Tracing"). With a list of hosts that have connected to the database, you can determine which ones are valid and allow only those hosts. In some cases, it may be easier to determine which hosts are not allowed (meaning everyone else is okay). The Valid Node Checking feature allows you to work either way, listing either the invited hosts or the excluded hosts. Note that you only need to configure one of these lists. Any host not on the invited list will automatically be excluded, while any host not on the excluded list will automatically be invited. If you are listing hosts as invited nodes, make sure to include *localhost*!

```
Valid Node Checking is enabled by adding the following lines to your sqlnet.ora
file:
tcp.validnode_checking = yes
tcp.invited_nodes = (localhost, AppServer, DBAworkstation, 192.168.100.21)
```

If you want to list excluded nodes instead, replace *tcp.invited_nodes* with *tcp.excluded_nodes*. You will need to list the hostnames and/or IP addresses all on one line. No wildcards are allowed; you must list each invited or excluded node individually.

When attempting to login to a system that has Valid Node Checking enabled, and you are connecting from an excluded node, you will get the following error (using SQLPlus as an example):

```
$ sqlplus system/manager@orcl

ERROR:
ORA-12537: TNS:connection closed
```

Summary

The Oracle TNS Listener is the database's gateway to the outside world, brokering network connections to the database and running external procedures on behalf of the database. The Listener is a crucial component of the Oracle infrastructure, and protecting it from attack must be taken seriously. Made up of a pair of executable binaries along with three configuration files, the Listener is a simple component, which is quite simple to lock down. However, many DBAs ignore Listener security, not realizing the level of damage that can be done if it is breached.

Over the years, many vulnerabilities have been found in the Oracle Listener. DoS attacks, buffer overflows, and common misconfigurations all leave the Listener vulnerable to a knowledgeable or determined attacker. Apply the latest patches, and you will eliminate most of the DoS issues and all of the known buffer overflows. Configure Listener settings securely by setting a strong password, turning on *ADMIN_RESTRICTIONS*, and enabling Listener logging. For systems that use ExtProc, set up a dedicated Listener to run the external procedures. Always remove the ExtProc entry from your primary database Listener. Finish the job by enabling Valid Node checking where it is practical to do so. Each of these steps will raise the bar on the security of your Listener and will help to reduce the chances of your database being successfully attacked. Don't fall into the default configuration trap! Take some time, lock down the Listener, and reap the rewards of better database security.

Solutions Fast Track

Introduction to the TNS Listener

- ☑ The Oracle TNS Listener is the component of the database system that brokers network connections to the database. It accepts incoming network connections, checks to make sure a valid database connection request has been sent, and then passes on communication to the appropriate SID.
- ☑ The Listener is made up of five pieces, two executables, and three configuration files. The main listener executable is *tnslsnr*, which runs all Listener operations and is usually run by the Oracle software owner account. A second executable, *lsnrctl*, is provided to send configuration commands to

the Listener, query the Listener for status, and stop/start the Listener process. The configuration files store the configuration parameters for the Listener, with *listener.ora* being the primary configuration file.

☑ With the release of Oracle10g, some key changes were made to the Listener to enhance security. Specifically, Oracle added the option for using local OS authentication to connect to the Listener. They enabled Listener security by default using local OS authentication only. This is intended to block any remote Listener administration, and has been quite effective.

Listener Vulnerabilities "By Design"

☑ There is no account lockout feature for the Oracle Listener. This means that an attacker can connect to the Listener and guess passwords forever without the Listener taking steps to protect itself. Brute-force password guessing against an Oracle Listener is a viable attack.

☑ When authenticating across the network, Oracle sends the Listener password in cleartext, providing an attacker that is monitoring network communications with the Listener password. With password in hand, an attacker can reconfigure the Listener to block access to the database or worse to take control of the database server.

☑ When authenticating to the Listener (versions prior to 10g only), it will accept either the correct Listener password or the Listener password hash. This means that the Listener password hash stored in *listener.ora* needs to be protected as if it were a cleartext password. Users that are not allowed to administer the Listener must be prevented from having any access (including read-only) to the *listener.ora* file.

Fixing Listener Vulnerabilities by Applying Oracle Patch Sets and CPUs

☑ Over the last five years, a large number of vulnerabilities have been discovered in the TNS Listener. These vulnerabilities are caused by coding errors made by Oracle developers. Oracle fixes these vulnerabilities once they are reported, then distributes the fixes together with database Patch Sets and CPUs.

☑ DoS weaknesses make up the majority of Listener vulnerabilities that can only be fixed by applying patches. A DoS attack against an Oracle Listener can manifest itself as a DoS against the database, as all new network connections to the database must be brokered by an active, working Listener.

☑ Several buffer overflows have been found in the Listener. These are particularly troubling, because in general they allow an unauthenticated user to run arbitrary code on the database server, with the authority of the OS account that owns the Listener (typically Oracle on *nix and the Local System on Windows). Listener buffer overflows can give an attacker complete control over the database server and all the data it contains.

Securing the Listener Configuration

☑ Make sure Listener Security is enabled by setting a strong password for each TNS Listener in your environment. By setting a password (and thus enabling Listener Security), you restrict the ability to run any Listener commands to those people who know the password.

☑ Enable *ADMIN_RESTRICTIONS* on each and every Listener on your network. *ADMIN_RESTRICTIONS* blocks all *SET* commands from running, forcing you to administer the Listener locally by modifying the *listener.ora* file. This blocks anyone from remotely administering the Listener, a capability which when put into the wrong hands can quickly compromise your database.

☑ Enable Listener Logging and regularly review the log files. The Listener logs all sorts of important security information, such as which hosts are connecting to the database, who is administering the Listener (or trying to and failing because of your security controls), and what external procedures are being run.

☑ Remove any ExtProc entries from your primary Listener (the one that handles connections to the database). Instead, set up a dedicated Listener that runs with a minimally privileged account to run ExtProc. If you are not actively using ExtProc, make sure the feature is disabled by removing ExtProc entries from all Listeners.

Valid Node Checking

- ☑ Valid Node Checking is a Listener feature that allows you to specify which hosts can and cannot make network connections to each Oracle database.
- ☑ In environments where it is practical to determine which hosts should be able to connect to the database, it makes sense to enable Valid Node Checking. For environments that constantly change, or where dozens or more clients are allowed to connect, Valid Node Checking is probably not worth implementing.
- ☑ Enable Valid Node Checking in the *sqlnet.ora* file. You must provide a list of either invited hosts (those that are allowed to connect), which automatically excludes everyone else, or a list of excluded nodes (those that are not allowed to connect) which automatically invites everyone else.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Can I get rid of my TNS Listener and all of its associated vulnerabilities and have the database manage network connections directly instead?

A: No. Oracle requires the Listener be present and operational in order to initiate network connections with a database.

Q: Oracle says that the 10g Listener is secure by default. Does it make sense for me to use the default configuration?

A: While the Oracle10g Listener has some significant security improvements over previous Listeners, we still recommend making some changes. Set a listener password, enable *ADMIN_RESTRICTIONS*, turn on Logging, and remove ExtProc.

Q: Does the Listener really proxy all communications with the database, or is it only used to set up network connections?

A: The Listener is only involved in setting up the communications channel with the database. Once the Listener has validated a connection request, it usually redirects the client to connect to a different TCP port from the Listener port. Once the connection is made to the new port, communication is direct with the database and no longer passes through the Listener.

Q: What is the best way to set a password on my Listener?

A: The best method to set the password is to use the Listener Control Utility (ls*n*rctl). This means removing *ADMIN_RESTRICTIONS* for a moment while the password is set or changed. While passwords can be written directly into *listener.ora*, writing them there directly will result in the password being stored in cleartext. When passwords are set through *lsnrctl*, they are hashed and stored as a hash. While this has no impact on Oracle 9i and older, with 10g, this provides added security, because authentication is only allowed with the true password, not the password hash.

Q: Can a determined attacker get around Valid Node Checking? If so, how?

A: Yes. Valid Node Checking is not unbreakable. The filtering is performed based on IP addresses or hostnames, both of which are spoofable with the right set of tools. For a knowledgeable attacker, this is trivial; however, it is like the lock on your front door. It helps to keep the honest people honest.

Chapter 4

Managing Default Accounts

Solutions in this chapter:

- The Role of Oracle Default Accounts from 9i to 10g
- Lock Accounts and Expire Default Passwords
- Configure Strong Passwords
- Unlock Accounts and Configure Impossible Passwords
- Automating the Process of Identifying Default Accounts

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Oracle databases have a dizzying array of features and options available to tune and manage the database. Many of these features have been implemented such that they require an account to run them and own their data. When Oracle is installed, many features are installed by default, leading to a large number of default accounts existing in the database. Each default account has a default password. This has become a serious problem as the default passwords have been published and now present an open door to any attacker who wants access to an Oracle database. There are even several hacker tools out there that scan for the existence of default accounts and passwords, making any enabled default account an easy target for attack.

The problem is not just limited to Oracle's default accounts. Other vendors use defaults as well. Of importance to us are the application accounts used to connect to the database. These accounts, which we will refer to as well-known accounts, are also a problem as their usernames and passwords have also been published. We firmly believe that default accounts with default passwords are the biggest database security issue that exists today. In this chapter we will take you through the steps of identifying default accounts in your databases and then discuss several strategies to secure them.

Here is an example using the Oracle Password Guesser written by Patrik Karlsson and available on www.cqure.net.:

```
c:\opwg -s 127.0.0.1
Oracle Password Guesser v1.3.1 by patrik@cqure.net
--------------------------------------------------
Skipping PLSExtProc ...
INFO: Running pwcheck on SID orcl
Successfully logged in with DBSNMP/DBSNMP
Successfully logged in with SCOTT/TIGER
Successfully logged in with OUTLN/OUTLN
Successfully logged in with SYSTEM/MANAGER
Successfully logged in with CTXSYS/CTXSYS
Successfully logged in with XDB/CHANGE_ON_INSTALL
```

Tools & Traps…

Attacking Default Accounts in Oracle

The first thing that an attacker tries when attempting to break into an Oracle database is to log in using default accounts with default passwords. It is simply the easiest and most successful way into the system. Until Oracle10g, default accounts were enabled on install and each had a set password and often powerful permissions. Attackers take advantage of the fact that most companies don't have a process to ensure default accounts are removed, locked, or have their passwords changed. The success rates of breaking into a production Oracle database using default accounts and passwords are incredibly high.

There are thousands of combinations of default credentials that work for various versions of Oracle. Several accounts have been removed or have had their default passwords changed with various Oracle releases, but many constants remain. Automated tools and scripts make it easy for anyone to get a list of default Oracle database accounts and then attempt to log in to any Oracle database using those accounts.

One of the most well known resources for obtaining information about Oracle database defaults is from Pentest Limited, an Information Technology (IT) security and penetration testing services firm based in the United Kingdom. Pentest Limited maintains a script that is available for public download that attempts to find default accounts with default passwords in Oracle. The script requires an Oracle database user that has rights to select from the DBA_USERS view, however, the script contains a listing of usernames and passwords that can easily be extracted and used to attempt to log in to the database with each account. This script can be found at www.pentest.co.uk/sql/check_users.sql

Pentest Limited is no longer the best source of this information as they have not kept up with the changes over the last few years. Others have picked up the torch and today there is more detailed information along with better tools than ever before.

Pete Finnigan is one of the world's leading experts on Oracle security. On his Web site (www.petefinnigan.com), he maintains an updated listing of Oracle default accounts. The current version has 598 unique combinations covering several versions of Oracle. The list is available in a variety of formats, making it easy to import the data into an attack tool. Pete Finnigan also offers an automated script for public download, however, it also requires a valid

Continued

database log in with permissions on DBA_USERS. The list of default passwords can be found at www.petefinnigan.com/default/default_password_list.htm and the test script can be found at www.petefinnigan.com/default/default_password_checker.htm. Running the script yields results like this one:

```
Username: SCOTT
Password: TIGER
-----------------------------------------------
WARNING! The password of SCOTT is a default password. It is well known to
hackers


Additional information:
This is a training account. It should not be available in a production
environment.
```

Too many organizations make it all too easy to break into their Oracle databases by leaving default accounts in place with their default passwords. By simply changing these defaults, many hackers will move on to the next target, looking for easy pickings.

The Role of Oracle Default Accounts From 9i to 10g

Ever since the beginning of time, Oracle has shipped with default accounts. As more optional features have been created both in and around the database, the number of default accounts installed with the database has grown significantly. These accounts serve a number of purposes and it is unlikely that any single database installation really requires the use of every default account. It's important to have an understanding of what default accounts exist in each database and what they are used for. It's also important to know about the changes Oracle has made over time. With the release of Oracle database 9i and 10g, major changes were made to the way default accounts are handled.

By the time Oracle9i was in development, it had become clear that default accounts and passwords in any system represented a significant security issue. The default account names and their passwords had been appearing on hacker sites and vulnerability tracking sites for quite some time. Oracle had received their fair share of complaints from the community, so they took action and with the release of 9i had locked and expired most of the default accounts in the database. This meant that a newly installed database had only eight of 32 default accounts unlocked and accessible and only five of those accounts had default passwords (the other three get random passwords). Oracle also began removing default accounts that were becoming unnecessary, further reducing the scope of the default account issue.

These were big steps for Oracle to take. They probably figured that with only eight accounts to deal with, most database administrators (DBAs) would change the password for each and, therefore, eliminate the issue. Unfortunately, they figured wrong and were forced to re-think their strategy when they released 10g. When you install Oracle today, only the SYS and SYSTEM accounts are unlocked. These accounts no longer have default passwords. Instead, you are forced to assign each your own password during the install. To be clear, default passwords still remain in place for all accounts that are expired and locked on install. As long as these accounts are never unlocked, this provides some security; however, we recommend not taking the risk and changing these passwords anyway. We'll discuss several strategies for dealing with default accounts later in this chapter.

Default Accounts

In order to create a sensible plan to deal with default accounts, it's important to understand what each account is there for. This key first step will form the basis of the security plan for default accounts, identifying those accounts that are not needed and can be removed from the database. The following list covers Oracle9i and 10g, presenting the account name, default password, versions affected, and a description of the account's function.

Account: ADAMS

Default password: WOOD
Password hash: 72CDEF4A3483F60D
Versions affected: 9i, 9iR2
Description: ADAMS is a sample schema owner and training account. It should always be removed in a production environment.

Account: ANONYMOUS

Default password: ANONYMOUS
Password hash: FE0E8CE7C92504E9
Versions affected: 10g

Description: ANONYMOUS is used to provide (you guessed it) anonymous access to an Oracle XML DB repository via Hypertext Transfer Protocol (HTTP). It's almost always a really bad idea to offer anonymous access. Oracle's documentation points out the security risks of enabling this account. If you are not granting anonymous access to XML DB, this account should be removed from the system.

Account: AURORA$JIS$UTILITY$

Default password: N/A (This account gets a randomly generated password)
Password hash: N/A
Versions affected: 9i
Description: JSERV account used by CORBA tools and Enterprise Java Beans. CORBA (ORB) is a standard for communication between objects in a distributed computing system. This account is created during the install of the Oracle Servlet Engine. Changing the password for this account is not allowed. However, if you are not using the Oracle Java Virtual Machine (JVM) or CORBA, this account should be removed.

Account: AURORA$ORB$UNAUTHENTICATED

Default password: N/A (This account gets a randomly generated password)
Password hash: N/A
Versions affected: 9i
Description: JSERV account used by CORBA tools and Enterprise Java Beans. CORBA (ORB) is a standard for communication between objects in a distributed computing system. This account is created during the install of the Oracle Servlet Engine. Changing the password for this account is not allowed. However, if you are not using the Oracle JVM or CORBA, this account should be removed.

Account: BLAKE

Default password: PAPER
Password hash: 9435F2E60569158E
Versions affected: 9i
Description: BLAKE is a training account. It should always be removed in a production environment.

Account: CLARK

Default password: CLOTH
Password hash: 7AAFE7D01511D73F
Versions affected: 9i
Description: CLARK is a training account. It should always be removed in a production environment.

Account: CTXSYS

Default password: CTXSYS or CHANGE_ON_INSTALL
Password hash: 24ABAB8B06281B4C or 71E687F036AD56E5
Versions affected: All
Description: CTXSYS supports the Oracle Text component of the interMedia (formerly ConText) Option. The interMedia feature allows the database to manage content (such as documents, audio, and video) in the same manner as it manages regular business data. This is a cool feature of Oracle, as it allows for indexing and searching of various document types, performing Structured Query Language (SQL) queries against documents and streaming audio, video, or images on the Web. CTXSYS is a highly privileged user with DBA level access. If you have no need for the interMedia option, the account can be removed. Otherwise, be sure to set a strong password and potentially lock the account.

Account: DBSNMP

Default password: DBSNMP
Password hash: E066D214D5421CCC
Versions affected: All
Description: DBSNMP is the account used to run the Oracle Intelligent Agent (OIA), which is the database resident component that allows the database to be managed by Oracle Enterprise Manager (OEM). OIA and therefore, DBSNMP, is used to discover databases that can be managed by OEM, monitor events on behalf of OEM, and execute tasks related to jobs submitted to OEM. DBSNMP is not a DBA, but has powerful permissions and can be used to become SYSDBA. Since OEM is widely used to manage Oracle databases, removing the OIA is probably not an option, but that doesn't mean that you need to keep DBSNMP. OEM allows you to specify both the username and password for the OIA account in the file *snmp_rw.ora*. Make sure you take advantage and at least set a strong password for this account.

Account: DIP

Default password: DIP
Password hash: CE4A36B8E06CA59C
Versions affected: 10g, 10gR2
Description: DIP is used by the Oracle Internet Directory (OID) and the associated Directory Integration Platform. OID is a Lightweight Directory Access Protocol (LDAP) v3 directory running on top of an Oracle database. DIP facilitates integration and synchronization of OID with other directories, allowing organizations to create a single directory system that provides access to resources across the enterprise. Oracle has chosen OID to replace Oracle Names as the product of choice for storage of database service names. If you are not using OID, the DIP account can be removed.

Account: DMSYS

Default password: DMSYS
Password hash: BFBA5A553FD9E28A
Versions affected: 10g, 10gR2
Description: DMSYS is the account used to run Oracle Data Mining (ODM). ODM is Oracle's solution for performing deep, detailed analysis on massive amounts of data in order to find patterns and gain new insights. Businesses use ODM for several purposes, including profiling customers, detecting fraud, and using historical data to predict future business trends. Oracle is a leader in the data mining market; however, data mining is not for everyone and certainly not for every Oracle database. If you aren't going to use ODM, you should remove the powerful DMSYS account.

Account: EXFSYS

Default password: EXFSYS
Password hash: 66F4EF5650C20355
Versions affected: 10g, 10gR2
Description: EXFSYS is used for Expression Filters, a new feature introduced in Oracle10g. Expression Filters allow you to define and store what amounts to a set of WHERE clauses that evaluate data as it is input into the database. The Expressions Filters can then alert you when data that match their evaluation criteria becomes available. While this feature is extremely useful, it is quite new and not very widely used. If you do not plan to use Expression Filters, you can remove the EXFSYS account.

Account: JONES

Default password: STEEL
Password hash: B9E99443032F059D
Versions affected: 9i
Description: JONES is a training account. It should always be removed in a production environment.

Account: HR

Default password: HR or CHANGE_ON_INSTALL
Password hash: 4C6D73C3E8B0F0DA or 6399F3B38EDF3288
Versions affected: All
Description: Human Resources sample schema owner. HR is a training account. It should always be removed in a production environment.

Account: LBACSYS

Default password: LBACSYS
Password hash: AC9700FD3F1410EB
Versions affected: All
Description: LBACSYS is the account used to manage Oracle Label Security (OLS). The account is created when OLS is installed and acts as the administrator for the OLS system. If you are not using Label Security, there is no need for the LBACSYS account and it should be removed.

Account: MDDATA

Default password: MDDATA
Password hash: DF02A496267DEE66
Versions affected: 10g, 10gR2
Description: The MDDATA account is used by the Oracle Spatial Option, which provides location-based services to applications running on Oracle databases. Oracle Spatial is typically used in applications designed for wireless, online, and in-vehicle deployments. The MDDATA schema is used specifically for the Geocoder (computes latitude/longitude coordinates for a given address) and the Router (generates routes such as driving directions between two locations). If you are not using the Oracle Spatial Option, MDDATA should be removed.

Account: MDSYS

Default password: MDSYS
Password hash: 72979A94BAD2AF80
Versions affected: All
Description: MDSYS is the Oracle Spatial Option administrator account. This is a powerful account, with an access level roughly equivalent with that of a DBA. There are several attacks out there that target packages owned by MDSYS. It is critical to remove this account if you are not using Oracle Spatial. Otherwise, make sure you set a strong password for MDSYS and keep your database patches up-to-date.

Account: ODM

Default password: ODM
Password hash: C252E8FA117AF049
Versions affected: 9iR2, 10g, 10gR2
Description: The ODM account and schema is used by Oracle Data Mining (see DMSYS for an explanation of Oracle Data Mining). Remove this account if you are not using ODM. Otherwise, be sure to set a strong password.

Account: ODM_MTR

Default password: MTRPW
Password hash: A7A32CD03D3CE8D5
Versions affected: 9iR2, 10g, 10gR2
Description: The ODM_MTR schema is used to store sample data for Oracle Data Mining (see DMSYS for an explanation of Oracle Data Mining). ODM_MTR is the least powerful of the ODM accounts; however, it has been granted some roles and permissions that would be extremely useful to an attacker. This account should be removed in a production system.

Account: OE

Default password: OE or CHANGE_ON_INSTALL
Password hash: D1A2DFC623FDA40A or 9C30855E7E0CB02D
Versions affected: All
Description: Order Entry (OE) sample schema owner. OE is a training account. It should always be removed in a production environment.

Account: OLAPDBA

Default password: OLAPDBA
Password hash: 1AF71599EDACFB00
Versions affected: 9i
Description: The OLAPDBA account is one of several accounts used by Oracle OnLine Analytical Processing (OLAP). OLAP services provide fast analysis of multidimensional information, primarily used in creating financial reports, budgeting, and business forecasting. OLAPDBA is the administrator account for OLAP services. If you are not using OLAP, the account can be removed. Otherwise, assign OLAPDBA a strong password, then configure OLAP services to use the new password using the OLAP Instance Manager tool.

Account: OLAPSVR

Default password: OLAPSVR or INSTANCE
Password hash: 3B3F6DB781927D0F or AF52CFD036E8F425
Versions affected: 9i
Description: The OLAPSVR account is used by Oracle OLAP Services. OLAPSVR is used as a proxy account for OLAP. If you are using OLAP, assign a strong password to OLAPSVR and then input that password into the OLAP Instance Manager tool. Be sure to restart the instance of OLAP services after changing the password.

Account: OLAPSYS

Default password: OLAPSYS or MANAGER
Password hash: C1510E7AC8F0D90D or 3FB8EF9DB538647C
Versions affected: All
Description: The OLAPSYS account is used by Oracle OLAP Services. OLAPSYS owns the schema that holds the OLAP catalog, a collection of metadata that manages OLAP constructs such as dimensions, measures, cubes, levels, and hierarchies. Several significant vulnerabilities have been discovered in packages owned by OLAPSYS. If you are not using OLAP Services, you should remove OLAPSYS. Otherwise, be sure to configure a strong password for the account.

Account: ORDPLUGINS

Default password: ORDPLUGINS
Password hash: 88A2B2C183431F00
Versions affected: All
Description: The ORDPLUGINS account is a component of interMedia. ORDPLUGINS owns the schema used to install both Oracle and third-party plug-ins used to extend the functionality of interMedia to support new data types, new data sources, and processing of audio or video data. The ORDPLUGINS schema is required for proper operation of interMedia, so it should not be removed independently of other interMedia components. If you're not using interMedia, remove ORDPLUGINS. Otherwise, be sure to configure a strong password for the account as ORDPLUGINS is highly privileged.

Account: ORDSYS

Default password: ORDSYS
Password hash: 7EFA02EC7EA6B86F
Versions affected: All
Description: The ORDSYS account is the administrator for interMedia and owns the schema that holds all code and default data for the interMedia option. Oracle Time Series (used to timestamp a set of data) is also installed in the ORDSYS schema. ORDSYS has some powerful permissions, including the ability to run commands on the database host operating system (OS) using external libraries. There have also been a number of vulnerabilities discovered in the ORDSYS schema, including susceptibility to Denial of Service (DoS) attacks that can bring the database to a grinding halt. Be sure to set a strong password for ORDSYS if you are using interMedia. Otherwise, remove the account from the database.

Account: OSE$HTTP$ADMIN

Default password: N/A (This account gets a randomly generated password)
Password hash: N/A
Versions affected: 9i
Description: The account OSE$HTTP$ADMIN is used to run the Oracle Servlet Engine for Enterprise Java Beans and CORBA tools. Changing the password for this account is not allowed. However, if you are not using the Oracle JVM or CORBA, this account should be removed.

Account: OUTLN

Default password: OUTLN
Password hash: 4A3BA55E08595C81
Versions affected: All
Description: The account OUTLN is used to support Oracle's Plan Stability feature. OUTLN owns the schema used to store and manage outlines, essentially query executions plans that ensure SQL queries will be run the same way regardless of changes to data in or the structure of the target tables, changes in system configuration, or even use of different optimizers. The OUTLN account is a DBA, so it is critical to protect this account with a strong password. If you are not using Plan Stability, OUTLN should be removed.

Account: PM

Default password: PM
Password hash: C7A235E6D2AF6018
Versions affected: All
Description: Product Media (PM) sample schema owner. PM is a training account. It should always be removed in a production environment.

Account: QS

Default password: QS
Password hash: 4603BCD2744BDE4F
Versions affected: All
Description: Queued Shipping (QS) sample schema owner. QS is a training account. Several other training accounts exist to support the Queued Shipping sample: QS_ADM, QS_CB, QS_CBADM, QS_CS, QS_ES, QS_OS, QS_WS. Each of these supporting accounts have a default password that is the same as their username. Every one of the QS accounts should be removed in a production environment.

Account: RMAN

Default password: RMAN
Password hash: E7B5D92911C831E1
Versions affected: 10g, 10gR2
Description: The account RMAN is used by the Oracle Recovery Manager, the tool Oracle recommends for backing up and recovering Oracle databases. We recommend

it as well, particularly if you enable the encrypted backup option in 10gR2. The RMAN account is used to collect the backup data and ship it off for storage; therefore, this account has access to data in the database and may be used to inappropriately modify data as it is being backed up. If you are using Recovery Manager, be sure to set a strong password for RMAN, otherwise, the account should be removed.

Account: SCOTT

Default password: TIGER
Password hash: F894844C34402B67
Versions affected: All
Description: The account SCOTT is one of the most well-known Oracle default accounts. SCOTT owns sample schemas that are often referenced in Oracle documentation. Despite SCOTT's popularity, the account is for training purposes only and should always be removed in a production environment.

Account: SH

Default password: SH
Password hash: 54B253CBBAAA8C48
Versions affected: All
Description: Sales History sample schema owner. SH is a training account. It should always be removed in a production environment.

Account: SI_INFORMTN_SCHEMA

Default password: SI_INFORMTN_SCHEMA
Password hash: 84B8CBCA4D477FA3
Versions affected: 10g, 10gR2
Description: The account SI_INFORMTN_SCHEMA is used to support the Oracle interMedia Option. SI_INFORMTN_SCHEMA owns the schema that stores data for the SQL/MM Still Image Standard. This standard allows the database to do all sorts of interesting things with images using SQL, from image storage and retrieval, to complex searches on visual aspects such as size, color, and texture. If you are not storing or processing images in your database, you should remove this account. If SI_INFORMTN_SCHEMA is in use, be sure to configure a strong password for the account.

Account: SYS

Default password: CHANGE_ON_INSTALL
Password hash: D4C5016086B2DC6A
Versions affected: All
Description: The most powerful account in the Oracle database, SYS owns the schema that holds the data dictionary. Objects owned by SYS are extremely sensitive and are not directly modifiable, as mistakes could easily lead to functional problems in the database. The SYS account is required for the proper operation of Oracle; it can never be removed or replaced. It is absolutely critical to set an extremely strong password for SYS immediately after installing the database (or during the install with 10g). Never, ever leave the SYS password set to CHANGE_ON_INSTALL on any Oracle database, production or not.

Account: SYSMAN

Default password: OEM_TEMP
Password hash: 639C32A115D2CA57
Versions affected: 10g, 10gR2
Description: The SYSMAN account is used to manage OEM. This is an extremely powerful account as it has rights on any database managed by OEM. Make sure to set a strong password for SYSMAN when you install the database.

Account: SYSTEM

Default password: MANAGER
Password hash: D4DF7931AB130E37
Versions affected: All
Description: The SYSTEM account is another extremely powerful Oracle account, second only to SYS. SYSTEM is more than just a DBA, as the account owns and has access to internal tables and views that contain administrative data and are used by Oracle tools to manage and monitor the database. It is critical to ensure that the SYSTEM account has a strong password set immediately after installing the database (or during the install with 10g). Never run any Oracle database with the SYSTEM password set to MANAGER.

Account: TSMSYS

Default password: TSMSYS
Password hash: 3DF26A8B17D0F29F
Versions affected: 10gR2
Description: The account TSMSYS is used to perform Transparent Session Migration, a feature that supports grid computing. Transparent Session Migration migrates database sessions (essentially connections) from instance to instance in a manner that is completely transparent to the end user. This can be used to implement load balancing across the grid and can handle moving sessions off an instance that is about to be shut down. Not in a grid? TSMSYS should be removed.

Account: WK_TEST

Default password: WK_TEST
Password hash: 29802572EB547DBF
Versions affected: 10g, 10gR2
Description: The account WK_TEST is used by the Oracle Ultra Search feature. Ultra Search is a tool that Oracle provides with their database, application server, and collaboration suite that makes it simple to index and search databases, Web pages, files, and e-mail for text strings. Ultra Search is dependant on Oracle Text and on the CTXSYS schema. WK_TEST owns the schema used to host the default instance of Ultra Search (WK_INST). If you are not using Ultra Search, the WK_TEST account should be removed. Otherwise, configure a strong password for the account and then set that same password in the Ultra Search Administration Tool on the Edit Instance page.

Account: WKPROXY

Default password: WKPROXY or CHANGE_ON_INSTALL
Password hash: AA3CB2A4D9188DDB or B97545C4DD2ABE54
Versions affected: 9iR2, 10g, 10gR2
Description: The account WKPROXY is also used by the Oracle Ultra Search feature. Similar to the other Ultra Search accounts, WKPROXY should be configured with a strong password or removed entirely if you are not using Ultra Search.

Account: WKSYS

Default password: WKSYS or CHANGE_ON_INSTALL
Password hash: 545E13456B7DDEA0 or 69ED49EE1851900D
Versions affected: 9iR2, 10g, 10gR2
Description: The account WKSYS is the admin account for Oracle Ultra Search. WKSYS is an extremely powerful account that has been granted the DBA role along with several "ANY" system privileges. WKSYS owns many of the packages used to administer and run Ultra Search. If you are not using Ultra Search, the WKSYS account should be removed. Otherwise, be sure to give it a strong password after install.

Account: WMSYS

Default password: WMSYS
Password hash: 7C9BA362F8314299
Versions affected: 9iR2, 10g, 10gR2
Description: The WMSYS account owns the schema used to store metadata information for the Oracle Workspace Manager. Workspace Manager is a tool used to managed multiple versions of data within the same database. This feature is typically used to run queries against historical, current, and proposed future versions of data for various types of analysis. WMSYS is granted the powerful permission UNLIMITED TABLESPACE and has rights to execute functions known to be vulnerable to buffer overflow attacks that allow for arbitrary code execution on the database server. Be sure to remove this account if Workspace Manager is not in use. Otherwise, give the account a strong password after install.

Account: XDB

Default password: CHANGE_ON_INSTALL
Password hash: 88D8364765FCE6AF
Versions affected: 9iR2, 10g, 10gR2
Description: The XDB account is used to manage Oracle's XML database (XDB). The XDB account is granted the built-in role RESOURCE. If you are not using XML DB, the account can be removed.

Notes from the Underground…

DBSNMP Account

No matter where we find ourselves, if we are looking at an Oracle database, the chances are extraordinarily high that we will find the DBSNMP account enabled with its default password in place. It is astounding how few systems have actually secured this extremely powerful account. Leaving DBSNMP in place with its default password creates an enormous hole in the security of a database, regardless of how many firewalls protect it from the Internet. The trend of leaving DBSNMP in place may be surprising from a security perspective, but from an operations perspective, it's the easiest way to ensure an Oracle database can be managed by Oracle Enterprise Manager (OEM).

If you were a database hacker, you would know about this already. Accessing databases for nefarious purposes by exploiting the default password for DBSNMP is commonplace. DBSNMP has powerful permissions in Oracle; by default, the account has rights to select from many data dictionary tables and views. In a nutshell, if an attacker can get access via DBSNMP, they can collect the information they need to assume the role of SYSDBA and take ownership of your database and every bit of data inside it.

What many DBAs do not realize is that they can fix the problem with DBSNMP without affecting OEM's ability to monitor and manage the database. Within the configuration files for OEM, there are entries where a username and password can be specified to run the database Simple Network Management Protocol (SNMP) system. It's simple. First, change the username and password of DBSNMP in the database. Next, make the corresponding changes in the *snmp_rw.ora* (typically located in the directory */var/opt/oracle*, or *$ORACLE_HOME/network/admin*) for each *{SID}* that you want the agent to connect to. Set the following lines:

```
snmp.connect.{SID}.NAME = dbsnmp (or new account name)
snmp.connect.{SID}.PASSWORD = {new password}
```

Locking down the DBSNMP account is a critical step in securing your Oracle database. DBSNMP is considered low-hanging fruit by the hacker community, as it is often left unsecured. Take the time to modify your configuration to ensure that your database does not fall victim to attack by one of the most powerful and most well-known default accounts out there.

There are several approaches you can take to managing default accounts in your database. You will find it to be extremely helpful to create a plan for managing default accounts when you first install a database and then periodically review that plan to ensure it continues to meet the needs of your system. Get started with a solid understanding of how applications will use the database and what features are and are not required. For each account, decide upon the technique that you'll use to secure it and write it all down to create a default account checklist. With checklist in hand, you can implement your plan for each database individually, ensuring that only the features and users that you require are present and those that are present are properly locked down.

Lock Accounts and Expire Default Passwords

The strategy Oracle has taken for dealing with default accounts is to lock and expire each of the default accounts that are not critical to the operation or management of a database. This is a good starting point and, as long as nothing changes, it provides adequate protection for default accounts. This approach, however, is lacking. It's all together too easy for one or several of these accounts to be unlocked with their passwords unchanged.

There are two primary ways that we have seen locked default accounts become unlocked and exposed. One way is by simple user error. Perhaps a newbie DBA finds the locked accounts and in an attempt to be helpful unlocks each one. This can happen quietly and will not affect the operation of the database, potentially creating an exposure that can last for a long time without being noticed.

Perhaps more common is the case where a feature is needed that requires one of the default accounts. Particularly if the request is made to enable the feature with some time urgency, there is a tendency among DBAs to simply get the job done, unlocking the account to enable the feature without taking the time to change the password and reconfigure whatever components rely on it.

Because of these shortcomings, we recommend that you never rely on locking and expiring alone to secure default accounts. Additional steps must be taken to provide appropriate security for systems containing or linked to production data.

Configure Strong Passwords

It's impossible to go about securing any application without some understanding of what makes up a strong password versus a weak one. Configuring strong passwords is a cornerstone of securing Oracle, so it is imperative to learn about password strength and about the methods attackers use to guess passwords in order to penetrate systems. In all Oracle systems, some default accounts (such as SYS and SYSTEM) can only be secured by establishing a strong password. For those systems that use some of the optional features in the database, the default accounts required to run those features must also be protected with a strong password.

Password strength is measured by two characteristics, *length* and *complexity*. Password length is the simplest to understand. In general, the longer the password, the more difficult it is to guess and the stronger it is. While it is not entirely correct to assume that password strength increases linearly with an increase in the number of characters in the password, there is certainly a correlation between strength and length.

Password complexity is more subtle and includes several factors. It's easiest to explain by discussing the ways that most people choose their passwords and then by looking at the tools attackers build and use to guess passwords and break into systems. It's very common for people in the workforce today to require access to several different systems in order to do their jobs—each system with its own authentication system. At the same time, with the growth of the Internet, people have several personal accounts that they use for various purposes (e-mail, banking, travel, news, and so on) that also require passwords. This means that folks are asked to remember several, possibly dozens of passwords at any given time. Making the situation more difficult, we are often told not to write our passwords down, to remember them instead. This forces even security conscious people to choose easy to remember passwords and to reuse the same passwords on multiple systems. The easiest passwords to remember are simple words, places, dates, or easy-to-type text strings. Favorite sports teams, cities, names, birthdays and even strings like "12345" or "qwerty" are very commonly used. These are all weak passwords. What makes them weak are the hacker tools used to "guess" passwords, generally with a method often called brute-force. In the simplest implementations, password guessing tools are scripts that repeatedly attempt to log in to a system using a set of known passwords stored in a file referred to as a *password dictionary*. When the script is successful in logging into a system, it stops and displays the password to the attacker.

Password dictionaries range from as small as a few dozen words to as large as 100,000 entries or more. These password dictionaries generally include all the words in the regular dictionary along with names of places, teams and other commonly used passwords (common strings of numbers or letters). Fancy password guessers can even run permutations on the words in their dictionaries, replacing certain letters with numbers, a technique often called hacker speak. Replace the letters "i" and "l" with the number 1, replace "e" with 3, "a" with @, "s" with $, and so on. The tools do the work for the hackers and they do it lightning fast.

When choosing a strong password, keep the hacker community and their tools in mind. The best passwords are those that cannot be remembered. Long strings that include a jumble of letters, numbers, and special characters are best. There are password management tools out there that will generate strong passwords for you and then store them securely. My favorite is a free tool called Password Safe, which can be downloaded from http://sourceforge.net/projects/passwordsafe.

If you don't have password management software available to you, start with a phrase you can remember. Next, find a way to mix in upper- and lower-case characters. This can be done by alternating the first and last letter of each word as upper case (e.g., The browN Fox jumpS Over thE Lazy duckS). Now remove the spaces, replacing some with special characters (e.g., ThebrowN-Fox_jumpS=Over-thELazyduckS). Finally, replace some of the letters with numbers (e.g., Th3br0wN-Fox_jump$=Ov3r-thEL@zyduckS). You get the idea—there are many options. In this example, we have constructed a strong password that remains possible to remember. (Maybe you can remember it; I'm sticking with the password management tools!).

Configuring strong passwords is one of the primary strategies we recommend for dealing with default accounts. You should ensure that any default accounts that cannot be removed from the database are configured with a strong password.

Unlock Accounts and Configure Impossible Passwords

An Impossible Password is not just a really, really strong password. It's actually a password that can be configured but is impossible to use to log in to Oracle. In order to understand this method, you must know a little about how Oracle stores passwords and performs user authentication.

Computer systems that include a user authentication function reliant on usernames and passwords need a way to verify that when a user attempts to log in, they have provided valid credentials. This means storing everybody's password somewhere. For a long time, this was a problem. The industry looked to cryptographers for a solution and before long passwords were stored as a hash. Little has changed since then. Most systems continue to use hash values for storing passwords, Oracle included.

NOTE

A hash is a special mathematical function that has several important properties that make it particularly suitable for storing passwords (it's also often used for data integrity checking). First and foremost, a hash is a one-way function. This means that it is impossible to determine the input to the function by examining the output; you cannot work backwards. The next important property is that unique input creates unique output. This means that any data you hash has a unique output value, or more simply put, no two differing inputs to a hash function can produce identical output. In fact with modern hash functions, changing one bit of the input string results in changing approximately 50 percent of the bits in the output string. Finally, hash calculations are consistent, meaning that the same input always results in the same output.

There are several hash algorithms in use today, some based on encryption algorithms (like the one used by Oracle), others designed purely for hashing. Each of these functions produces a fixed size output, called a *hash* or *message digest*. Output size is typically between 16 and 20 bytes. Commonly used hash algorithms include Message Digest 5 (MD-5), Secure Hashing Algorithm 1 (SHA-1), and SHA-256.

Oracle's Password Hashing Algorithm

Oracle had a lot of options when they decided on a password hashing algorithm. Their designers made the decision that a simple hash of a user's password was too weak and would be susceptible to dictionary attacks where the attacker calculates a hash for each word in a dictionary and then compares that hash with the value stored in the SYS.USER$ table. In order to add a layer of complexity, Oracle chose a keyed hash, where a secret value called a *key* is used as part of the hash calculation. Keeping the key secret was critical to the security of the hash algorithm, but keeping secrets is a difficult proposition in any software package. Oracle ended up designing their own hash algorithm based on the Data Encryption Standard (DES). It's likely that they considered a home-grown hashing algorithm to be more secure than an industry standard one, because the algorithm was to be kept secret. Security through obscurity is never a good thing, and it did not take long for both the hashing algorithm and the secret key to be discovered and disclosed to the public.

Oracle's password hashing algorithm is fairly simple. Start by concatenating the username with the password, then convert the resulting string to uppercase. Next, convert the string to double-byte characters (American Standard Code for Information Interchange [ASCII] is converted by simply setting the high order byte to 0) and add padding (bytes of 0's) until the string's length is a multiple of 8 bytes. Encrypt the string with Triple Data Encryption Standard (3DES) using a fixed key in CBC mode. Take the final 8 bytes of encrypted text and use it as a key to encrypt the whole encrypted string again using 3DES in CBC mode. The password hash is the final 8 bytes of the resulting string, converted to printable hexadecimal format.

TIP

When Oracle releases 11g, they will make an important change in the databases authentication system and thus to this algorithm. The database will be configured by default to enforce case sensitivity on passwords. No longer will the database convert each password to upper-case before running it through the password hashing algorithm. This allows users to add a significant amount of complexity to their passwords by mixing upper- and lower-case characters, but more importantly, vastly increases the effort required for an attacker to brute force their way into the database. In order to facilitate backwards compatibility, Oracle will include a configuration setting to disable this new feature and remove case sensitivity on passwords.

We strongly suggest that you do not disable this new security feature. Almost every other computer system that people interact with already enforces case sensitivity on passwords. In fact, most Oracle users presume that their passwords are already case sensitive and type their password in with the same case each time they log in. It will be a minor shift at most to have a whole organization remembering to input their password in the proper case, so take advantage of the situation and allow Oracle to help you improve the password security in your databases.

Defining Impossible Passwords

When configuring a password for a user, Oracle gives you the choice of calculating the password hash normally or bypassing the hash process and writing a value directly to the SYS.USER$ PASSWORD column. By writing a value directly, it is simple to store a password that can never be used to log in to the database. Remember, when you log in to Oracle, you are not given a choice about whether the password you provide should be hashed or not, the hashing is mandatory and automatic.

Consider the following example. Create a user named oracle with the password set to password as follows:

```
create user oracle identified by password;
```

The resulting entry in the SYS.USER$ table will look like:

```
USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACLE                         427458AF9CC65444
```

The value stored is a password hash, calculated as described above. When this user provides their credentials to Oracle to log in, Oracle will take the credentials, create the password hash, and then compare it with the value stored in SYS.USER$, find a match, and allow the user to connect. However, if we created the user and explicitly set the password value, things would be different.

```
create user oracle identified by values 'password';
```

The resulting entry in the SYS.USER$ table will look like:

```
USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACLE                         password
```

This time the value stored is the text string PASSWORD. This is an impossible password, because regardless of the input, the Oracle password hashing algorithm could never produce output that matches (remember passwords are stored as hexadecimal). This results in a situation where the user, in this case the user ORACLE, can never log in to the database because it has an impossible password.

Deploying Impossible Passwords

You're probably wondering why you would set an impossible password instead of simply locking an account. At the surface, it looks like the two methods accomplish the same thing, and as far as their primary function goes, they do. However, there are important differences that make the impossible password approach more secure.

When you attempt to log in to Oracle using an account that is locked, the database responds with an error message.

```
ERROR:
ORA-28000: the account is locked
```

Herein lies the problem. This error message offers an attacker more information than they need to know. Locking default accounts tells an attacker what schemas and therefore what features are installed on a given database. This allows them to create a plan of attack before they ever penetrate the database, building a list of potential exploits with the benefit of knowing which potentially vulnerable features are installed. Furthermore, the attacker now knows the password for each account is set to the default value! Setting an impossible password and unlocking the accounts prevents an attacker from logging in, but also provides no information about the existence of each account/schema or their password. When you attempt to log in to Oracle with a password that doesn't match, the database responds with an ambiguous message—the same message that you get when you attempt to log in with an account that doesn't exist.

```
ERROR:
ORA-01017: invalid username/password; logon denied
```

With impossible passwords, an attacker gets no information from an attempt to log in to the database with each default account and password. Less information is always better and it's our responsibility to make a hacker's job as difficult as possible. Finding places where information about the database flows too freely and eliminating them will go a long way towards achieving your security goals.

Automating the Process of Identifying Default Accounts

Manually verifying the existence of default accounts and then checking for their default passwords is a time-consuming and painful task for a single Oracle System Identifier (SID). Forget about trying to manually assess default accounts across an enterprise with dozens, hundreds, or thousands of Oracle databases. In order to effectively identify and eliminate the risks associated with default accounts that have default passwords, you have to find and deploy some kind of automated scanner.

There are several different approaches to automation that can be taken here. You can take the list of default accounts and password hashes from this book and build your own script to check for them. You can download one of the freely available Oracle default password scanners from the Internet. You can find and download the hacker tools that scan for Oracle default passwords. Or, you can go all the way and buy a commercial database vulnerability scanner that includes the capability to scan for Oracle default passwords. Regardless of the method you choose, we recommend that you run these password scans on each of your Oracle SIDs once a month, and then carefully review and deal with the results. Don't leave your system exposed to default passwords, it's too easy to remove them!

Creating Your Own Default Password Scanning Script

It's actually quite simple to create this script, but before you do, consider if it is an approach that will really meet your needs. There are a couple of significant drawbacks to writing your own script, first and foremost that you will essentially be "re-inventing the wheel." There are already free default password scanners out there that you can download. The code is written to scan for the passwords and a password list is included with the scanner. There are more interesting and fruitful ways to spend your time. If your goal is to secure Oracle, spend your time doing exactly that. There are a lot of steps to get there and while the default passwords are one of the most critical issues, they are one with the most resources out there to help you resolve.

Other aspects to consider about writing your own script are completeness and maintenance. We've covered only the commonly used feature set of the latest versions of Oracle here. If you are running older versions anywhere or using features we have not discussed, you'll need to expand the password list. Maintenance comes into play as well. As Oracle makes new releases and adds new default accounts, someone will need to go back and update the script to check for them. This part is easy to forget about, particularly if you don't know that a new default account has been introduced.

Still interested in writing your own script? Here is a plan for how it can be done:

1. Create a text file that lists the defaults you want to check for. An easy-to-use format is for each line in the file to identify one default account and its password hash, with a separator between the two.
2. In the script, start by finding and opening the password file. Prepare to read from the first entry.
3. Read in the next username and password hash from the file and store as variables *user* and *pwd*.

4. Determine if the account exists in the target database, and if so, collect the password hash.

```
SELECT PASSWORD FROM DBA_USERS WHERE USERNAME = user;
```

5. If the query returned a value, we know the account exists in the database. Compare the value returned by the database with the value stored in the variable *pwd*.
6. If the passwords match, print a helpful message, something to the tune of: "User: *user* exists in the database with a default password."
7. If you have not reached the end of the file, go back to step 3.

Using a Freely Available Default Password Scanner

One of the best and least expensive ways to identify default accounts in an Oracle database is to download and use one of the freely available default password scanners that have been posted on the Internet. There are several out there and some are better than others. Here are a few that we recommend:

- Checkpwd from Red Database Security
  www.red-database-security.com/software/checkpwd.html
- Default password scanner (DPS) from Oracle (Metalink 4926128)
  http://updates.oracle.com/ARULink/PatchDetails/process_form?patch_num=4926128
- Oracle Security Probe from Pete Finnigan (actually written by Marcel-Jan Krijgsman)
  www.petefinnigan.com/default/default_password_checker.htm

Each of these tools are quite similar to one another. All of them require you to provide a database login and all of them select from either DBA_USERS or SYS.USER$ and then compare the data selected to a dictionary of known default accounts and passwords. The tools are all fairly high speed, taking only a few seconds to scan an Oracle SID for a large list of default passwords. The most complete free tools scan for nearly 700 combinations of default accounts and passwords; however, commercially available tools scan for even more combinations. The great thing about these tools is that they are readily available, well tested, and periodically updated. All you need to do is download them and go. Each of the tools prints out some kind of report that shows which default passwords are in place. You can then work from this list to change the passwords or remove the accounts.

Here is a sample of the output you can expect from running a free password scanner. This from the Oracle Security Probe on Pete Finnigan's Web site.

```
*********************************************
*                                           *
*  Welcome to the Oracle Security Probe     *
*                                           *
*********************************************

Connectstring (destination database): orcl
Password of oraprobe?: ********
Connected.
Oracle accounts with default passwords
======================================

Username: SYS
Password: CHANGE_ON_INSTALL
-----------------------------------------------
WARNING! The password of SYS is a default password. It is well known to hackers

Additional information:
SYS is Oracle's most powerful database management account. It allows to read,
change and destroy all data in your database.

Username: SYSTEM
Password: MANAGER
-----------------------------------------------
WARNING! The password of SYSTEM is a default password. It is well known to
hackers

Additional information:
SYSTEM is Oracle's database management account. It allows to read, change and
destroy all data in your database.


Username: SCOTT
Password: TIGER
-----------------------------------------------
WARNING! The password of SCOTT is a default password. It is well known to
```

```
hackers

Additional information:
This is a training account. It should not be available in a production
environment.


Username: DBSNMP
Password: DBSNMP
----------------------------------------------
WARNING! The password of DBSNMP is a default password. It is well known to
hackers

Additional information:
DBSNMP is an account for the Oracle Intelligent Agent. Under certain
circumstances it allows to read passwords from memory.
```

Using a Commercial Database Vulnerability Scanner

The most powerful tools for finding default passwords in Oracle databases are sold commercially, generally as a component of a complete database vulnerability scanning product. These highly specialized products generally offer the most complete listing of default passwords across a wide range of Oracle database versions. The better of the products out there allow for scanning from both inside and outside the database, allowing the customers of these products to get a complete understanding of which accounts would be easily identified by a hacker looking for a way into the database, as well as a malicious insider who uses their existing database account to gain information about what default accounts exist in the system (by selecting from the ALL_USERS view). Commercial scanners also generally allow for scanning multiple systems in parallel, then aggregating the results into various reporting formats.

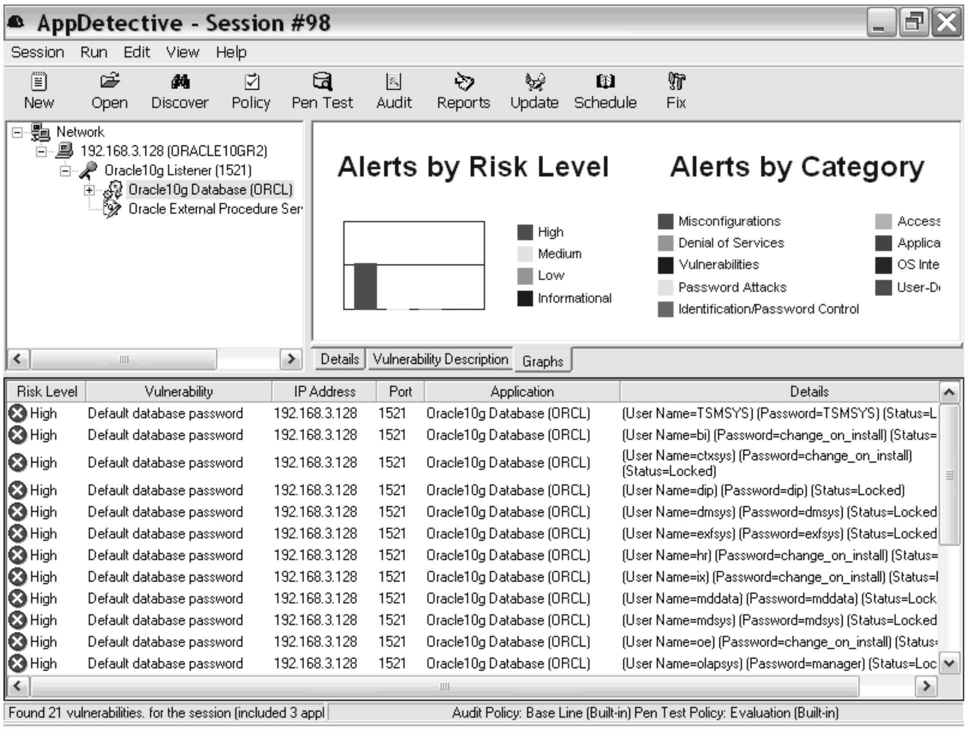
The database vulnerability scanning marketplace can be a bit confusing. There are a small group of intensely focused vendors who do little or nothing but database security, and then there is a larger group of fringe players who specialize in scanning other computer systems for vulnerabilities, but have taken note of the quickly growing market for database security and have added a small number of database vulnerability checks to their products in order to make marketing claims of support for securing databases. Be leery of vendors that do not have a keen focus on database security, as their products likely offer very little meaningful coverage and may give you a false sense of reassurance that your databases have been secured, when in fact only a small subset of potential issues and vulnerabilities have been checked. Also, be sure to ask any vulnerability scanning vendor about their update process and update frequency. Like anti-virus tools, a vulnerability scanner is only as good as its last update. Vendors who update once or twice a year or with even less frequency are not working diligently to protect their customers. For example, Oracle releases a critical security patch every quarter called a Critical Patch Update (CPU). At the bare minimum, a commercial scanner needs to be updated with each CPU release to check for the myriad of vulnerabilities that have been announced and corrected. Purchasing a database vulnerability scanner for enterprise-wide use can be an expensive proposition. Be sure to choose a vendor that is committed to delivering the value that you have paid for.

The focused vendors/products in the database vulnerability scanning space include:

- Application Security: DbProtect and AppDetectivePro
- IPLocks: IPLocks Vulnerability Assessment, IPLocks Database Security & Compliance
- Next Generation Security Software: NGSQuirreL

Here is a sample of the output you can expect from running a commercial database vulnerability scanner configured to scan for Oracle default passwords. Figure 4.1 is from Application Security's AppDetectivePro product.

Figure 4.1 AppDetectivePro


AppDetective - Session #98
Session Run Edit View Help
New Open Discover Policy Pen Test Audit Reports Update Schedule Fix
Alerts by Risk Level
High
Medium
Low
Informational
Alerts by Category
Misconfigurations
Denial of Services
Vulnerabilities
Password Attacks
Identification/Password Control
Access
Applica
OS Inte
User-D
Details
Vulnerability Description
Graphs

```
Network
  192.168.3.128 (ORACLE10GR2)
    Oracle10g Listener (1521)
      Oracle10g Database (ORCL)
      Oracle External Procedure Ser
```

Risk Level	Vulnerability	IP Address	Port	Application	Details
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=TSMSYS) (Password=TSMSYS) (Status=L
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=bi) (Password=change_on_install) (Status=
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=ctxsys) (Password=change_on_install) (Status=Locked)
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=dip) (Password=dip) (Status=Locked)
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=dmsys) (Password=dmsys) (Status=Locked
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=exfsys) (Password=exfsys) (Status=Locked
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=hr) (Password=change_on_install) (Status=
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=ix) (Password=change_on_install) (Status=l
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=mddata) (Password=mddata) (Status=Lock
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=mdsys) (Password=mdsys) (Status=Locked
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=oe) (Password=change_on_install) (Status=
High	Default database password	192.168.3.128	1521	Oracle10g Database (ORCL)	(User Name=olapsys) (Password=manager) (Status=Loc

Found 21 vulnerabilities. for the session (included 3 appl   Audit Policy: Base Line (Built-in) Pen Test Policy: Evaluation (Built-in)

Summary

As far as Oracle database security issues go, default passwords are without any doubt at the top of the list of the most powerful and easy-to-exploit vulnerabilities. Therefore, they should be at the top of your list of items to verify and fix. Effectively cleaning up default accounts requires that you understand the functionality utilized by your database and applications. With a list of the features you do and don't use, you can begin the process by removing any unnecessary accounts from your system, completely eliminating any risk those accounts may present. You can then move to secure the accounts you do need, either by changing their password to a strong one that is extremely difficult to guess, or by setting an impossible password so that the account can never log in.

In environments with more than a handful of databases, it is important to establish an automated means for locating and reporting on default accounts with default passwords. There are several options available, from scripts that you build yourself to free scripts for download, to complete database vulnerability scanning products you can purchase. These password scans really need to be run with some regularity, ideally once a month on every database.

Locking down default accounts is easier than it sounds, but it is a task that can and should be completed on any and every database within your organization. By putting in the effort to remove or secure these accounts, you will have eliminated the lowest hanging fruit in the database security realm and made real measurable progress towards your data security goals.

Solutions Fast Track

The Role of Oracle Default Accounts from 9i to 10g

☑ Oracle is typically installed with a number of features and packages, each of which generally require one or more default accounts.

☑ Each default account in an Oracle database has a default password associated with it. These default passwords are well known and have been published on the Internet.

☑ Default accounts often have powerful permissions in the database, the kind of permissions an attacker would love to access. Combine this with the ease of exploiting default passwords to gain access to a database and the widespread existence of default passwords on production databases, and you have the most critical database security issue that there is.

Lock Accounts and Expire Default Passwords

- ☑ The method Oracle has selected for securing default accounts is to lock the accounts so they cannot log in and expire the passwords so that they must be changed on their next use.
- ☑ This method provides security, but is not the best approach. Attempts to log in to a locked account reveal the existence of that account and default password in the system. Inexperienced DBAs may unlock default accounts in order to be helpful, leaving a gaping hole in the security of the system.

Configure Strong Passwords

- ☑ For those default accounts that are in use and must be allowed to log in to the database, it is critical to configure them with a strong password.
- ☑ Password strength is measured in terms of password length and password complexity. Strong passwords are never dictionary words, dates, names of places, or names of sports teams.
- ☑ Strong passwords include a mix of letters, numbers, and special characters. One way to create strong passwords that can be memorized is to start with a simple phrase, then substitute some letters for numbers or special characters. For example, take the phrase "hard to guess password" and make it the strong password "h@rdt0gu3s5pa5sw0rd."

Unlock Accounts and Configure Impossible Passwords

- ☑ The best way to secure default accounts that own important data or functionality but do not need to log in, is to set an impossible password and unlock the account.
- ☑ Impossible passwords are those passwords that can never be used to log in to the database. Set impossible passwords by telling Oracle to use a value for the password as follows:

```
ALTER USER scott IDENTIFIED BY VALUES 'impossible
  password';
```

- ☑ Unlocking the accounts has the additional security benefit of ambiguity. Oracle will not confirm the existence of an unlocked account when the wrong password is provided in an attempt to log in. Instead, it will give an ambiguous message stating that the username/password is invalid.

Automating the Process of Identifying Default Accounts

- ☑ When you have more than two or three databases to deal with, identifying default accounts manually becomes an overwhelming and non-productive task.
- ☑ Automated tools are available for scanning Oracle databases for default accounts. You can either write your own scripts, download a free tool, or purchase a commercial database vulnerability scanner.
- ☑ Whatever route you choose, it is a best practice to scan all of your Oracle databases for default accounts and passwords on regular basis, ideally once a month. Once defaults are identified, you should move to correct the situation as quickly as possible.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How many default account and password combinations exist for Oracle databases?

A: There is actually some debate on that topic and the answer depends on if you count only Oracle-provided defaults, or add in those of well-known third-party vendors. Oracle on their own has had nearly 700 combinations over the years. If you add in just the most widely used vendors, that number doubles to about 1400 default account/password combinations.

Q: Why does Oracle come with so many default accounts?

A: Oracle has implemented their database and applications in a modular fasion. Each module generally requires an account (or schema) to hold its data and often to run its functions. For the convienience of their customers, Oracle has chosen to install a set of commonly used features by default when the database is installed.

Q: Does Oracle ever change their default passwords?

A: Yes, but not with any regularity or clear purpose. Sometimes a default account will have its password changed with a new release of the database. These passwords are always immediately published though, so the change does nothing to enhance security, in fact if it does anything it harms security. Think about it. If a default password gets changed, you need to update your process for finding those new defaults. If you don't know about a change or forget to update, you could find yourself in a situation where a default password is in place when you are sure there are none because your monthly scans come back clean.

Q: Does Oracle consider default accounts and passwords to be a security risk?

A: Yes. Oracle has acknowledged the issue and have taken steps to address the problem. First, they began to lock and expire accounts at install. Then they took the step of releasing a free-to-download default password scanner. Unfortunately, Oracle's efforts thus far have not been effective in eliminating the problem, so additional tools and processes are neccessary, even with the latest database releases.

Q: Some of these default accounts have minimal permissions. Do I really need to secure them?

A: Yes! Any access is a problem and can lead to a wider breach. Later on in the book we will discuss other vulnerabilities such as privilege escalation that allow a user with no more then PUBLIC permissions to take complete control (become DBA) over an un-patched database.

Q: What does it mean to lock an account?

A: Oracle provides a means to stop an account from logging into the database. This is called locking an account. When a user attempts to log in to Oracle with a locked account, an error message is displayed explaining that the user cannot log in because their account has been locked.

Q: What does it mean to expire a password?

A: Oracle provides a means to force users to periodically change their passwords. When a password reaches a determined age, that password can be expired. When a user logs in to Oracle using an expired password, the database will force the user to change their password before granting access to the database. Passwords can also be expired immediately on creation, forcing a user to reset their password upon their first log in to the database.

Q: How do I configure a new password for an account in Oracle?

A: There are a couple of ways, including both Graphical User Interface (GUI) and command line options. Passwords can be changed in Oracle Enterprise Manager or Oracle Internet Directory GUIs. Passwords can also be changed at the command line (using SQL Plus for example) with the following query:

```
ALTER USER scott IDENTIFIED BY new_password;
```

Q: What are best practices for minimum password length and complexity?

A: Minimum password length should be set to 8 characters. For complexity, each password should contain at least 2 digits and 2 special characters (e.g., @, $, %, &).

Q: What is an impossible password and how do I configure it?

A: An impossible password is one that can never be used to log in to the database. Impossible passwords can generally only be configured from the command line. Use the following query:

```
ALTER USER scott IDENTIFIED BY VALUES 'impossible_password';
```

Q: Where can I get free tools to scan my databases for default accounts and passwords?

A: Oracle offers a default password scanner on Metalink, Pete Finnigan offers one on his Web site (www.petefinnigan.com), and Alex Kornburst offers one on his website (www.red-database-security.com).

Q: What should I look for in a commercial product to scan for default passwords?

A: Make sure the product is updated regularly, at least with every database release or patch. Make sure the product has a complete listing of default account and password combinations (600+). Make sure the product can run multiple password scans in parallel and then roll up the results into high level aggregate reports.

Q: How often should I scan my databases for default passwords?

A: At least four times each year, but ideally once per month. Using an automated tool makes password scanning simple and quick.

Chapter 5

PUBLIC Privileges

Solutions in this chapter:

- The PUBLIC Group
- Default Privileges on Sensitive Functions
- Privileges You Should Never Grant to PUBLIC

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Oracle databases come built in with an interesting entity called PUBLIC. PUBLIC is neither a Role nor a User, although in many ways it is managed like both. Every user in the database is a member of PUBLIC by default, and there is nothing that can be done to change that. PUBLIC is a permanent fixture in Oracle, it cannot be removed by any means. PUBLIC is, in fact, a unique construct, it is a user group, and it is the only user group that exists. It is likely that at some point, Oracle intended to add a feature called user groups, but that feature was never realized beyond the group PUBLIC. Regardless of how it is implemented in the database, managing permissions to PUBLIC is a critical step in securing your Oracle database. Any permissions granted to PUBLIC are granted to everyone in the database, therefore you must use great care in doling out PUBLIC permissions, particularly when it comes to sensitive functions or powerful system permissions. In this chapter, we will review the role the PUBLIC plays in the database and discuss which default permissions need to be revoked and which you should never grant. After reading this chapter, you should be able to create a simple policy to guide your decisions about granting permissions to PUBLIC going forward.

The PUBLIC Group

In Oracle's terms, PUBLIC is everyone with access to a database. It is a User Group where every possible user is in the group and cannot be removed. PUBLIC is an entity that can be granted any privilege and assigned any role. All database users automatically inherit all the rights assigned to PUBLIC. It's a simple concept designed for convenience. Want to make some data public? Grant rights on it to PUBLIC. Got a stored procedure that you want anyone to be able to run? Grant execute rights to PUBLIC. It is a great time saving tool, eliminating the need to assign permissions individually, or to every role. Granting permissions to PUBLIC is easy to do, and often the easiest way to get a system working, however, the security implications of using the PUBLIC group to assign permissions are significant. PUBLIC should be treated with care and used sparingly. Far too many systems have been compromised by the lowest privileged users, all by exploiting access they were granted via PUBLIC.

Use roles instead of granting permissions to PUBLIC. Roles have the advantage of selective membership, where only members of a role gain access to the permissions assigned to that role. Roles allow you to grant access only as needed, whereas granting to PUBLIC leads to an increased risk that a user will gain unauthorized privileges.

This is not to say that PUBLIC is bad or that no permissions should ever be granted to PUBLIC. In some cases, PUBLIC grants make sense; many are required for the proper operation of the database. You cannot just log in to Oracle and revoke all permissions from PUBLIC, but you can selectively remove the permissions that are powerful, risky, and unnecessary. We will discuss which permissions to revoke a little later in this chapter.

Here are some important characteristics of PUBLIC:

- PUBLIC is neither a role nor a user, it is a user group, the only one in an Oracle database.
- PUBLIC is built-in to the database and cannot be removed.
- Every user is a member of the PUBLIC group, there is no way to remove anyone.
- PUBLIC can be granted any role, object permission, or system permission.
- PUBLIC cannot own any schema objects.
- Links and synonyms can be "owned" by PUBLIC (public links and public synonyms)
- PUBLIC is granted access to SELECT from any tables or views prefixed with USER or ALL.

WARNING

If you have granted any DML related system permissions to PUBLIC, you will need to be careful when removing them. Revoking any of these system permissions from PUBLIC has a cascading effect, revoking permissions from every user. For example, if you have granted SELECT_ANY_TABLE to PUBLIC and then you revoke it; all users will lose all SELECT permissions. You will need to grant access back to roles or users before data can be accessed again.

The Big Picture: Oracle Privileges and Roles

Before we get into the details of how the clean up and manage PUBLIC permissions, we would like to offer a little refresher on how roles and privileges work in Oracle databases. If you are familiar with the basics of roles and understand the differences between object privileges and system privileges, you can skip ahead to the next section.

Oracle Privileges

Oracle privileges control the rights to see, use or modify items in the database. "Items" is a term used broadly here, covering schema objects and data. There are two types of privileges in Oracle, system privileges and object privileges. System privileges are rights that are not tied to any specific object or schema. Object privileges are just the opposite, those that are directly related to a specific object or schema.

Here is a listing of just a few object and system privileges:

System Privilege	Object Privilege
GRANT ANY PRIVILEGE	GRANT
ALTER ANY ROLE	ALTER
ALTER DATABASE	SELECT
AUDIT SYSTEM	INSERT
CREATE ANY PROCEDURE	UPDATE
CREATE TABLE	DELETE
DROP USER	INDEX
EXECUTE ANY PROCEDURE	EXECUTE
SELECT ANY TABLE	AUDIT

Information about system privileges is most easily accessed via the DBA_SYS_PRIVS view.

```
SQL> describe dba_sys_privs;

Name                   Null?                Type
---------------------- -------------------- --------------------
GRANTEE                NOT NULL             VARCHAR2(30)
PRIVILEGE              NOT NULL             VARCHAR2(40)
ADMIN_OPTION                                VARCHAR2(3)
```

This view makes things very simple.

- GRANTEE is the user or role that has been granted a system privilege, such as SYS or SCOTT.
- PRIVILEGE is the name of the privilege that has been granted, such as ALTER USER or DROP DATABASE.
- ADMIN_OPTION is a flag that tells Oracle if the grantee is allowed to grant this privilege to other users. The admin option is very powerful and important, setting this flag not only grants the privilege, but it grants the right to grant the privilege to other users. System privileges granted with the admin option (to non-DBAs) should be carefully reviewed and removed where possible.

In order to determine what system privileges have been granted to a user or role, you can run a simple select on dba_sys_privs:

```
SQL> select * from dba_sys_privs where GRANTEE = 'SCOTT';

GRANTEE                        PRIVILEGE                                ADM
------------------------------ ---------------------------------------- ---
SCOTT                          UNLIMITED TABLESPACE                     NO
```

Object privileges are most easily accessed from the DBA_TAB_PRIVS view.

```
SQL> describe dba_tab_privs;

Name                    Null?                Type
----------------------- -------------------- --------------------
GRANTEE                 NOT NULL             VARCHAR2(30)
OWNER                   NOT NULL             VARCHAR2(30)
TABLE_NAME              NOT NULL             VARCHAR2(30)
GRANTOR                 NOT NULL             VARCHAR2(30)
PRIVILEGE               NOT NULL             VARCHAR2(40)
GRANTABLE                                    VARCHAR2(3)
HIERARCHY                                    VARCHAR2(3)
```

Object privileges are a little more complex, but still manageable.

- GRANTEE is the user or role that has been granted the object privilege, such as SYSTEM or DBSNMP.

- OWNER is the schema owner of the object on which the privilege is being granted, such as MDSYS or CTXSYS.
- TABLE_NAME is the object name in question, such as dba_tab_privs or dba_sys_privs. Note that while this field is called TABLE_NAME it may refer to other objects as well, such as views, stored procedures, and functions.
- GRANTOR is the user that granted the privilege to the grantee, such as SYS or SYSTEM.
- PRIVILEGE is the privilege that has been granted, such as SELECT or EXECUTE.
- GRANTABLE is similar to ADMIN_OPTION from dba_sys_privs. It is a flag that indicates the if grantee has the right to grant this privilege to other users. This is one of those sensitive items that should be carefully reviewed and removed if possible.
- HIERARCHY applies to SELECT privileges on an object view only. This flag indicates if permissions have been granted on all subviews within the object view hierarchy or only on the view referenced in TABLE_NAME.

In order to determine what object privileges have been granted to a user or role, you can run a simple select on dba_sys_privs:

```
SQL> select OWNER, TABLE_NAME, PRIVILEGE, GRANTABLE from dba_tab_privs where
GRANTEE = 'HR';

OWNER                TABLE_NAME             PRIVILEGE               GRANTABLE
-------------------- ---------------------- ----------------------- ---------
SYS                  DBMS_STATS             EXECUTE                 NO
```

Tools & Traps…

SQL92_SECURITY

SQL92_SECURITY is an important Oracle configuration setting that most folks know very little about. From the name of the configuration setting, it is clear that there is some impact on security, but exactly what that impact is and why it is important is a mystery to most. The change in the way the database operates when SQL92_SECURITY is set to TRUE is fairly simple, but the reasons why it is important are more subtle.

When SQL92_SECURITY is enabled (set to TRUE), Oracle requires users to have SELECT privileges on a table or view in order to run UPDATE or DELETE statements that include a WHERE clause on that table or view. Put another way, the SQL92_SECURITY parameter requires that users have both UPDATE/DELETE and SELECT privileges on a table or view in order to run queries that include a WHERE clause.

Things may become clearer with an example. Consider a table named *hrapp.employees* containing two columns, *employee_name* and *salary*. AARON is a database user that has been granted privileges to SELECT and UPDATE the *employees* table. JOSH is a database user that has only been granted privileges to UPDATE the *employees* table.

Both AARON and JOSH run the following query:

```
UPDATE hrapp.employees SET employee_name = 'test' WHERE salary > 100000;
```

With SQL92_SECURITY disabled, this query will run properly for both AARON and JOSH. No errors will be returned. However, with SQL92_SECURITY set to TRUE, this query will run properly for AARON but will fail with an error for JOSH.

The impetus behind the implementation of this security configuration is to stop data leakage. In the example above, AARON has rights to see all the data in the *employee* table. If he wants to see the salary data, he can simply SELECT them. JOSH on the other hand does not have rights to see the data in the *employees* table. If he wants to see the salaries, he must find a way to escalate his privileges and get around the fact that he is not granted SELECT on that table. Without SQL92_SECURITY, there is a loophole that would allow JOSH to "see" the salary data in the *employees* table. JOSH could use the following query to determine the salary of any employee stored in the system:

Continued

```
BEGIN
set transaction read only;
DELETE FROM hrapp.employees WHERE employee_name='JSMITH' AND SALARY >
100000;
rollback;
END;
```

Running this query repeatedly while changing the values for *employee_name* and *salary* will give JOSH all the information he wants, without making any changes to the database (each transaction is immediately rolled back).

By simply setting SQL92_SECURITY = TRUE in init.ora, this type of attack will be blocked. SQL92_SECURITY should be enabled on every one of your Oracle databases.

Oracle Roles

We briefly touched on Roles way back in Chapter 1 during the discussion about the history of security features in Oracle. Roles have evolved over time to become a simple means for managing privileges for a group of users all at once. Essentially, you can grant privileges to a role the same way that you would to a user. Instead of having to grant privileges to each user individually, (which quickly becomes a management nightmare) you take care of it all in one place, and then grant users the role. Oracle comes built-in with several pre-defined roles, such as DBA, CONNECT and RESOURCE. You can also create your own custom roles. A user can be a member of more then one role, and roles can even be members of other roles.

Granting and revoking complex sets of privileges is error prone and difficult to manage. As users join and leave your organization, or even change duties in your organization, the job of correctly managing the privileges is tricky. In addition, if the set of privileges ever changes slightly, applying the new change to all accounts is problematic.

A better way to deal with managing these privileges is to create roles that represent the duties a typical user may need. Privileges are then granted only to the roles, never to a specific user. The roles are then granted to the users that need them. Then when a user leaves a job or takes on a new duty, the only change that needs to occur is for the role granted to the user to be changed. In addition, if the privileges ever change, they are changed on the role and automatically take effect for all users with the role.

It is even possible to grant roles to PUBLIC, but that is a really bad idea. The whole reason we are creating and using roles is to avoid grants to PUBLIC!

There are a few key views that you will need to familiarize yourself with in order to effectively manage roles in Oracle, dba_roles and dba_role_privs. dba_roles provides a listing of the roles in the system, while dba_role_privs lists membership in each role.

```
SQL> describe dba_roles;

Name                   Null?                Type
---------------------- -------------------- --------------------
ROLE                   NOT NULL             VARCHAR2(30)
PASSWORD_REQUIRED                           VARCHAR2(8)
```

This is about as simple as it gets.

- ROLE is the name of a role that exists in the system, such as JAVA_ADMIN or DBA.
- PASSWORD_REQUIRED is a flag to indicate if the role is password protected. This feature allows you to assign a role to a user, but then to require that user to input the correct password before setting (activating) the role.

```
SQL> describe dba_role_privs;

Name                   Null?                Type
---------------------- -------------------- --------------------
GRANTEE                                     VARCHAR2(30)
GRANTED_ROLE           NOT NULL             VARCHAR2(30)
ADMIN_OPTION                                VARCHAR2(3)
DEFAULT_ROLE                                VARCHAR2(3)
```

This view is a little more complex, but still quite manageable.

- GRANTEE is the user or role that has been granted a role, such as SCOTT or OE.
- GRANTED_ROLE is the name of the role that has been granted, such as CONNECT or RESOURCE.
- ADMIN_OPTION is a flag that tells Oracle if the grantee is allowed to grant this role to other users. The admin option is very powerful and impor-

tant, setting this flag not only grants the role, but it grants the right to grant the role to other users.

- DEFAULT_ROLE is a flag that tells Oracle if this role should be enabled for the grantee automatically by default. When this is not set, the user must explicitly set their role using the SET ROLE command.

Selecting from these views will give you a list of the roles in your system (there are 33 default roles in Oracle10g release 2) and will tell you which users and roles are members of each role. For example, to determine which users are members of the DBA role, run the following query:

```
SQL> select GRANTEE, ADMIN_OPTION, DEFAULT_ROLE from dba_role_privs where
GRANTED_ROLE = 'DBA';

GRANTEE                        ADM       DEF
------------------------------ --------- ---------
SYS                            YES       YES
SYSMAN                         NO        YES
SYSTEM                         YES       YES
```

TIP

Wondering which privileges have been assigned to a particular role? Where privileges are concerned, roles are treated exactly like users. You can get a list of system privileges by selecting from dba_sys_privs and a list of object privileges by selecting from dba_tab_privs. It is definitely worth a look, particularly at the system privileges. You may find that some roles are far more powerful then you had expected.

Roles Granted to PUBLIC

When it comes to roles, PUBLIC is treated just like any other user or role for that matter. This means that PUBLIC can be granted membership in any role. While it is easy and fully supported to grant roles to PUBLIC, we strongly recommend against ever doing it. In a default installation of Oracle 10g Release 2, you will find that PUBLIC has no role membership.

```
SQL> select * from dba_role_privs where GRANTEE = 'PUBLIC';

no rows selected
```

This is exactly what you are hoping for: no PUBLIC roles. If you do find that a role has been granted to PUBLIC, use the views described in this chapter to determine what privileges are being given out. In general, you can find a better way than granting to PUBLIC. Secure systems will always grant privileges through roles and will always assign roles either to other roles or directly to the users who have a business need for the privileges carried by the role.

Default Privileges on Sensitive Functions

By default, PUBLIC is granted a whole bunch of object privileges, more then 21,000 of them in 10g Release 2 on Windows. It is not practical nor is it necessary to review all of these privileges; however, hidden among the masses are a handful of powerful privileges that grant access to sensitive functions in the database. The next step in your security process is to remove these privileges. It should be noted that no system privileges or roles are granted to PUBLIC by default and it makes sense to keep things that way!

WARNING

Do not be surprised if you revoke some default privileges from PUBLIC only to find that those privileges were replaced when you applied a patch set or CPU. Every DBA we know is too frustrated about it to discuss why they think this occurs with us. Let's just call it one of those quirks. However, this really drives home the need for periodic review and assessment of the security controls and settings you have put in place in your database. The "set it and forget it" mentality may work in infomercials for handy cooking appliances, but it definitely does not work in the world of Oracle security.

The following are functions on which PUBLIC has been granted permissions by default. Our recommendation is that you revoke EXECUTE privileges on each of these from PUBLIC in all of your databases. Unfortunately, some applications and even database features require PUBLIC access to these functions in order to operate.

The only viable solution here is to test any changes thoroughly before deploying them into your production environment.

DBMS_RANDOM

DBMS_RANDOM is a built-in package in the SYS schema. It allows a user to generate random numbers. Random number generators are often used to create keys for encrypting and decrypting data. The problems with DBMS_RANDOM all revolve around encryption.

The issue is DBMS_RANDOM does not really serve up random numbers. What it tries to do is generate pseudo-random numbers (not truly random, but very close), but it actually does a poor job at it. With the right set of analysis tools, an attacker can make educated guesses about past and future output values from the package. Because of this, granting PUBLIC access to DBMS_RANDOM in environments where the function is used in cryptographic key generation could lead to compromise of the encrypted data. An attacker that is able to guess the "random" output could use that information to reconstruct the encryption keys protecting sensitive data. It would then be a trivial matter to take the hacked key and use it to decipher the encrypted data.

In Oracle 11g, DBMS_RANDOM will be deprecated in favor of a better quality pseudo-random number generator in the DBMS_CRYPTO package.

UTL_FILE

Oracle provides a set of utility functions designed to allow the database to interact with the outside world. These utility functions are grouped into Oracle packages based on the interaction they allow with elements outside the database. The privilege to execute these packages of utility functions is granted to PUBLIC by default. While the utility packages are very useful and bring a whole dimension of flexibility to database developers, the functions are powerful and can be abused by an attacker. Oracle recognizes the risk and recommends revoking the PUBLIC privileges on these packages; however, they continue to grant the privileges by default. Oracle will also restore the PUBLIC permissions on these packages during the application of a patch to the database. It is almost as if they are trying to poke holes in your carefully crafted security implementation.

UTL_FILE is undoubtedly the most powerful of the utility packages. It provides direct access to read and write operating system files on the database server. This package is in demand by the hacker world, as it can be used to take control over the database server itself, and thus, control over the database and all the data within. The UTL_FILE can also be used to corrupt or delete data or to destabilize the database or database server by overwriting key executable files. With this kind of potential for disaster, it's no wonder that even Oracle recommends revoking PUBLIC rights

Are You 0wned?

Using UTL_FILE to Take Control over the Database Server

The UTL_FILE package allows an Oracle database to interact with operating system files on the database server. Functions are provided to both read and write to the OS, all with the privileges of the Oracle software owner account (usually oracle on *nix systems and Local System on Windows). As we have seen before, giving out unrestricted (PUBLIC) OS access with the Oracle software owner account is a really bad idea.

Both reading and writing OS files pose some type of risk. Reading files can result in the disclosure of configuration information, SYSDBA password hashes and sensitive information stored in files rather than in the database. An attacker may read the database data files, then use another utility package to send that data to a remote server. Database log files can also be a target for attack, as these files will contain information about changes made in the database, including before and after data values. This kind of information is often all an attacker is looking for: a few credit card numbers, salary information, and company financial data. If your system is processing this kind of information, the database logs likely contain a significant amount of private information.

The ability for PUBLIC to write to the OS is a far bigger risk then reading the OS poses. Writing to the OS can allow an attacker to take control over the OS and the database, cause a denial of service attack against the OS and the database and even erase the audit and log files that could otherwise be used to detect the attack and track down the attacker.

To take control over the OS and database, an attacker can use the UTL_FILE to create or modify key files on the OS. A simple attack involves creating an .rhosts file, thereby granting rlogin access to the OS. The attacker can then connect directly to the OS and execute any commands. This effectively grants full control over the system to the attacker. The UTL_FILE can be used to create a glogin.sql file. Once this file is in place, it will be automatically run every time sqlplus is used to connect to the database locally. That may not happen every day, but when it does, it's almost always with a DBA account.

Continued

The attacker could use this script to create their own DBA level account in the database or grant the account they have already compromised powerful system privileges such as SELECT_ANY_TABLE.

Denial of service attacks are also simple to execute. Replace the Oracle executables with empty files and it will not take long before the database crashes and refuses to come back online. While this may not be as devastating as an attack that steals sensitive data, it can cause havoc and significant monetary damage. Imagine an online retail site that goes down during the holiday shopping rush or a stock trading system that stops functioning during heavy trading hours. The implications of a DoS attack can be severe.

There are two ways to eliminate the risks posed by the UTL_FILE. Far and away, the best choice is to revoke all PUBLIC privileges on the package, then restrict access to only those roles that require access to the OS in order to function. In most Oracle database systems, it turns out that this package is not used at all and no rights need to be granted on it. In systems where UTL_FILE is used, it should be restricted so only a small set of directories can be accessed. This is done with the UTL_FILE_DIR parameter in the init.ora file. Set this parameter with a list of directories to which you require access via the database. Never set UTL_FILE_DIR to *, as this indicates unrestricted access to any file accessible to the Oracle software owner account.

The functions of the UTL_FILE package mimic the standard C file functions such as fopen and fclose. These functions allow a database user to read and write from the operating system through PL/SQL. This feature extends Oracle's functionality greatly and is very useful. The files that can be accessed by the UTL_FILE package depends on the security context Oracle runs under. On *nix systems, the Oracle process typically runs under the context of an account called Oracle. The UTL_FILE package has the privileges to read or write any files the Oracle account has permissions to read or write. On Windows systems, the Oracle process is typically run under the context of the LocalSystem account, which has unlimited access on the database server.

Which directories the UTL_FILE package can access can be restricted by setting the UTL_FILE_DIR parameter (stored in the init.ora configuration file). Setting the UTL_FILE_DIR parameter to * tells Oracle to impose no restrictions on the directories that can be read and written by the UTL_FILE package. Setting the UTL_FILE_DIR parameter to a specific directory or list of directories, will instruct Oracle to restrict the UTL_FILE package by allowing access to only those specified directories.

The UTL_FILE functions are quite easy to use:

```
declare
file_name UTL_FILE.FILE_TYPE;
begin
file_name := UTL_FILE.FOPEN ('/temp_dir', 'utl_file_test.txt', 'W');
UTL_FILE.PUT_LINE (file_name, 'Testing UTL_FILE', TRUE);
UTL_FILE.FCLOSE (file_name);
end;
```

This code will use the UTL_FILE package to open the file /temp_dir/utl_file_test.txt for writing (if the file does not exist, it will create it). It will then write the line "Testing UTL_FILE' into the file, then closes the file. Note this will only work if the UTL_FILE_DIR parameter is set to allow writing to the /temp_dir folder.

Granting PUBLIC execute permissions on the UTL_FILE package is extremely dangerous, particularly when UTL_FILE_DIR is unrestricted. With this access, a nefarious user can

- read the remote login password file
- write commands to the autoexec.bat or glogin.sql file
- create a .rhost file
- corrupt a log or control file

Remember, UTL_FILE runs with the operating system permissions of the Oracle software owner. It can read, write or even delete any files that Oracle can access, even the Oracle data files. An adversary can write a program that intentionally corrupts a data file. Corrupt the SYSTEM tablespace and the database ceases to exist. Remove the active logs or archived log files and recovery of the system beyond the destroyed log file becomes impossible.

In Oracle9i, Oracle allowed DIRECTORY objects to be used with the UTL_FILE package instead of specifying a value for UTL_FILE_FIR. Using a DIRECTORY object, a program can specify a location to read/write instead of restricting things to the hard coded values in init.ora. Defining a DIRECTORY value requires the CREATE ANY DIRECTORY system privilege. This is clearly a very powerful privilege and should not be granted to users unless absolutely necessary. The syntax to define a DIRECTORY object is as follows:

```
CREATE OR REPLACE DIRECTORY temp_dir as '/temp_dir';
```

In order to use a DIRECTORY, a user must be explicitly granted rights on it. Those rights can be issued on a granular level, allowing an administrator to control read and write access independently. The syntax used here is fairly unique:

```
GRANT READ ON DIRECTORY temp_dir to 'SCOTT';
GRANT WRITE ON DIRECTORY temp_dir to 'SCOTT';
```

DIRECTORY objects offer distinct advantages in both security and flexibility over the UTL_FILE_DIR parameter. From a security perspective, DIRECTORY objects eliminate the risk caused by setting UTL_FILE_DIR to *. Greater flexibility is achieved by allowing new directories to be defined and used without restarting the database, while any change to the UTL_FILE_DIR parameter requires a database restart to take effect. Unless you are running Oracle8i or older, we recommend removing the UTL_FILE_DIR parameter from init.ora and using DIRECTORY objects instead.

UTL_HTTP

UTL_HTTP is another powerful utility package that has execute rights granted to PUBLIC by default. Like UTL_FILE, this utility is owned by SYS and is used to give the database access to the outside world. UTL_HTTP is designed to allow the database to send and receive HTTP and HTTPS requests and responses.

The UTL_HTTP package itself does not pose a direct threat to the database, nor can it be used to compromise the host operating system. At the same time, it is a real security threat and something you need to take action on and revoke PUBLIC access. With the power to freely communicate across the network using HTTP, or better yet to privately communicate using SSL protected HTTPS, an attacker may be able to send data to himself without notice often regardless of firewalls or data classification / extrusion detection systems.

Consider a situation where an Oracle database is acting as the backend data repository for an on-line retail store. An attacker has found a SQL Injection vulnerability in the web application and is able to send queries directly to the backend database. While the attacker knows the queries are reaching the database and are likely being processed, the web application is not returning any results to the browser. Without some means of collecting the results of the injected queries, there is little value in exploiting the vulnerability.

However, with the UTL_HTTP package, the attacker has another means of returning the query results. HTTP or encrypted HTTPS links can be established from the database back to the attackers system. These links can be used to carry any query results, so the sensitive data in the database, credit card data and customer information can be packed up and sent completely bypassing the web application. Firewalls will typically not stop this because it looks like normal web traffic, with a session being initiated from inside the network. Using HTTPS, data classification or extrusion detection devices that monitor the network will also fail here, unable to decipher the encrypted traffic.

UTL_HTTP is an important tool for an attacker and therefore a component to which you must tightly control access. We strongly recommend revoking EXEC privileges on UTL_HTTP from PUBLIC in all your production Oracle database systems along with all non-production systems that contain sensitive data.

UTL_SMTP

Similar to the UTL_HTTP package, UTL_SMTP is used to send SMTP messages (also known as email) to an SMTP server. If you are curious, SMTP stands for Simple Mail Transfer Protocol. This package is also built-in and has EXEC privileges granted to PUBLIC by default. The UTL_SMTP package itself is not dangerous; however, it can be used to send sensitive data across the network as email. An attacker can use this to send themselves query results when they are unable to receive the results through other means, such as a web application with a SQL Injection vulnerability.

Because of the risk that this package will be used by an attacker, we recommend that you revoke EXEC privileges on UTL_SMTP from PUBLIC in all your production Oracle database systems along with all non-production systems that contain sensitive data.

UTL_TCP

UTL_TCP is very similar to both UTL_HTTP and UTL_SMTP. The built-in package has rights granted to PUBLIC by default and is used to send messages across the network from the database, this time using raw TCP packets. Using TCP directly requires more networking know-how than a typical database user has. The attacker would have to manage the protocol on the database side and setup a client that can receive and store the messages being sent across the network. The skills needed to orchestrate this type of attack are very well known in the hacker world. Breaking down network traffic, capturing and inspecting TCP packets, this is simple Hacker 101 stuff.

Just as with the other UTL networking packages, we recommend revoking PUBLIC privileges on UTL_TCP on all production systems along with all non-production systems that contain sensitive data.

Privileges You Should Never Grant to PUBLIC

Some privileges are just too powerful or sensitive to ever grant to PUBLIC. These are privileges that can lead to complete compromise of your database and the data inside it. Furthermore, these are privileges that need to be explicitly granted either by an administrator or during an application install. These privileges are not granted by default.

Before we proceed, let's be clear about the privileges we are not discussing here, which are execute privileges on functions that contain vulnerabilities. By granting PUBLIC privileges to access the vulnerable functionality, you create a security hole. Sometimes the holes are huge, allowing anyone to run SQL commands as a DBA. There have been many vulnerabilities like this reported in Oracle databases over the years, and the flow of new vulnerability reports has certainly not subsided.

While it is true that granting privileges on these vulnerable functions to PUBLIC creates a security issue, the best way to correct the issue is not go revoking privileges all over the place. With dozens of functions to deal with, and few resources out there that provide a complete list, attempting to revoke PUBLIC privileges on all vulnerable Oracle functions would be a monumental task in any organization with more then a small handful of databases. Instead, apply the CPUs and Patch Sets that fix the root of the problems. For more detail on patching, see Chapter 6.

System Privileges

We can make this very easy. Don't ever give out system privileges to PUBLIC! There is just no good reason for it. If you need to grant system privileges, create a role and grant the privilege to the role, then assign the role to just those users who require the privilege. Many system privileges lead to direct compromise of your database, while others facilitate indirect compromise by using a number of attack vectors.

There are a few system privileges that are more of an issue than all the others are. A list of these system privileges follows along with a description for each. Please do not take this limited list as an invitation to grant other system privileges to PUBLIC, rather look at it as a priority list of items to revoke first.

ANY System Privileges

Several system privileges come with the word ANY in them. These are DBA level privileges that allow the grantee to manipulate any object of the type granted. The absolute worst privileges to grant to PUBLIC are those with ANY in the name.

WARNING

Nearly every ANY system privilege allows the grantee to become a DBA. There are many attacks out there that have been described by various Oracle security experts showing how to exploit various ANY privileges to take control of the system. Do not fall into this trap! Make sure you never grant an ANY system privilege to PUBLIC in any of your databases.

- GRANT ANY PRIVILEGE: This is essentially equal to granting DBA rights on the database. A user with access to GRANT ANY PRIVILEGE can grant themselves any privilege. This allows them to access any data in the database by simply granting themselves the right to access it. Similar privileges include:
  1. GRANT ANY OBJECT PRIVILEGE
  2. GRANT ANY ROLE
- EXECUTE ANY PROCEDURE: This privilege indirectly gives the grantee full DBA rights. There are several definer rights procedures in the SYS schema that can be used to run arbitrary SQL statements. An attacker simply needs to run one of these procedures and ask it to process a SQL statement that escalates their privileges to DBA. Here are a few examples:

  ```
  EXEC SYS.DBMS_STREAMS_RPC.EXECUTE_STMT ('GRANT DBA TO ATTACKER');
  EXEC SYS.DBMS_REPACT_SQL_UTIL.DO_SQL('GRANT DBA TO ATTACKER', TRUE);
  EXEC SYS.LTADM.EXECSQL('GRANT DBA TO ATTACKER');
  ```

  The following privileges have similar affect, as they can be used to create a procedure in a DBA's schema that run arbitrary SQL:

1. CREATE ANY PROCEDURE
2. ALTER ANY PROCEDURE

- SELECT ANY TABLE: This privilege gives an attacker access to read any data in the database. It does not lead to DBA privileges, but who needs to be a DBA when they can read any data they want? SELECT ANY TABLE gives an attacker the ability to query the system tables for configuration information, and then go after the user tables to collect and take home any sensitive data in the system. Similar privileges that allow a user to access any data in the database include:
  - UPDATE ANY TABLE
  - DELETE ANY TABLE
  - INSERT ANY TABLE
- CREATE ANY VIEW: This privilege gives an attacker the ability to set up an attack that will likely lead to DBA privileges. The attack involves creating a procedure that attempts to grant the attacker DBA rights, then creating a view in a DBA's schema (such as SYSTEM) which runs the procedure. The last step is the lucky part, convincing a DBA to access the view, which is really not very tricky. If you are interested in the details, check out the Oracle Hacker's Handbook by David Litchfield. The following similar privilege may also allow a user to become a DBA:
  - ALTER ANY VIEW
  - CREATE ANY TRIGGER: This privilege can also be used to become a DBA; the attack is simple. Start by creating a procedure that grants you DBA rights. Be sure it's an invoker rights procedure. Identify a user with DBA rights (there are several default users that will do). Pick one that has a table that PUBLIC has some kind of privilege on (SELECT, INSERT, UPDATE, DELETE, any will do). Create a TRIGGER on the table you have identified for an action PUBLIC has rights to perform. Inside the trigger, call your procedure to grant DBA. Finally, access the object you placed the TRIGGER on and enjoy your DBA rights. This works because the TRIGGER will be run as its schema owner, who is a DBA. This attack has also been described in detail by Litchfield in the Oracle Hackers Handbook. The following similar privilege can be used to the same affect:
  - ALTER ANY TRIGGER

- CREATE ANY SYNONYM: Allowing users to create SYNONYMS can lead to all sorts of havoc. By creating SYNONYMS for commonly used objects, an attacker can make the database behave in unexpected ways. This privilege was used by Alex Kornbrust in his demonstration of Oracle root kits at Black Hat in 2005. By creating synonyms for important system views, he was able to create and effectively hide a DBA level user. The attacks can be simple. For example, create a PUBLIC SYNONYM for DBA_USERS that lists all users in the real DBA_USERS VIEW except the attacker. Anyone accessing DBA_USERS would assume they got directly to the VIEW, never realizing they have been tricked into accepting a partial view of the real data.
- CREATE ANY DIRECTORY: As discussed earlier in this chapter, DIRECTORY objects are powerful items, particularly when combined with access to packages such as UTL_FILE and DBMS_LOB. By creating DIRECTORY objects, a user with the proper privileges can take advantage of packages that allow access to the OS to read, change or delete arbitrary files. As we have seen previously, this can quickly lead to DBA rights in the database or admin access to the OS.
- CREATE ANY JOB: This privilege allows a user to easily become a DBA. The attack is trivial: create a JOB (scheduled task) that grants the attacker the DBA role and place it in any DBA?s schema. Schedule the job to run, then sit back and wait for DBA privileges to come to you.

Individual System Privileges

There are several system privileges that do not include the word ANY yet are powerful enough that you should never even consider granting them to PUBLIC. These privileges may give an attacker the ability to become a DBA, to disable the database or to disrupt important business processes. The following are the most sensitive individual system privileges that, if found, you should revoke from PUBLIC right away.

- ALTER SYSTEM: This privilege allows a user to set configuration parameters and perform system management tasks on an Oracle database. This can include anything from disabling auditing (allowing an attacker to hide their tracks) to dumping a datafile (allowing an attacker to take raw data from the database). There is no reason to grant ALTER SYSTEM to PUBLIC.

- CREATE PROCEDURE: This privilege itself is not dangerous, however, when combined with other permissions (such as CREATE ANY TRIGGER or CREATE ANY VIEW), can lead to full system compromise. It does not make sense to grant CREATE PROCEDURE to PUBLIC. Instead grant the privilege to a role and then grant access to the role to only the users that have a legitimate business need.
- AUDIT ANY / AUDIT SYSTEM: These privileges allow a user to take control over the Oracle audit system. By doing so, an attacker can disable auditing in order to effectively hide their tracks after an attack. PUBLIC never needs the right to modify the databases auditing subsystem.
- ALTER DATABASE: This privilege allows a user to change the configuration of a database, controlling items such as datafiles and even allowing starting and stopping of the database. This privilege can be used to execute a simple DoS attack by stopping the database, and may even be used to steal data by directing Oracle to use a datafile that is accessible by UTL_FILE or other packages that can access the OS.
- ALTER USER: This privilege allows any user to become a DBA. Simply identify an existing user that has DBA rights. In a typical Oracle database, there are several default accounts with DBA rights set up but locked. Using ALTER USER, the attacker can unlock the DBA account or change the password. They now have credentials to log in to the database as a DBA. ALTER USER should never be granted to anyone other than a DBA.
- DROP USER: This privilege can be used to execute a DoS attack against the database. By dropping key users and their associated schema, applications and the database itself will likely become unstable and crash. DROP USER should never be granted to anyone other than a DBA.
- ALTER PROFILE: This privilege allows a user to change password control settings for the system. An attacker can use ALTER PROFILE to indirectly gain DBA rights. By using ALTER PROFILE to disable the account lockout feature (FAILED_LOGIN_ATTEMPTS = UNLIMITED), an attacker would make it possible to execute a brute force password guessing attack against any DBA account in the database. It may take time, but with automated tools, a DBA account will have its password cracked.

- EXPORT FULL DATABASE / IMPORT FULL DATABASE: These privileges are designed to allow an entire database to be imported or exported, usually for backup and recovery purposes. To do this job, access to all the data in the database is required. Granting this privilege to PUBLIC is similar to granting SELECT ANY TABLE and UPDATE ANY TABLE. Needless to say, this should never be done.
- SYSDBA / SYSOPER: These privileges grant DBA rights, however they cannot be granted to PUBLIC, nor can they be granted to a role.

Tools & Traps…

O7_DICTIONARY_ACCESSIBILITY

O7_DICTIONARY_ACCESSIBILITY is a configuration parameter intended to facilitate backwards compatibility with the way Oracle7 databases permits access to the data dictionary. This parameter has serious security implications however many folks are not aware of its existence or function. In practice, it is not uncommon to see O7_DICTIONARY_ACCESSIBILITY set to TRUE on modern production Oracle databases.

When enabled, O7_DICTIONARY_ACCESSIBILITY (often called the dictionary protection mechanism) instructs Oracle to apply System Privileges that have been granted with an ANY clause (SELECT_ANY_TABLE, EXECUTE_ANY_PROCEDURE, Etc.) to the data dictionary or SYS schema. In Oracle7, this was not an option, ANY system privileges allowed a user to directly access (and sometimes modify) the data dictionary. Oracle realized that this represents a security hole, effectively allowing non-DBA users to administer the database.

With the release of Oracle8i, the database's default behavior was changed in order to plug up the security hole, however in order to facilitate a smooth transition from older versions of the database to 8i, Oracle introduced the O7_DICTIONARY_ACCESSIBILITY option to allow users to revert back to the old behavior. Through the years, this parameter has never been dropped, and remains in place today, even in Oracle11g, even though almost nobody needs to use it.

Continued

The O7_DICTIONARY_ACCESSIBILITY parameter is set in init.ora. In order to disable this compatibility mode and enable the security feature, set the parameter to FALSE. Unless it is required for the proper operation of a legacy application, O7_DICTIONARY_ACCESSIBILITY should be FALSE on all of your Oracle databases.

Object Privileges in the SYS Schema

In a perfect world, there would never be any reason to grant PUBLIC any privileges on objects in the SYS schema. However, in the real world, things are not that simple. By default, PUBLIC is granted some privileges on objects in the SYS schema. Actually, it's more then some privileges, in Oracle 10gR2 PUBLIC is granted over 18,000 privileges in the SYS schema. These privileges have been carefully considered by Oracle and should pose no security risk (that is, as long as you are fully patched). However, as we have seen with the case of UTL_FILE, UTL_HTTP, and the other dangerous functions mentioned earlier in the chapter, sometimes Oracle misses something.

Despite the large number of default grants, the majority of objects in the SYS schema are not safe for public use. It makes no sense to add any PUBLIC grants on SYS objects. If you know you have done this already, review the privileges you have granted and revoke them if possible. To get a list of PUBLIC privileges on SYS objects, run the following (and expect a lot of output):

```
select PRIVILEGE, TABLE_NAME, GRANTABLE from dba_tab_privs where OWNER = 'SYS' and
GRANTEE = 'PUBLIC';
```

Use Common Sense

When granting any privilege to PUBLIC, think twice. If there is any way to limit the rights to a group of users by granting the privilege to a role, take advantage of it. Practice the principal of least privilege and grant privileges only to those users that need it. If you are granting rights on functionality owned by a powerful user that runs with definer rights (AUTHID DEFINER), consider the risk of a determined attacker finding a SQL Injection hole in the code. Limit your exposure by limiting PUBLIC access.

Summary

The PUBLIC user group is a unique structure in Oracle databases. PUBLIC is bound to every user like a permanent default role, acting as a vehicle to distribute privileges to every user in the database at once. Oracle grants a large number of privileges to PUBLIC by default, some of which are powerful and should be revoked to limit the risk of a successful attack against the data in the system. Additional privileges can be granted to PUBLIC at the discretion of the DBA, however, it is a poor practice to make PUBLIC grants. It's important that powerful system privileges are not granted to PUBLIC, as they often lead to DBA rights.

Review the privileges granted to PUBLIC on a regular basis. Make sure rights on sensitive packages such as UTL_FILE have been revoked. Ensure that no system privileges have been granted and review rights on any objects in the SYS schema. It's helpful to create a baseline list of acceptable PUBLIC privileges from a properly locked down fresh install and then compare that list to deployed systems. This gives a quick view of differences and may help to highlight unknown and potentially unnecessary PUBLIC grants.

Best practices for security dictates that access should be restricted to those with a need to know. Handing out privileges to PUBLIC is a quick way to violate this best practice. By locking down default permissions and then making future grants through roles, you can eliminate a major avenue of vulnerability exploitation and make a significant impact on the security of your sensitive data.

Solutions Fast Track

The PUBLIC Group

☑ PUBLIC is a User Group that is built in to Oracle and cannot be removed. All users are members of PUBLIC by default. There is no way to revoke PUBLIC membership.

☑ PUBLIC can be granted any privilege or role. Any grants to PUBLIC take immediate effect on all users in the database.

☑ PUBLIC access can be used by an attacker to gain DBA rights and take control of the database. It is critical to review and secure any PUBLIC grants on sensitive or powerful functions.

Default Privileges on Sensitive Functions

- ☑ Oracle grants many privileges to PUBLIC by default. Most are safe, but there are a few that are too powerful and should be removed.
- ☑ EXEC rights on the UTL_FILE package are granted to PUBLIC by default. This package gives the database access to read and write files on the OS. PUBLIC access to this function can lead to compromise of the database via the host OS.
- ☑ EXEC rights on UTL_HTTP, UTL_SMTP, and UTL_TCP are granted to PUBLIC by default. These packages allow the database to communicate across the network using standard protocols. They can be used by an attacker to send themselves sensitive data, bypassing existing security measures.

Privileges You Should Never Grant to PUBLIC

- ☑ PUBLIC should never be granted a system privilege with an ANY clause. These privileges affect every schema in the database SID and almost always have the potential to lead to DBA rights.
- ☑ Many individual system privileges should never be granted to PUBLIC. Privileges that either directly or indirectly lead to DBA rights are common. The best practice is never to grant any system privilege to PUBLIC.
- ☑ Use common sense when granting new privileges to PUBLIC. Never grant rights on objects in the SYS schema. Carefully consider granting rights on objects in a schema owned by a DBA or other powerful user. In general, grant privileges through roles rather then through PUBLIC.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Is PUBLIC a user or a role or what?

A: PUBLIC is a unique construct called a user group. Everyone is a member of this group by default. There is no capability to remove the group or administer its membership.

Q: Is it possible to create objects that are owned by PUBLIC?

A: No. PUBLIC cannot own objects. PUBLIC does have its own schema, though. It becomes active when you log in as SYSOPER.

Q: What is the difference between system privileges and object privileges?

A: System privileges are rights that are not tied to any specific object or schema. Object privileges are just the opposite; those, which are directly related to a specific object or schema.

Q: Are there default PUBLIC privileges that need to be removed in order to secure my Oracle database?

A: Yes. Revoke execute rights on the utility packages UTL_FILE, UTL_HTTP, UTL_SMTP, UTL_TCP and on the weak random number generator DBMS_RANDOM.

Q: Are there other privileges that I should avoid granting to PUBLIC?

A: Yes. In general, avoid PUBLIC grants entirely. Never grant system privileges to PUBLIC, particularly those with ANY in the privilege name.

Chapter 6

Software Updates

Solutions in this chapter:

- Understanding Oracle's Patching Philosophy
- Examining a Critical Patch Update
- Installing a Critical Patch Update
- Evaluating Security Alerts

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Every software application has bugs to fix and new features to implement. Some of those bug fixes impact your security posture, as do some of those new features. The determination of whether or not to update your Oracle system must be made carefully, comparing the reward of securing vulnerabilities against the risk of causing new problems and the expense of implementation. It is a difficult decision to make and Oracle has made great strides in recent years in helping you make it.

Oracle has several kinds of updates for your software. First and most notably are Critical Patch Updates (CPUs). This will encompass most of the discussion in this chapter. Oracle also provides *security alerts* for emergency security patching, *patchsets*, which include a multitude of security and non-security patches, and major *releases* which contain patches, as well as new features. They are all important from a security perspective and each will play a part in your security lockdown plan.

While Oracle has significantly improved their ability to meet enterprise scheduling and resource requirements to minimize the cost of security, their security posture still has room for improvement to ensure their customers' systems are as bulletproof as possible. Every CPU released to date has fixed many major problems, but there are still many left to fix. There also have been numerous instances where CPUs did not patch all of the problems their advisories claimed to patch. In this chapter, we will discuss ways to mitigate this to minimize your exposure. These include keeping yourself abreast of publicly known security issues and using an activity monitoring system to notify you if an attacker is attempting to breach your system.

Understanding Oracle's Patching Philosophy

January 2005 brought the dawn of a new patching philosophy at Oracle. Security comes at a cost and the cost was very high prior to then. Responding to heavy demand from their customers, Oracle began releasing patches on a schedule. Now, CPUs are released quarterly and each one includes patches to dozens, sometimes over 100, of the most severe vulnerabilities. The obvious benefit of CPUs is that they plug security holes, but receiving them on a well-defined schedule has some hidden benefits, too. Imagine for a moment that Oracle released a security fix every time they discovered a problem or when a friendly hacker informed them of one. Database administrators would be inundated with fixes, sometimes more than one a day. So, customers started asking for a reliable schedule around which they could plan.

Before the launch of CPUs, Oracle released patches containing only security-related fixes as *security alerts*. They did this as needed for the most severe problems, but not on a regular schedule. They also released patchsets periodically to fix both functional and security problems. Both of these mechanisms are still in use today. However, Oracle releases security alerts much less often now, because CPUs are the primary means to deliver these types of fixes.

Patchsets are still the primary means to get a large number of fixes at once. These are significant updates to Oracle software, which contain the fixes already released as CPUs, plus more security fixes and many non-security improvements as well. In the past, customers used patchsets as their primary means of updating their software to reduce their exposure to attacks. Because they contained non-security work as well, this increased the potential of the patchset to break a critical function within Oracle beyond the potential of the security work. Customers often spent a lot of time and resources on a long test cycle to reduce this risk. This was not only expensive, it also increased their exposure to threats. While those tests were taking place, hackers were busy investigating the now public vulnerabilities. Because of this, customers asked for patches that limited code changes to only those that improved security.

Security

Oracle focuses their vulnerability fix development around maximizing security and reducing its cost. To address the former, they carefully assess their list of internally known vulnerabilities and prioritize what they consider to be the most dangerous ones to be fixed first. An external group of Oracle security experts who are aware of vulnerabilities not known to the general public meets periodically to help them with this effort. Once the list has been formed, Oracle engineers get to work on fixing the bugs, first on their active code line, which is for a future patchset or release, and then on the next CPU's code line.

If some vulnerability is particularly dangerous, they'll also back-port the code to previous patch sets and get a fix out to the public as quickly as possible as a Security Alert. Let's say a hacker let loose a 0-day exploit, which gives an attacker database administrator (DBA) privileges and can be executed remotely without a password. This qualifies as an emergency and you can expect Oracle to be very responsive. However, be aware that there are dozens of undisclosed vulnerabilities that have yet to be fixed. In July 2005, Red Database Security published details of flaws for six bugs in Oracle Reports and Oracle Forms. Their CEO Alexander Kornbrust said, "I decided to publish these vulnerabilities because it is possible to mitigate the risk of these vulnerabilities by using the workarounds provided in the advisories." They released these advisories in response to the July 2005 CPU lacking fixes for what they considered critical issues, some of which had been outstanding for over a year-and-a-half.

CPUs cover a number of different categories of hacks that can be particularly damaging to the database service or to the business that owns it. Anything that allows a user access to data to which he or she is unauthorized to access is a form of a *privilege escalation attack*. Such an attack could give a read-only user the ability to update bank account balances, it could give a Web user the ability to query for large volumes of credit card data, or it could give complete database administration permissions to an unauthenticated user. You will often see advisories describe how a user can gain SYS rights by running a specially crafted command. For example, an announcement from Application Security, Inc. (www.appsecinc.com/resources/alerts/oracle/2005-03.html) describes how one particular problem, which was fixed in the April 2005 CPU, allows the OBJECT_TYPE parameter of DBMS_METADATA package to be used to execute code with SYS privileges:

```
Team SHATTER Security Alert

Multiple SQL Injection vulnerabilities in DBMS_METADATA package

April 18, 2005

To determine if you are vulnerable to this attack, download AppDetective from
http://www.appsecinc.com/products/appdetective/oracle

Risk level: High

Credits: This vulnerability was discovered and researched by Esteban Martínez Fayó
of Argeniss for Application Security, Inc.

Affected Versions:
Oracle Database Server versions 9i and 10g

Details:
The OBJECT_TYPE parameter -- used in various procedures of the DBMS_METADATA
package -- is vulnerable to SQL injection. Although this package executes with the
privileges of the calling user, it internally uses another package that executes
```

```
the injected SQL with the privileges of the SYS user thereby allowing an attacker
to gain DBA privileges. By default PUBLIC has EXECUTE privilege on DBMS_METADATA.

Impact:
Any low privileged database user can execute functions with DBA privileges. Users
with privileges to create or modify a function can inject a user-defined function in
the vulnerable procedure and thus execute SQL statements with DBA privileges.

Workaround:
Revoke Execute privilege on the vulnerable package.

Vendor Status:
Vendor was contacted and a patch was released.

Fix:
Apply Oracle Critical Patch Update April 2005 available at
http://metalink.oracle.com

Links:
Oracle Security Alert:
http://www.oracle.com/technology/deploy/security/pdf/cpuapr2005.pdf

Advanced SQL Injection in Oracle databases presentation:
http://www.argeniss.com/research.html
```

Another type of attack commonly fixed in CPUs is called a Denial of Service (DoS) attack. DoS attacks reduce the ability of a service to perform its function. A DoS may slow a database or completely stop it. Some of these attacks can crash an Oracle server process or the listener service. There are many other different kinds of problems that CPUs will fix. Privilege escalations and DoS attacks are just two examples.

Notes from the Underground…

Room for Improvement

Oracle’s security posture has several fundamental flaws. First, they reduce the priority of fixes for vulnerabilities that require a privileged user such as “system” to exploit, often deferring these to the next patchset or release rather than including them in a CPU. Their logic is that such users have authority to do anything they want anyway, so the risk of gaining additional access is low. Let’s imagine that a skilled hacker exploited a vulnerability such as the one in the above advisory, to gain rights equivalent to those of a DBA. It’s not hard to imagine that the same hacker could exploit a second problem which requires such rights and allows him or her to execute arbitrary code. This code could do anything, not just as a DBA, but also as the “Oracle” user on the host operating system (OS). This makes destroying evidence (such as error logs) of the attack possible. If this code could instead be constructed to give access to DataGuard, the user could be able to alter data that was previously protected even from a DBA. Database security researchers, including your humble authors, disagree with this policy.

Additionally, Oracle currently has a backlog of over 100 security problems to fix, many of which were given to them by the database security industry, a.k.a. “white hat” hackers, and some of these have been around for over eighteen18 months. As mentioned above, Oracle’s backlog contained problems of the same age as of July 2005, meaning they have not improved their fix rate since then. Although the known Oracle problems are not published outside of the security community, they do exist. A “black hat” hacker might discover the same problem on his or her own and try to exploit it. We would very much like to see Oracle increase the rate at which they improve the security of their software.

Considering a public security advisory is a signal to black hats to try to attack the same module in other ways (see the “Are You 0wned?” sidebar below), Oracle should be particularly careful to fix everything in a given area of the system at the same time. Historically, they have not been successful at this. For example, Hypertext Transfer Protocol (HTTP) server problems, usually in server modules, have been in eight of the nine CPUs from January 2005 through January 2007. The XML Database (XDB) feature has appeared in seven of them, and the “spatial” component has been in six of them. Oracle should do a thorough security-oriented code review of any module that has a

Continued

security flaw. Recurring problems in the same area of software is a clear indicator that their patching effort has room for improvement.

The Common Vulnerability Scoring System (CVSS), which we will discuss later in this chapter, is a quantitative way to measure the risk of security problems in software. This is a worldwide standard developed by leading security analysts from a wide range of companies. Part of the standard is for the vendor, in this case Oracle, to indicate the risk to the confidentiality, integrity, and availability of data. CVSS defines three values for those fields: "None," "Partial," and "Complete," indicating no risk of loss, risk of partial loss, or risk of complete loss, respectively. Oracle has interpreted the standard to mean that these values should apply to the entire host (i.e., all software running on it). If Oracle is the primary purpose of the machine, then this makes sense. Gaining privileged access to the host in this situation implies privileged access to Oracle but nothing else of significance. On machines that run other applications though, unless Oracle is running as "root" on UNIX or "Administrator" on Microsoft Windows, it is highly unlikely that a breach in Oracle will affect the other applications in a significant way. Because of this, since they began using the CVSS standard in the October 2006 CPU, Oracle has never given themselves a value of "Complete" in these fields for the database server product even if the user gains SYS rights by exploiting the vulnerability. They instead use a value of "Partial+" to indicate a wide area of effect across the database system. When calculating their CVSS base score, however, they still use the numerical equivalent of "Partial." In my opinion, this biases the score to be lower than is representative of the true threat to the database, and they should change their practices to use the standard value "Complete" rather than the non-standard "Partial+."

Cost

The cost of applying a patch includes staff, training, tools, equipment, and time. Database administrators will need to understand their current patch level, the benefits of the next patch, and how to apply it. The former two can be automated with vulnerability assessment tools specifically designed for databases. The process of patching is automated with a tool from Oracle called OPatch, which we'll discuss in more detail later in this chapter. In most environments, the DBA will want to use a test environment to practice the patching procedure and to verify that it will not cause any adverse effects in a production environment. DBAs will want to spend some time assessing each patch 1) to determine if the bugs that have been fixed apply, 2) to what extent they fix a relevant problem, and 3) to determine the risk in not patching. For example, a fix to Real Application Clusters (RAC) probably will not matter to a database that does not use clustering. Also, bug fixes do not always completely fix a problem; it may not be worth the effort of applying a partial fix first and then a more complete fix later. A DBA may also opt to not patch his or her system if the risk of not patching is acceptably low compared with the risk of changing a critical production environment. See the "Examining a Critical Patch Update" solution in this chapter for a review of how to assess patches.

Platforms

Another part of Oracle's patching philosophy involves platforms (i.e., the operating systems on which Oracle runs). Linux is a very popular one and it also happens to be the platform on which Oracle developers write their software. They will release on this platform first. Microsoft Windows has been gaining in popularity as an Oracle platform and it typically uses the same x86 or x86_64 platform as most Linux installations, so that is usually next. Oracle has a separate team to port code to other platforms at a later date. Customers don't always download every platform for every CPU, so Oracle does not create a patch set for several platforms unless a customer requests it. If you need a CPU on a particular platform that does not already have one available, submit a request to Oracle and they will port it. According to www.oracle.com/security/critical-patch-update.html, "Oracle will no longer issue proactive patches for platform-version combinations that have had fewer than 10 downloads in the prior CPU period. Beginning with CPUJul2007, Oracle will deliver patches for these historically inactive platform-version combinations to customers on an on-demand basis. Oracle will continue to include fixes for those vulnerabilities in the main product code line and current patchset(s)." Commonly used platforms aside from Linux and Windows include Solaris, AIX, HP-UX, and others. CPUs for uncommon platforms such as VMS and Tru64 are available upon request.

Examining a CPU

CPUS are the primary mechanism for patching Oracle software in between releases. They contain security patches and any prerequisite work required for those patches. Each CPU comes with an advisory describing its contents, availability, and installation procedures. It briefly describes each vulnerability fixed in the CPU, giving you just enough background to make decisions about applying it. Include a step in your security lockdown plan to assess CPUs quarterly. This section discusses the decisions you can make about each CPU and how to go about making them.

There are several actions you can take based on the information provided in the CPU's advisory. First and foremost is to apply the CPU in its entirety. As of the July 2007 CPU, you also can choose to apply parts of the CPU without applying all of it. Additionally, workarounds are often available, such as disabling components, revoking permission to use them from a particular group of users, or putting up firewalls to block certain kinds of traffic. Lastly, you can opt to take no action and wait for the next CPU to reassess the situation. Which action you should choose depends on several factors. Among these are the *risk matrix* included in the advisory, the existence of a publicly available exploit, the applicability of the threat to your environment, the effort of evaluating and applying the CPU, and the net risk to your business of plugging security holes versus altering your system with a patch. Since every CPU contains, as the name suggests, critical patches, you should very strongly consider applying every one of them in their entirety.

WARNING

CPUs contain fixes to severe problems in Oracle that have a very real potential to be exploited. Your security lockdown plan should include a review of the latest quarterly CPU and your default action should be to apply it fully. Every CPU starts with the sentence "Oracle strongly recommends applying the patches as soon as possible," and we couldn't agree more. Get buy-in from senior management on this policy. Any deviation from it should require sign-off from top-level personnel.

Assessing the Risk Matrix

Each CPU advisory comes with a set of risk matrices indicating the relative danger of each security flaw newly addressed in the patch. There is a separate risk matrix for each Oracle product (e.g., Oracle Database, Oracle Application Server). The advisory does not detail any means to exploit the problems on an unpatched system, but it does let you know in which component of the product the flaw resides. It also lets you know to what package or privilege a rogue user needs access in order to successfully execute a hack.

For Oracle Database and some other Oracle products, the CPU is cumulative, meaning that fixes from previous CPUs are included in newer ones. Flaws first addressed in a prior CPU are detailed in that CPU's advisory. So, to get a complete picture of the patches that will be applied by the current CPU, concatenate the matrices from every CPU that has not already been applied to your database. Applying the aggregation of several CPUs increases the theoretical risk of patching—the more changes you make to your database system, the higher the risk of failure. Consider this when assessing how and if you will apply a CPU.

A risk matrix contains the following columns:

Column	Description
Vuln#	A unique identifier, apparently for the fix and not for the vulnerability. Historically, when a vulnerability has been fixed on more than one occasion, it gets a different Vuln# in each CPU in which it appears. If the flaw is not in the core software of the product, but rather, is included from another Oracle product, this value will be in italics.
Component	The component of the product in which the flaw resides.
Protocol	The communications protocol used to exploit the vulnerability. For Oracle Database, this is often "Oracle NET."
Package and/or Privilege Required	The package or privilege required to execute a potential exploit (e.g., "Create Session" or "Execute on MDSYS.MD").
Remote Exploit without Auth.?	A "Yes" or "No" indicating the possibility of exploiting the problem remotely without first authenticating to Oracle. A "Yes" in this column generally is dangerous, but the severity really depends on the particular flaw. This means that someone who is not a user on the system can access it.
CVSS Version 1.0 Risk	A set of seven values adhering to version 1.0 of the Common Vulnerability Scoring System (CVSS). See below for details of these fields.

Continued

Column	Description
Earliest Supported Release Affected	The earliest supported release in which the flaw appeared. Any release since the one declared in this column may be affected, and any supported releases before it are not. For example, if a bug didn't exist in 9i (possibly because the corresponding feature didn't exist then) but it is new in 10g, this column will say "10g." For the purposes of this column, 9i, 9iR2, 10g, 10gR2, and 11g are considered different releases. Unsupported releases are not tested, so whether they are affected or not is unknown.
Last Affected Patch set (per Supported Release)	For each supported release, this column lists the most recent patch set that has the flaw. Newer patch sets are not affected by it.

The Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) is a "common language to discuss vulnerability severity and impact"[1] coordinated by the Forum for Incident Response and Security Teams (FIRST). It defines quantifiable criteria so that you can measure risk. The CVSS columns in the risk matrix are as follows:

CVSS Column	Description
Base Score	The quantifiable risk of the vulnerability on a scale of 0 to 10 without regard to any temporal or environmental factors. An example of a temporal factor is the appearance of an exploit on the Internet. The exploit may not have existed before the announcement of the CPU, but hackers suddenly developed one once they knew there was an opening to be exploited. An example of an environmental factor may be that an Lightweight Directory Access Protocol (LDAP) vulnerability doesn't apply to your system because it doesn't use that technology.
Access Vector	"Local" or "Remote" indicating an attack must be initiated from the local machine or that it can be initiated remotely.

Continued

CVSS Column	Description
Access Complexity	Indicates the difficulty in exploiting an attack once Oracle has been accessed. Possible values are "High" and "Low." Attacks over the network that take advantage of, as an example, a blank listener password can be executed at will. In this case, the access complexity is "Low." In another example, there may be a vulnerability in a non-standard component. Since the component must first be installed or activated, the access complexity in this case is "High."
Authentication	"Required" or "Not Required" indicating whether an attacker needs to authenticate to Oracle before executing the attack
Confidentiality	Indicates the extent to which confidential information may be disclosed to an unauthorized user. In terms of data, this generally refers to "read" or "select" access. "None" indicates no potential confidentiality loss. "Partial" indicates some information may be disclosed (e.g., a small number of tables). "Partial+" is a non-standard value which Oracle uses to indicate "a wide range of resources (e.g., all database tables.")[2] "Complete" in Oracle's interpretation means total visibility of all data on the operating system, such as a root user would have. Note the difference between "authorized" here and "authenticated" above. Someone who is logged in is authenticated, but he or she may not be authorized to view certain tables or records. A "privilege escalation" attack could potentially give such access.
Integrity	Indicates the extent to which data may be modified by an unauthorized user. In terms of data, this generally refers to "update/insert/delete" access. "None" indicates no potential modifications by an unauthorized user. "Partial," "Partial+," and "Complete" are analogous to their definitions for "Confidentiality."

Continued

CVSS Column	Description
Availability	Indicates the extent to which an unauthorized user can alter the system's availability. This can mean crashing the database, slowing it to a crawl, or otherwise making it less than fully available. "None" indicates no potential loss of availability due to the vulnerability. "Partial" and "Partial+" refer to loss of database resources. "Complete" refers to the machine and its operating system as a whole. Note that "Partial+" is a value provided by Oracle but is not a part of the CVSS standard.

Acting on Security Advisories

The announcement of a security flaw implies that you need to take two avenues of action. First, update your activity monitoring solution to include the latest attack signatures. It will then inform you if someone attacks your system before you have a chance to take your second action, which is to patch or workaround the problem.

TIP

An activity monitoring solution lowers the risk of attacks by informing you in real-time when they occur. This gives you the ability to take defensive measures such as disabling a particular user account or taking the system offline. Too often, a CPU advisory will announce a fix to a particular problem but it is actually not fixed in its entirety and sometimes not at all. As an example, DB05 in the January 2007 CPU reappeared as DB12 in the July 2007 CPU, because the former completely failed to fix the issue. For that six-month period, the threat to Oracle users was higher because an unpatched attack vector was publicly known. Those who had a monitoring system in place were in a much better position to deal with the situation.

Altering a system takes time and resources and adds risk. When patching, carefully weigh the benefits against the drawbacks and plan your rollout carefully. For example, one consideration is which systems to patch. Systems with highly sensitive data should be patched if the "confidentiality" is anything other than "None." Mission-critical systems should be patched if the combination of threats to confidentiality, integrity, and availability are at more than acceptable levels, and if an exploit is known and easy to execute by someone without credentials. Don't forget development and test systems. Very often their configurations become the production configurations. In some organizations, the test machines are swapped into production as the old production boxes are removed.

Another decision is when to patch. You may want to patch now, wait until some major rollout of an enterprise system completes, or maybe your company policy is to go through this effort only every six months rather than three. Some companies still go with the "patchset only" philosophy in that they only upgrade to the latest patchset and don't use CPUs. Some vulnerabilities can be mitigated by workarounds, such as disabling an unused feature, and some vulnerabilities can be patched without patching others.

CVSS offers a standard way to quantify risks. Combine what you learn from the CPU advisory with industry knowledge of known exploits (and other temporal factors) and your knowledge of your systems. Bias each of these three factors according to your organization's preference, and view them together when making a decision. When you are done, develop a checklist of every system, what action you will take with each, and when that action should occur. In the next section, we will discuss applying the patch including planning a rollout.

Are You 0wned?

If at First You Succeed, Try, Try Again

Database hackers are some of the most skilled software analysts in the world. Most are software developers in their professional lives. Not surprisingly, they are intimately familiar with the process of creating and shipping software products. They know that even in the largest corporations, developers are often lazy or poorly trained in basic security issues. For example, a common problem I've seen in production code is populating a buffer with more data than it can hold. This gives an attacker the ability to write data or even code somewhere it does not belong. Additionally, for the sake of time, people will often skip code reviews or security-related testing.

Practical programming managers will assign a given module of code to as few developers as possible, because this is more efficient than throwing a lot of people at tiny pieces of development. Quality Assurance managers likewise

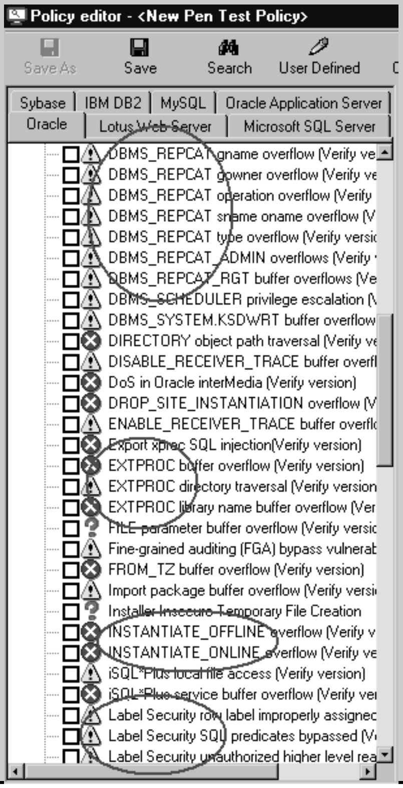
Continued

will assign as few people as possible to test a module. Because there is usually a small group of engineering "owners" of the code, hackers know that if there is a vulnerability in one part of a module, there is likely to be a similar problem elsewhere in the same module. Once they succeed in breaking in via one method, they will spend weeks trying different variations of the same method to ensure every last hole in the vicinity is uncovered. As the hacker learns his way around a group of functions, it will be easier for him or her to find new problems.

As a real world example, let's take a look at the DBMS_REPCAT package used for replication services. In Oracle 9.2.0.4, it had buffer overflows in the *fname*, *gname*, *gowner*, *sname*, and *type* parameters across many of the functions of that package. Likewise, there were patterns of problems in EXTPROC used for extended procedures, and in XDB used for XML processing in various versions and patchsets of Oracle.

Figure 6.1 is from AppDetective Pro from Application Security, Inc. Observe the patterns of the names of the penetration tests. They indicate times when hackers focused their efforts on a particular area of Oracle once they successfully attacked it.

Figure 6.1 New Pen Test Policy



```
DBMS_REPCAT gname overflow (Verify ve
DBMS_REPCAT gowner overflow (Verify ve
DBMS_REPCAT operation overflow (Verify
DBMS_REPCAT sname oname overflow (V
DBMS_REPCAT type overflow (Verify versio
DBMS_REPCAT_ADMIN overflows (Verify
DBMS_REPCAT_RGT buffer overflows (Ve
DBMS_SCHEDULER privilege escalation (V
DBMS_SYSTEM.KSDWRT buffer overflow
DIRECTORY object path traversal (Verify ve
DISABLE_RECEIVER_TRACE buffer overfl
DoS in Oracle interMedia (Verify version)
DROP_SITE_INSTANTIATION overflow (V
ENABLE_RECEIVER_TRACE buffer overflo
Export xprec SQL injection(Verify version)
EXTPROC buffer overflow (Verify version)
EXTPROC directory traversal (Verify version
EXTPROC library name buffer overflow (Ver
FILE parameter buffer overflow (Verify versio
Fine-grained auditing (FGA) bypass vulnerab
FROM_TZ buffer overflow (Verify version)
Import package buffer overflow (Verify versi
Installer Insecure Temporary File Creation
INSTANTIATE_OFFLINE overflow (Verify v
INSTANTIATE_ONLINE overflow (Verify ve
iSQL*Plus local file access (Verify version)
iSQL*Plus service buffer overflow (Verify ver
Label Security row label improperly assignec
Label Security SQL predicates bypassed (V
Label Security unauthorized higher level rea
```

Installing a Critical Patch Update

As with any change, you must plan the implementation of a CPU, test it, and then deploy it. Planning consists of reviewing the CPU as described above, selecting to which systems to apply it, enumerating steps indicating how to apply it to each of those systems, and finally scheduling those steps. Each system has unique characteristics so expect the application steps to vary. Your plan should include several steps for testing to ensure that your data is intact, that the system functions properly, that the system is secure, and that it continues to perform well. Deploying the patch must done carefully to avoid downtime. Include in your plan steps to abort the patch in case something goes wrong, and be sure to test them beforehand.

Database administrators who are familiar with updating their software can safely skim or even skip this section. If your role is not that of someone responsible for updating software, I encourage you to read this section in its entirety. Security auditors, for example, should familiarize themselves with the effort involved here, because it has a direct effect on an organization's ability to keep their software up to date.

Planning

The planning phase of updating to the latest CPU will take a significant amount of time the first time you do it. You have not written down these steps yet and they have not matured after several iterations of use. Over time, planning will become less expensive, because you will reuse the same plan quarter-to-quarter as Oracle releases each CPU.

NOTE

Old patch sets are generally not supported in the newest CPUs. For example, as of the October 2007 CPU, 9.2.0.7 is not supported. Oracle provides maintenance for "de-supported" versions via their Extended Maintenance Support (EMS) plan.

The first part of your plan should contain top-level steps that you perform regardless of how many systems you have. These will generally include "Identify Systems," "Review CPU," "Gather Intelligence," "Choose Actions," and "Prioritize Systems." The first identifies all of the databases that you are responsible for patching. The second and third steps review the CPU advisory and gather intelligence from the security community about the fixes in the update. The output of these steps should include detailed information about the problems that were addressed and about current threats in the wild. In the next step, choose what action(s) you would like to take for each system. Base this on a cost-benefit analysis that includes the cost of patching combined with the net risk of patching versus not patching. Consider using the CVSS standard to calculate the risk of not patching. You will find it easier to define a few relative classes of systems, e.g. "development", "test," and "production," each indicating a group of servers to which you apply relatively the same changes. The template of each class can be tailored as needed for individual systems. Also, consider using subclasses that indicate availability characteristics (e.g., "99.999 percent uptime" or "9-to-5 on weekdays in North America"), because these distinctions will affect how you apply or don't apply each update. Prioritize your systems as a final step before getting into the details of each. If you run out of time or resources, this will ensure that you acted on the most important servers first.

TIP

As of July 2007, you can opt to apply a subset of the patches in a CPU via a new technology called *molecules*. Each molecule represents a particular area of the system that needs to be patched. This gives you the ability to update one area of Oracle with a low risk of affecting a different area. Moving to this new *napply* technology requires you to revert to the last patchset and then apply the latest CPU. See Oracle MetaLink Note 438314.1 entitled "Critical Patch Update - Introduction to Database Napply CPUs" for more details.

Testing and Deploying

Each system or category of systems will need "Backup/Restoration," "Deploy," and "Test" procedures. Before getting into specifics, determine the level of time and resources you feel comfortable spending on each of those three areas. This will help you lay out a plan of attack that fits within your constraints.

Your backup and restoration procedures should include backing up at least your software and data, but possibly your entire filesystem. Include steps to ensure that you can revert to your original state. Some large corporations backup their production system, restore it on identical hardware, and then upgrade that clone. If the upgrade on the clone tests successfully, they put the clone into production rather than the original machine. This makes aborting an upgrade quick, easy, and with low risk.

If you don't have spare hardware available, let's review the core concepts involved in backup and restoration. First, test your backups to ensure they can be restored. You don't want to be in a situation where you need to abort the upgrade and only then discover that your backup didn't work. From experience, you'll be up all night trying to get the system back online by the open of business the next day. Set a firm "Go/No-Go" time when you decide if you need to abort. Make sure it allows enough time for your restoration process to finish by the time you need the application back online.

Read the CPU's README carefully for instructions on how to apply the CPU. These always include running *OPatch*, Oracle's "One-Off Patch Installer," but often include other steps that must be executed carefully and in a specific order. If you have applied any one-off patches since the last CPU (e.g., one from a security alert), after applying the CPU, be sure to test that the fix stills works. It's possible that the CPU unintentionally reverted the patch. OPatch might inform you that the CPU cannot be applied to your environment due to one of your patches. If this is the case, log in to Oracle's Web site and create an *itar* to request assistance. Oracle may re-release the one-off patch with changes from the CPU.

Test the CPU thoroughly. Include a reasonable spread of tests across your application(s) and your DBA activities. Understanding the changes mentioned in the advisory will help you focus your test efforts on affected areas. There's a chance that performance was impacted by the CPU so test that, too.

Also, patch everything on the box that relates to Oracle. Any service listening on a non-localhost port which runs as the Oracle user (usually "oracle"), a user with any access to Oracle files and services, or the administrative user ("root" on UNIX systems; there can be several in addition to "Administrator" on Windows) should be on your lockdown plan. These services, if successfully hacked, can be the gateway to an attack against your Oracle installation.

Patch your test environments before going to production. This gives you an opportunity to test your deployment plan in a relatively safe environment. Your tests should include backing up the system, executing the steps in the patch's README, and ensuring your application still functions properly. Also, test your abort procedures and rerun your tests to ensure the reverted system works.

When you're done with the upgrade and you're ready to put your server back online, go through your filesystem lockdown plan that you created in Chapter 2.

Ensure file permissions are accurate and then update the cryptographic hashes of static files.

Evaluating Security Alerts

A security alert is an emergency update from Oracle to address a particularly harmful exploit or exposed vulnerability. Oracle's Chief Security Officer, Mary Ann Davidson, states "in event of the cyber-equivalent of 'imminent bodily harm,' Oracle will issue an unscheduled security alert and post the patch for immediate download."[3] It is important to keep apprised of security alerts by getting on Oracle's security alert mailing list. Go to www.oracle.com/admin/account to subscribe.

If Oracle releases a security alert, strongly consider applying it. With the advent of CPUs, Oracle will rarely release a security-related one-off patch. People in the database security industry will generally post their own advisories shortly after the alert. Read them carefully to understand the nature of the problem and the effect of the fix on your system.

TIP

Your security procedures should include subscribing to one or more security mailing lists. Having a third party proactively inform you of issues is easier than checking for them yourself. Oracle does an excellent job of informing their customers when a fix is available. If you use a database security product, subscribe to their mailing list as well. Several other public mailing lists are excellent ways to get immediate knowledge of any newly announced exploits that may affect you. These include SecurityFocus's BugTraq (bugtraq@securityfocus.com), Full-Disclosure (full-disclosure@lists.grok.org.uk) and VulnWatch (vulnwatch@vulnwatch.org).

Summary

Patching software is an important part of any security lockdown plan. Oracle offers an automated tool, *opatch*,_to make this easier. They offer regular CPUs so that you can schedule the process of evaluating, planning, testing, and deploying them. Internally, Oracle's processes are geared toward the same goal as yours: increased security at a decreased cost. Every CPU contains critical updates, including those that prevent a hacker from gaining SYS privileges, so be sure to apply each one.

Patching at regular intervals ensures that your systems are protected. Keeping abreast of emergency fixes and determining if they should or should not be applied will ensure that you have the most up-to-date software available for your environment. However, be sure to carefully evaluate each patch comparing the benefit with the risk. Most often, patches work perfectly and completely resolve the problem at hand. However, occasionally patches don't work as well as they could, and it's possible that they can have adverse effects. Test them thoroughly and deploy them only when you are ready.

Your organization's security posture can be greatly improved by applying security patches as soon as possible once they have been released. However, in most large-scale environments, this turnaround time can be weeks or even months. In that time, be sure to use an activity monitoring system that can detect attacks targeted specifically for your database. Use a third-party tool that has frequent updates to their attack signature library to ensure the best coverage.

Solutions Fast Track

Understanding Oracle's Patching Philosophy

- ☑ Oracle focuses on the severity of problems when prioritizing fixes. There is an ongoing debate in the database security community regarding Oracle's definition of severity levels.
- ☑ Oracle CPU's are designed to maximize security while keeping costs down
- ☑ CPU's are delivered quarterly, on approximately the 15th of January, April, July, and October.
- ☑ Use an activity monitoring solution to warn you of attacks in the time between Oracle's launch of the CPU and your application of it. It will also

help you in those cases where the CPU did not fix the problems it claimed to fix.

Examining a CPU

- ☑ When Oracle announces vulnerabilities as part of a CPU advisory, you can patch your systems or you can work around some problems rather than patching them. Each CPU advisory comes with a risk matrix that helps you decide what actions to take.
- ☑ Oracle recommends that you apply every CPU.
- ☑ In addition to the risk matrix, gather your own information about your environments and anything that may be temporal in nature such as the sudden appearance of a worm. Use all of these types of information in your decision-making process.

Installing a Critical Patch Update

- ☑ Read the CPU and information from the database security community when identifying affecting systems and determining how to patch them.
- ☑ Test your system thoroughly including your backup/restoration procedures
- ☑ Patch other applications running on the same machine as Oracle in case they have vulnerabilities that affect Oracle.

Evaluating Security Alerts

- ☑ Oracle will announce a Security Alert only for imminent security threats.
- ☑ Subscribe to security mailing lists to keep abreast of threats.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www. syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How often does Oracle release security patches?

A: Oracle releases *security alerts,* which are patches to particularly dangerous problems, as needed. If a 0-day exploit exists on the Internet, Oracle will generally release a security alert rather than waiting for the next scheduled CPU. CPUs are released quarterly and they include all of the security alerts plus many other patches. They also release patchsets periodically, which include both security and non-security alerts.

Q: Do CPUs account for all of the security problems in Oracle software?

A: No. Patchsets will contain more security fixes than are available in CPUs. You should keep up-to-date with all patch types.

Q: What are some of the problems that have been fixed in Oracle's Database Server recently?

A: Privilege escalation attacks that grant SYS access, DoS attacks that crash Oracle, and buffer overflows that have the potential to run hacker-defined code are common.

Q: My database backs a mission-critical 99.999 percent uptime application. How can I apply a CPU?

A: Every system, even the most highly available ones, has a way to open a maintenance window. Very often these systems use RAC or another clustering technology having several nodes. Nodes can be taken down one at a time for upgrades.

Q: Security Alert #68 contained many fixes. Do security alerts still carry such an impact to my system?

A: No. Security alerts have been almost entirely replaced by scheduled CPUs. Security Alerts will now only be released in case of emergency.

Q: What actions do organizations typically take if their activity monitoring solution indicates an attack?

A: That depends on the nature of the system being attacked. If it's a low priority system, the DBA will generally take it offline. In production environments, it's common to put up a firewall to block the source of the attack or to kill the process and then disable the user who executed the command.

1 www.first.org/cvss/v1/guide.html

2 Oracle Metalink article 394487.1 – Use of Common Vulnerability Scoring System (CVSS) by Oracle.

3 "Introducing Oracle's Quarterly Critical Patch Updates," Mary Ann Davidson, Jan 15, 2005. www.oracle.com/technology/pub/columns/davidson_cpu.html

Chapter 7

Passwords and Password Controls

Solutions in this chapter:

- Configuring Strong Passwords
- Password Controls Using Oracle Profiles
- OS Authentication
- Automated Scanning for Weak Passwords

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Authentication is the gateway to the database, and for the vast majority of Oracle systems, the gatekeeper requires no more than a valid username and password pair to allow anyone to pass. One can argue over the merits of username- and password-based authentication, and can make claims about external authentication mechanisms being better, stronger, faster. However, those arguments will not be made here. Instead, we're going to focus on what Oracle has given us, and what we see practically every production Oracle system using. The native Oracle authentication mechanisms are secure enough for almost all systems, but only when used properly.

Passwords are a critical component of your Oracle security infrastructure. This chapter focuses on establishing a system that ensures that all your passwords are difficult to guess, then configuring protections to thwart password-guessing attacks against your databases.

Configuring Strong Passwords

Keeping unauthorized individuals out of your Oracle databases requires you to ensure that every user has a strong password. Passwords are important. They hold the key to each database, allowing anybody with the right password into the system. Passwords are also a target of attackers and their powerful automated attack tools. There are more password crackers out there than any other kind of hacker tool. Try searching in Google. I got 1.7 million results when I searched on the term "password cracker". The last result on the first page (see Figure 7.1) was particularly interesting.

Figure 7.1 Oracle Password Cracker

Oracle **Password Cracker** (Checker)
Checkpwd - Oracle **password cracker**.
www.red-database-security.com/software/checkpwd.html - 17k - Cached - Similar pages

You don't make it to the front page on a list of nearly 2 million search entries without a lot of clicks. The notorious John the Ripper was just a few entries above this one. The point is that it takes little more than the ability to point and click to download a powerful password-cracking tool. It's not much more difficult to point that tool at a database and start breaking in. These tools are out there and a large number of people are using them. Strong passwords are the first line of defense against these attack tools.

What Makes a Password Weak?

Weak passwords are easy to guess. This includes more than the passwords that are easy for a person to guess, but also those that are easy for a computer to guess. Password crackers are computer programs that are built to guess passwords. Password crackers can work in different ways, but the most common is dictionary-driven, where the tool cycles through a dictionary of passwords, trying each password in the dictionary for each known account (or even every username in a separate dictionary) until it is able to log in. Simply put, if a password is likely to end up in a password cracking tool's dictionary file, then it is a weak password. But how can you tell?

Start with the English dictionary. If a word is in there, it's easy to guess. Next add in cities and sports teams. Add numbers to make up dates, like birthdays or anniversaries. Finally, add simple patterns like 12345 or qwerty. You will find most if not all of these in a typical password cracker dictionary file.

Usernames are also weak passwords. It's very common to see accounts in Oracle databases where the password is the same as the username. This should really be no surprise, if anything this is a trend that Oracle themselves started. The majority of the accounts Oracle includes in the database by default have their password set to their username.

TIP

Oracle takes steps to protect passwords in the system. First, all passwords are stored as a password hash, never in cleartext. Looking at the password hash tells you nothing about the password. Second, Oracle blocks access to the password hashes, storing them in the SYS schema and only displaying them in the database administrator (DBA) views (in Oracle 11g, even the DBA_USERS view does not show passwords).

Usernames, however, are not protected. Anyone with access to the database can get a list of users by selecting from the ALL_USERS view (SELECT granted to PUBLIC by default). This makes it easy to test every account in the database for a password that equals the username, and potentially gain unauthorized access to the system.

Another form of password cracking is called brute-force password guessing. Brute-force is more aggressive than a dictionary attack, primarily focusing on short passwords. A brute-force password cracker takes aim at a certain number of characters (usually no more than 4 or 5 characters) and then guesses every combination of typeable characters of the maximum length or less. This can be a long process. Oracle actually limits the number of typeable characters by converting all passwords to uppercase before hashing (this changes in 11g). In total, there are 68 different characters that can form an Oracle password. To do a brute-force search on four-character passwords, involves searching on all one-character passwords (68 of them), all two-character passwords (4624 of them), all three-character passwords (314,432 of them), and all four-character passwords (21,381,376 of them). That's a lot of passwords, and the number keeps going up exponentially as you add more characters to the password. At or beyond six characters, brute-force password cracking is generally ineffective, as it requires guessing billions of combinations.

It's best to assume that a password that meets any of the following criteria is weak:

- It appears in the English dictionary
- It is the name of a well-known city anywhere in the world
- It is the name of any professional sports team
- It is a calendar date
- It is a simple pattern, such as abcdef, 98765, or jjjjjj
- It is the same as the username
- It is less than six characters long

Who Can Remember a Strong Password?

Actually, it's worse than just remembering one password. You need a different strong password for every system. That's hard, particularly when you want to choose passwords like *wygc?gb!* or *gy7★ui9clor.* What you need is a system that allows you to generate seemingly random strings that actually aren't random at all. It all starts by picking a methodology or technology for choosing your passwords.

Password Management Tools

The best choice for managing a large number of strong passwords is to use a secure password management tool. Earlier we mentioned Password Safe from sourceforge.net (http://sourceforge.net/projects/passwordsafe). Other similar free password managers are Password Corral from Cygnus Productions (www.cygnuspro-ductions.com/freeware/pc.asp) and KeePass from sourceforce.net (http://keepass. sourceforge.net/).

All three will generate strong passwords, and then store them for you in an encrypted file protected by a master pass phrase. With a tool like this, you can have different passwords for each system you use, but only need to remember the password to unlock the password manager. This is ideal, because the passwords for your databases can be randomly generated and impossible to remember. You only need to generate one strong password for yourself. If pass phrases are allowed (long strings with spaces), it is preferable to use a sentence. Pick your favorite line from a movie or part of the chorus of your favorite song, add some punctuation or mix in some uppercase letters and you've got yourself a passphrase. For example, this would make a nice pass phrase "It'S a SiciliaN message. IT means LucA brasi SleepS with ThE fishes."

Password Rule Sets

If you want to go it alone and remember all your passwords, there are ways to use strong passwords and remember them. If you've got a good memory, you can probably use a simple method, like choosing a phrase and picking the first letters of each word as a password. For example: "My daughter Marlee was born in August" becomes MdMwbi8, "I went to school at UMass Amherst" becomes "Iw2saUMA."

If memorizing many phrases and which systems they work with seems like a daunting task, there is another way. Gina Trapini wrote about Password Rule Sets on lifehacker.com. Rule sets make remembering lots of strong passwords fairly easy. The idea is to start with a base password, then customize the base password for each system.

Start with a complex looking but easy-to-type base password, something like iop[]\ (this is easy to type, the keys are all lined up in a row). Next add customization for each system. If you've got a SID named ORCL, create a password like ORiop[]\CL, LCiop[]\RO, or iop[]\CLOR. Even ORCLiop[]\ and iop[]\ORCL are pretty good passwords.

Make a rule for how you will customize the base password, then follow that rule for each system. If we picked ORiop[]\CL in the example above, a SID named HR would get the password Hiop[]\R and a SID named PEOPLESOFT would get the password PEOPLEiop[]\SOFT. In this case, the password rule splits the SID name and then places the base password in the middle. You really only need to remember the rule and the base password.

Passwords For Non-production Systems

It's important to take password management seriously on non-production systems as well as production systems. This goes beyond best practices and beyond a mindset for taking security seriously in all systems regardless of their value. The fact is, many non-production systems contain sensitive data that needs the same level of protection as is needed in a production environment.

The reality, however, is that most organizations completely ignore the security of their test and development databases. This is a big problem, as many of these ignored non-production systems contain a copy of production data or a copy of a production schema and configuration data. This is hacker gold. If you've got test systems that include even a partial copy of the data in your sensitive production systems, you need to secure those systems like production. That means removing default accounts and setting strong passwords to protect against unauthorized access. A hacker is just as happy to collect their booty from a test box as they are to collect it from a live production system. It's the data they are after.

Even if your non-production system only contains a copy of the production schema and perhaps database configuration, this needs to be protected. An attacker that can get this information can then use it as a roadmap to hack into the real production system where the sensitive data lives. It just makes sense, when it comes to protecting access to a database, to treat every one of them like production, and get the passwords and password controls locked down.

Password Controls Using Oracle Profiles

Oracle provides a robust set of security features around passwords and password controls. Options are available to lock accounts after a set number of failed logins, to expire passwords after a set period of time, and to enforce flexible complexity requirements on every new password added to the system. These features are all implemented within Oracle profiles.

The Oracle Profile

A profile allows an administrator to set limits on a user's resource utilization and to manage their password. You can create a number of profiles with different settings and then assign them to groups of users. A profile includes two groups of settings: *kernel limits* and *password limits*.

Kernel limits focus on controlling the resources a user can consume. There are settings for the maximum number of concurrent sessions a user can have, a maximum CPU time for a session and a call, a maximum idle time before a session is disconnected, a maximum read blocks per session and per call, and a whole slew of other resource-related items.

Password limits implement the security controls. Here are settings that control how complex a password must be, how long each password can be used before a change is required, and how many failed login attempts are allowed before an account is automatically locked. These settings enforce your password policies, and are the second line of defense against password crackers.

By default, there is one profile created, called DEFAULT. All users are assigned the DEFAULT profile unless otherwise specified. The DEFAULT profile is installed with all security options disabled, you must enable them on your own or create new profiles to enforce password controls.

Profile information is available in the DBA_PROFILES view.

```
SQL> describe dba_profiles;

Name                     Null?            Type
------------------------ ---------------- -------------------
PROFILE                  NOT NULL         VARCHAR2(30)
RESOURCE_NAME            NOT NULL         VARCHAR2(32)
RESOURCE_TYPE                             VARCHAR2(8)
LIMIT                                     VARCHAR2(40)
```

- PROFILE is the name of the profile being described, such as DEFAULT.
- RESOURCE_NAME is the parameter being described, such as FAILED_LOGIN_ATTEMPTS.
- RESOURCE_TYPE indicates if the parameter describes a KERNEL or a PASSWORD limit.
- LIMIT is the value of the parameter being described.

```
SQL> select PROFILE, RESOURCE_NAME, LIMIT from dba_profiles where RESOURCE_NAME =
'FAILED_LOGIN_ATTEMPTS';

PROFILE                RESOURCE_NAME              LIMIT
---------------------- -------------------------- ----------------------
DEFAULT                FAILED_LOGIN_ATTEMPTS      3
MONITORING_PROFILE     FAILED_LOGIN_ATTEMPTS      UNLIMITED
```

Failed Login Attempts

Of all the password controls in Oracle, this one is the most important. The FAILED_LOGIN_ATTEMPTS parameter in a profile indicates the number of failed log in attempts that are allowed before a user's status is automatically changed to LOCKED. This feature is commonly referred to as *account lockout*.

The feature is implemented as a simple counter. At first the count is set to 0. With each failed login attempt, the count is increased by 1 until the value for FAILED_LOGIN_ATTEMPTS has been reached, at which point the account is locked. Users cannot log in to locked accounts even with the correct password. An administrator must unlock the account or the system must be configured with a password lock time value that automatically unlocks accounts.

Account lockout features are an effective security measure against password crackers. By limiting the number of guesses the tool can make, you can dramatically reduce the risk posed by password attacks. If you configure your system for a reasonable number of failed login attempts, between two and five, it becomes almost impossible to mount a successful password-guessing attack. This is an important measure to take right away, especially if you have weak passwords in your system today, you can significantly reduce the risk by configuring FAILED_LOGIN_ATTEMPTS on every profile in your system.

Configure account lockout after three failed attempts in the DEFAULT profile with the following statement:

```
alter profile DEFAULT limit FAILED_LOGIN_ATTEMPTS 3;
```

Setting FAILED_LOGIN_ATTEMPTS to UNLIMITED disables the account lockout feature. There is no reason to disable account lockout in any Oracle database.

Password Life Time

The PASSWORD_LIFE_TIME parameter puts a limit on the number of days that a password can be used before it must be changed. This feature is often called *password expiration*, as it automatically forces users to change their password on a periodic basis. It works by prompting the user for a new password on the first login after a password has expired. Oracle will not allow the user to proceed without changing their password once their password lifetime (and password grace time) have expired.

Periodically changing passwords is a good security measure. This may keep an attacker guessing, or could take access away from someone who has stolen a password. It is not uncommon for an attacker to steal a large number of passwords and then crack them over time. If it takes 30 days to crack your password, but you change your password every 30 days, you can effectively limit an attacker's ability to gain unauthorized access to your account and system.

Configure password life time for 45 days in the DEFAULT profile with the following statement:

```
alter profile DEFAULT limit PASSWORD_LIFE_TIME 45;
```

Setting PASSWORD_LIFE_TIME to UNLIMITED disables the password expiration feature. While it may be necessary to have some accounts with passwords that do not expire, you should create a special profile for only those accounts and choose a strong password. Make sure account lockout is enabled and set to a low threshold.

Password Reuse Max

The PASSWORD_REUSE_MAX parameter controls the number of times a user's password must be changed before an old password can be reused. The intention is to limit reuse of passwords by forcing users to wait a long time before the system will accept the old password again. Delaying reuse of old passwords based on the number of password changes since an old password has been used, is confusing. Instead of using this setting, we recommend using Password Reuse Time instead. These two parameters work together. Setting to UNLIMITED voids the setting of the other and makes it so that passwords can never be reused. In order for either PASSWORD_REUSE_MAX or PASSWORD_REUSE_TIME to take effect, both must be set to an integer value.

In order to use PASSWORD_REUSE_TIME exclusively, set PASSWORD_REUSE_MAX to *0*.

```
alter profile DEFAULT limit PASSWORD_REUSE_MAX 0;
```

WARNING

In general, we do not recommend using the PASSWORD_REUSE_MAX parameter to control reuse of old passwords. When this feature is used in conjunction with the PASSWORD_REUSE_TIME parameter, things can get very confusing, as even after the reuse time has passed an old password still may not be able to be reused, because there have not been enough interim password changes to satisfy the reuse max parameter. Confusing enough?

When used alone (with PASSWORD_REUSE_TIME set to *0*), the PASSWORD_REUSE_MAX is generally useless. There is nothing stopping a user from changing their password repeatedly until the threshold is met and the system accepts an old password for reuse. Even if the value is set to 10 or 100, it takes only a few moments to change a password over and over again and completely defeat the feature.

Password Reuse Time

The PASSWORD_REUSE_TIME parameter controls the number of days a user must wait before reusing and old password. This ensures that password changes stick, and that a user cannot simply change their password to a new value, then immediately change it back to the old value. This setting works in conjunction with the PASSWORD_REUSE_MAX setting (see above for details).

We recommend that you configure the password reuse time to be one year or 365 days. Setting a value of UNLIMITED ensures that an old password can never be reused.

```
alter profile DEFAULT limit PASSWORD_REUSE_TIME 365;
```

Password Lock Time

The PASSWORD_LOCK_TIME parameter controls the number of days an account will remain locked as a result of the Failed Login Attempts limit being reached before the account is automatically unlocked. In general, there is no reason to automatically unlock accounts where password guessing may have been attempted. Instead, manually unlock accounts after verifying why the account became locked in the first place. Did the user forget their password, or was something more sinister going on?

In order to disable automatic unlocking of accounts, set the PASSWORD_LOCK_TIME to UNLIMITED.

```
alter profile DEFAULT limit PASSWORD_LOCK_TIME UNLIMITED;
```

Password Grace Time

The PASSWORD_GRACE_TIME parameter works together with PASSWORD_LIFE_TIME. The PASSWORD_GRACE_TIME represents the number of days after the password life time has expired before a user is forced to change their password, rather then being prompted with the option of changing their password. This setting is designed as a convenience for users, allowing the database to give them a gentle reminder that their password has expired and to please change it soon, before stepping in and demanding the password be changed immediately.

The PASSWORD_GRACE_TIME has limited security implications. For most systems, a setting of 5 to 15 days is reasonable; however, consider reducing the PASSWORD_LIFE_TIME if a password grace time is set. If no grace time is desired, set this parameter to *0*.

```
alter profile DEFAULT limit PASSWORD_GRACE_TIME 0;
```

Password Verify Function

The PASSWORD_VERIFY_FUNCTION parameter indicates which (if any) PL/SQL function will be used to check the complexity of a new or changed password. This is the most flexible of all the password controls, as it allows you to specify a function of your choosing to check password strength, before accepting the password into the database. By default, the PASSWORD_VERIFY_FUNCTION is disabled (set to NONE); however, Oracle comes with a sample verify function that performs a number of checks:

- Password must be a minimum of four characters in length
- Password must not equal the username
- Password must include at least one alphabet character, one numeric character, and one punctuation mark
- Password must not be welcome, account, database, user, password, oracle, computer, or abcd
- Password must differ from the previous password by at least three characters

There is nothing wrong with using the built-in Oracle function, although it clearly has room for improvement. The PASSWORD_VERIFY_FUNCTION must be present in the SYS schema. To install the default function, connect as sysdba and run the script in $ORACLE_HOME/rdbms/admin/UTLPWDMG.SQL. This will create the function *verify_function* which you can then specify to be used as follows:

```
alter profile DEFAULT limit PASSWORD_VERIFY_FUNCTION verify_function;
```

This function is written in PL/SQL and is well commented and easy to understand. It would be a simple task to tweak the function to your needs, increasing the minimum length to six characters and adding to the very basic password dictionary. Get a little fancier by reading the password dictionary out of a file or table, allowing for easy checking against a list of thousands of easily guessed passwords. You can find a sample implementation of a dictionary-based password verify function at http://orafaq.com/node/1027.

Assigning Profiles to Users

Once you have all your profile settings configured, you must assign each user to a profile. Unless otherwise specified, users are members of the DEFAULT profile. If you made your configuration changes to the DEFAULT profile, you do not need to assign the profile to your users. However, if you created a new profile, or multiple profiles based on user roles, assign the profile to users with the following SQL statement:

```
alter user SCOTT profile DEFAULT;
```

If you have created a new profile but have not specified values for all parameters, those unspecified will revert to the DEFAULT value, which is the value programmed for the same parameter in the DEFAULT profile. When you list profile settings by selecting from *dba_profiles*, it is not unusual to see parameter values set to DEFAULT.

```
SQL> select PROFILE, RESOURCE_NAME, LIMIT from dba_profiles where RESOURCE_TYPE =
'PASSWORD' and PROFILE = 'MONITORING_PROFILE';

PROFILE                RESOURCE_NAME               LIMIT
---------------------- --------------------------- ----------------------
MONITORING_PROFILE     FAILED_LOGIN_ATTEMPTS       UNLIMITED
MONITORING_PROFILE     PASSWORD_LIFE_TIME          DEFAULT
MONITORING_PROFILE     PASSWORD_REUSE_TIME         DEFAULT
MONITORING_PROFILE     PASSWORD_REUSE_MAX          DEFAULT
MONITORING_PROFILE     PASSWORD_VERIFY_FUNCTION    DEFAULT
MONITORING_PROFILE      PASSWORD_LOCK_TIME           DEFAULT
MONITORING_PROFILE      PASSWORD_GRACE_TIME          DEFAULT
```

You can determine a users profile by selecting from the *dba_users* view:

```
SQL> select USERNAME, PROFILE from dba_users where USERNAME = 'SCOTT';

USERNAME                PROFILE
----------------------- -------------------------
SCOTT                   DEFAULT
```

Are You 0wned?

Remote OS Authentication

Oracle offers a "feature" to allow clients to establish remote connections to the database while relying entirely on the client's operating system (OS) to authenticate the database user. This feature is controlled by an initialization parameter called *remote_os_authent*; setting the parameter to TRUE enables remote OS-authenticated connections. If you've got this feature enabled on any of your databases, you need to ask yourself, am I owned?

With remote OS authentication enabled, it would be trivial for an attacker to compromise any users in your database that are able to authenticate via the OS. The worse case scenario here would be if the OS authentication prefix is set to null. This would allow an attacker to load an Oracle client on any machine on the network, and start that client with a local OS user they create, such as SYSTEM. Point the client at the database in question and log in using *sqlplus /*.

Oracle will recognize the client username as *SYSTEM*, it will add the OS prefix, in this case null, and then it will check for the resulting account name in the database user list. It won't run this exact query, but it's the concept that matters:

```
select NAME from SYS.USER$ where NAME = 'SYSTEM';
```

Of course a match will be found, and the user will be authenticated to the database as SYSTEM. Perhaps this is an extreme example. Let's consider a system that has *os_authent_prefix = OPS$*. If an attacker can gain any access to the target database, he or she can get a list of users by selecting from the ALL_USERS view.

Continued

```
C:\>sqlplus scott/tiger

SQL*Plus: Release 10.2.0.1.0 - Production on Sat Sep 15 13:18:52 2007

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select username from all_users;

USERNAME
------------------------------
OPS$SEAN
OPS$ERIC
BOOTSY
SRS
ALLAN
AARON
MARLEE
JILL
JOSH
HAVIV
JONATHAN
```

You can see that two users are listed that are set up for OS authentication (*OPS$SEAN* and *OPS$ERIC*). If either of these OS users have more privilege than SCOTT, the attacker has not elevated their level of access. Remember, with remote OS authentication enabled, the attacker only needs to set up an account with the right name in his system (in this case SEAN or ERIC) and then connect to the database without specifying a username or password.

If it seems too easy, that's because it is! Make sure you have *remote_os_authent* set to FALSE on all of your databases.

OS Authentication

Oracle allows for other methods of authentication that reside outside the database. The most commonly used is Local OS Authentication, facilitating database connections for users of the database host OS. OS authentication has been part of Oracle since day one. In fact, that was the only authentication mechanism supported in the earliest releases of the database. OS authenticated access to the database is a good thing, particularly since most of the operating systems used to host Oracle databases have strong authentication and password management mechanisms.

OS authentication can also pose some serious risks to your system, particularly if things are not configured properly. It's actually all fairly simple, but some parts are definitely not intuitive. With an understanding of how to set up and identify OS authenticated accounts, and how to enable the security features around OS authentication, you can both enable and secure local OS authenticated access to your databases.

Remote OS Authentication

Remote OS Authentication is a feature that allows Oracle databases to blindly accept connections from users authenticated by a remote OS (meaning logged into a system that is not the database server). Remote OS Authentication represents a significant security risk to any database and should be disabled on all systems, both production and non-production.

This is an issue of trust. How can you trust any remote system to properly authenticate users. You have no way to verify that the remote system is configured securely, nor can you tell if it has been taken over by a hacker. If users need to connect with a remote client, allow them to authenticate to the database with a username and password. OS authentication is useful for local connections only.

Disable Remote OS Authentication by setting the REMOTE_OS_AUTHENT initialization parameter in *init.ora* to *FALSE* and restarting the database. Note that if you currently have remote clients connecting to the database with remote OS authentication, those connections will no longer work.

OS Authentication Prefix

Oracle provides a means for tagging OS authenticated users by adding a prefix to their OS username in order to form their database username. This prefix is called the OS Authentication Prefix, and is configurable to any value you like. The value is controlled by the initialization parameter *os_authent_prefix* in *init.ora*, and is set to *OPS$* by default. If you decide to change this value, make the change when you first configure the database and then stick with it. Changing the OS Authentication Prefix on a working system can take a lot of effort.

When a user wants to make an OS-authenticated connection to an Oracle database, they use the following syntax, with no username or password specified:

```
sqlplus /
```

The database automatically captures the OS username, prepends the OS authentication prefix, and checks the value against the list of users in the database. If there is a match, authentication succeeds. No password check is required; the OS already checked the password. If there is no match, Oracle responds with the standard failed login message:

```
ERROR
ORA-01017: invalid username/password; logon denied
```

There is nothing wrong with using the *os_authent_prefix* default setting of *OPS$*. There would, however, be a problem with changing the value to null:

```
os_authent_prefix = ""
```

A null OS authentication prefix removes the tag on OS-authenticated users. This can lead to a real security issue where OS users can be created with the same name as an existing (and powerful) database user. Those users can then connect to the database without a password. The situation is significantly worse if remote OS authentication is enabled. If an attacker could create an OS user named SYSTEM or OUTLN or any number of other usernames, and the OS Authentication Prefix was set to null, they could connect to the database without a password, no matter how strong of a password was set for SYSTEM, OUTLN, or whatever other user is attacked.

Be sure that you have set the OS Authentication Prefix to some value. Even if you never changed the default setting, check it out and be sure, as the risk of a null prefix is too great to ignore.

```
SQL> select NAME, VALUE from v$paramter where NAME = 'os_authent_prefix';

NAME                                               VALUE
-------------------------------------------------- --------------------------
os_authent_prefix                                 OPS$
```

WARNING

Beside the OS Authentication Prefix, Oracle does not differentiate very well between OS-authenticated users and regular password-authenticated users. In fact, you can set a password for an account that was originally set up as OS authenticated, and then connect to that account using either OS authentication or Oracle authentication with the password.

This is what makes the attack possible when the OS authentication prefix is null. Users that were created as Oracle authenticated users, can also be authenticated by the OS. Oracle simply doesn't automatically enforce the method by which a user was intended to authenticate to the database.

Creating and Identifying OS Authenticated Users

It's simple to create OS authenticated users. In general, it's also simple to identify the OS authenticated users you currently have in your databases.

To create an OS-authenticated Oracle database user account for an OS user named *PNUT*, use the following command (assuming *os_authent_prefix* = *OPS$*):

```
create user OPS$PNUT identified externally;
```

The key here is the *identified externally* part. That tells Oracle to mark the account as an OS-authenticated account and to only accept OS-authenticated connections for that account. Note that the addition of the prefix *OPS$* is not automatic. You must know the prefix and manually add it when you create the user, otherwise the OS-authenticated connection will fail.

Check out the entry for this user in *dba_users*:

```
SQL> select USERNAME, PASSSWORD from dba_users where USERNAME = 'OPS$PNUT';

USERNAME                                 PASSWORD
---------------------------------------- ------------------------------------
OPS$PNUT                                 EXTERNAL
```

There are two key elements here that identify this user as OS-authenticated. First and foremost is the username that starts with *OPS$*, next is the password that is set to *EXTERNAL*. This is the way you want to see the password set for all of your OS-authenticated users, as this ensures they can only log in to the database with an OS-authenticated connection, not with the option of OS or Oracle authentication. It is definitely a good idea to find all the users in your database that are set up for OS authentication and make sure their password is *EXTERNAL.* The following query will give you a list of users set up for OS authentication, but who also have an Oracle password (assuming *os_authent_prefix = OPS$*):

```
select USERNAME from dba_users where USERNAME like 'OPS$%' and PASSWORD <>
'EXTERNAL';
```

Automated Scanning for Weak Passwords

You can take big steps by teaching folks how to create strong passwords, and by implementing the password control settings in Oracle profiles. What those measures do not do, however, is help you to identify and eliminate the weak passwords that are already in use in your systems. In order to do this, you will need an automated tool. Manually driven weak password detection is really not feasible. Luckily, there are good tools out there to help you that are available as both free open source versions, and fully supported commercial products.

Tools & Traps…

Oracle Password Scanners

Oracle password scanners or crackers come in various packages. One mechanism for characterizing these tools is to examine how they operate, and what information is required to get them going. Some scanners work by repeatedly attempting to log on to an Oracle system. They may cycle through a dictionary of passwords, or brute force scan based on a number of characters, or check for default accounts or accounts with username = password. Whatever mechanism they use to pick a username and password is not important for this characterization, it's that they attempt to log on to the database to determine if a username/password combination is valid.

This method of password scanning is powerful as it requires almost no knowledge. Point at an Oracle database and off it goes. If you have a list of usernames, it can do even better, targeting only those accounts that exist. This kind of zero knowledge scanning is particularly useful to an attacker coming at the database from the outside. The downside of this kind of outside-in scanning is that it puts a large load on the database server to process all those failed login attempts.

Continued

The other noticeable problem with this approach comes into play when account lockout is enabled. It would be easy for a password scanner to lock out every single account in an Oracle database with only a few seconds worth of password guessing. If you are going to run a password scanner that works by repeated login attempts, make sure you temporarily disable account lockout or face the consequences of locking lots of accounts. One nice thing to know is that it's not possible to lock a *sysdba* account, so you can always connect to a system as *sysdba* and unlock all the users you locked out.

A second mechanism for password scanning requires more knowledge; either a single username/password hash pair, or access to a list of usernames and password hashes (usually accessed vie the DBA_USERS view). These tools work offline, making their password guesses, but instead of submitting login requests to the database, they calculate the password hashes on their own and compare to the known value.

These inside-in password scanners are the most powerful tools for finding weak passwords and are also the most efficient. Since only valid users are scanned, the time of the scan is reduced. More importantly, this analysis phase of this kind of scan can be performed on a dedicated scanning machine instead of on the database server. This means almost no performance impact on the database for guessing passwords all day and night.

If you are generally going to have access to log in to the databases you want to scan for weak passwords, you should find a tool that scans inside-in, collecting the list of usernames and password hashes from the database and using them to verify the existence of weak passwords, by scanning offline. If you have no other choice, outside-in password scans will often reveal a default account that can then be used for a more efficient inside-in scan.

Freeware Password Scanners

The freeware scanning tools out there are great for very technical users that deal with small environments of databases. Commercial products are more flexible and scalable and tend to be much simpler to use. If you use freeware scanners, keep in mind that you are basically using a hacker tool with no guarantees of what could result. Be careful with your selection of freeware password scanners. Make sure to select one that is widely used and well known.

Probably the best and fastest free password scanner for Oracle is checkpwd from red-database-security. It comes packaged in a few varieties, some including all of the necessary client drivers and a very large password dictionary. Download checkpwd at http://www.red-database-security.com/software/checkpwd.html. Checkpwd is a command-line tool and output from running the tool looks like this:

```
C:\>checkpwd system/strongpw@//123.34.54.123:1521/ORCL password_list.txt


Checkpwd 1.23 [Win] - (c) 2007 by Red-Database-Security GmbH
Oracle Security Consulting, Security Audits & Security Training
http://www.red-database-security.com

initializing Oracle client library
connecting to the database
retrieving users and password hash values
opening weak password list file
reading weak passwords list
checking passwords
Starting 2 threads
MDSYS has weak password MDSYS [EXPIRED & LOCKED]
ORDSYS has weak password ORDSYS [EXPIRED & LOCKED]
DUMMY123 has weak password DUMMY123 [OPEN]
DBSNMP OK [OPEN]
SCOTT has weak password TIGER [OPEN]
CTXSYS has weak password CHANGE_ON_INSTALL [EXPIRED & LOCKED]
SH has weak password CHANGE_ON_INSTALL [EXPIRED & LOCKED]
OUTLN has weak password OUTLN [EXPIRED & LOCKED]
DIP has weak password DIP [EXPIRED & LOCKED]
DUMMY321 has weak password 123YMMUD [OPEN]
[...]
SYS OK [OPEN]
SYSTEM OK [OPEN]

Done. Summary:
Passwords checked : 13900828
Weak passwords found : 23
Elapsed time (min:sec) : 0:54
Passwords / second : 265486
```

Another free password scanning tool is orabf from toolcrypt.org. orabf can run both a dictionary-based scan from a file, as well as a brute-force attack with a specified number of characters. In order to run orabf, you need a username and it's password hash. orabf will try to crack it. orabf is a command-line tool, and is easy to run. This is what it looks like:

```
C:\orabf>orabf 79805A2064887852:DBSNMP -n 4


orabf v0.7.6, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...done


Starting brute force session using charset:
#$0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_
```

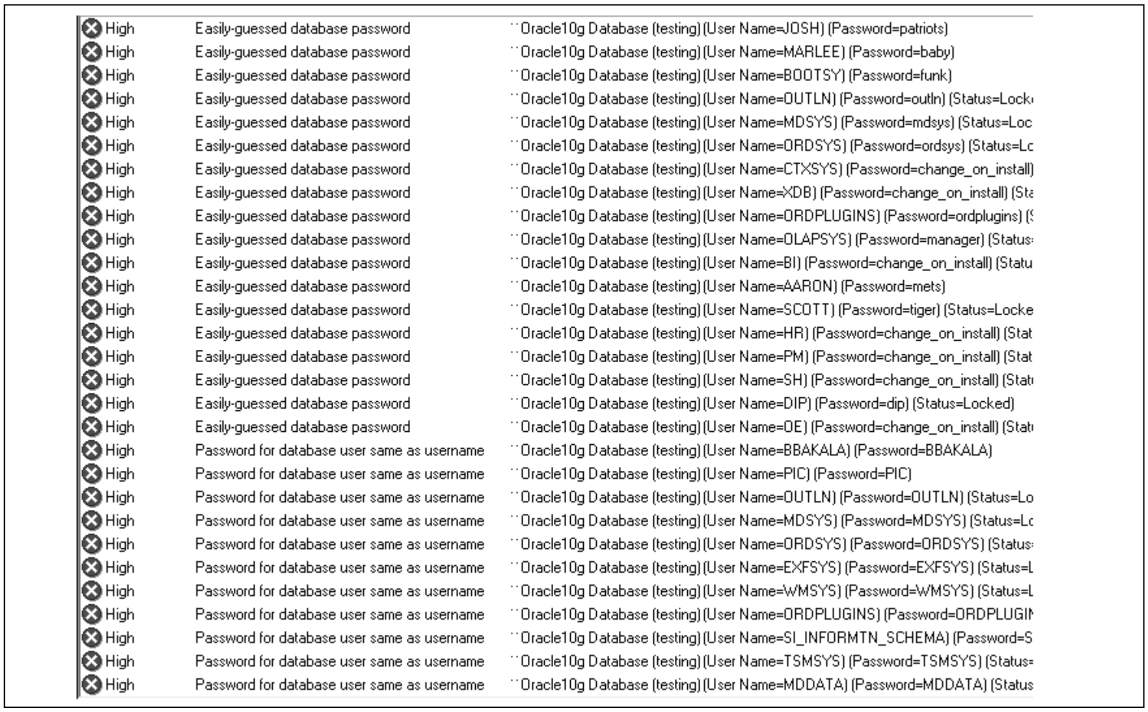
```
press 'q' to quit. any other key to see status

current password: BAJIE
4600301 passwords tried. elapsed time 00:00:06. t/s:721616

current password: CZNFM
8402692 passwords tried. elapsed time 00:00:11. t/s:722812

current password: EZCWF
13013499 passwords tried. elapsed time 00:00:18. t/s:722972

current password: IQY#F
21887793 passwords tried. elapsed time 00:00:30. t/s:707484

current password: KY60E
26824451 passwords tried. elapsed time 00:00:38. t/s:704747

177301218 passwords tried. elapsed time 00:04:22 t/s:674630
```

There are other free Oracle password scanners out there including John the Ripper (http://www.openwall.com/john) and Cain and Able (http://www.oxid.it/cain.html). Remember, these are powerful tools and you run them with no warranty or support. Use them on test systems before running in a production environment.

Commercial Password Scanners

You're probably not going to find all that many commercial products that do nothing but password scanning. Generally password scans are integrated into complete vulnerability scanning systems. There are a small number of commercially available vulnerability scanners that are focused on in-depth assessment of databases. These highly specialized tools can find almost any vulnerability or misconfigurations in your databases. They all offer powerful password scanning capability, some with both brute-force and dictionary scan features. Figure 7.2 shows Application Security, Inc's AppDetective Pro.

Figure 7.2 Application Security, Inc.'s AppDetectivePro Password Scan Results



```
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=JOSH) (Password=patriots)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=MARLEE) (Password=baby)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=BOOTSY) (Password=funk)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=OUTLN) (Password=outln) (Status=Lock
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=MDSYS) (Password=mdsys) (Status=Loc
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=ORDSYS) (Password=ordsys) (Status=L
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=CTXSYS) (Password=change_on_install)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=XDB) (Password=change_on_install) (St
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=ORDPLUGINS) (Password=ordplugins) (
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=OLAPSYS) (Password=manager) (Status
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=BI) (Password=change_on_install) (Statu
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=AARON) (Password=mets)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=SCOTT) (Password=tiger) (Status=Locke
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=HR) (Password=change_on_install) (Stat
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=PM) (Password=change_on_install) (Stat
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=SH) (Password=change_on_install) (Stat
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=DIP) (Password=dip) (Status=Locked)
High  Easily-guessed database password            ``Oracle10g Database (testing) (User Name=OE) (Password=change_on_install) (Stat
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=BBAKALA) (Password=BBAKALA)
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=PIC) (Password=PIC)
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=OUTLN) (Password=OUTLN) (Status=Lo
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=MDSYS) (Password=MDSYS) (Status=L
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=ORDSYS) (Password=ORDSYS) (Status
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=EXFSYS) (Password=EXFSYS) (Status=L
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=WMSYS) (Password=WMSYS) (Status=L
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=ORDPLUGINS) (Password=ORDPLUGI
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=SI_INFORMTN_SCHEMA) (Password=S
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=TSMSYS) (Password=TSMSYS) (Status=
High  Password for database user same as username ``Oracle10g Database (testing) (User Name=MDDATA) (Password=MDDATA) (Status
```

The top commercial products that focus on database vulnerability scanning are Application Security, Inc.'s DbProtect and AppDetectivePro (www.appsecinc.com) and Next Generation Security Software's NGSSQuirreL (www.ngssoftware.com). Both products have great password scanning capabilities, allowing you to run dictionary scans, brute-force scans, check for username = password, and do it all quickly and efficiently. Both products support other commercial database platforms as well, giving you options for scanning the non-Oracle databases within your organization.

Summary

Password crackers represent a significant risk to Information Technology (IT) systems that protect themselves with no more than a username and password. Powerful password scanners are freely available for download in various places on the Internet, allowing even the least experienced hackers the ability to break into what should be secured systems.

Oracle provides a number of security features to protect against password attacks. The features are easy to implement and can turn a database from an easy cracking target into a hardened fortress. Strong password policies are a key part of any security plan. Establish a system for managing and generating passwords, and teach your users how to take advantage of it.

The education process is important here. You can't just enable a complex password verify function, expire everyone's password, and expect things to work out. Users may be unable to pick an acceptable password or may choose something they instantly forget. In either case it means more work and hassle for everyone. Make a plan for better passwords, distribute it, and then take the step to have users reset their password based on the new standards. Security is a process, and everyone has to learn to be part of it.

In order to protect your systems, you should periodically run a password-cracking scan and change any weak passwords that are discovered. Both freeware and commercial tools are available, the choice is yours. A system with weak passwords and minimal password controls is like a safe with the combination taped to the lock. The safe looks imposing and secure from a distance, but once you get close it's a piece of cake to get inside. Make the lock on the door to your Oracle database work for you—lock down your passwords and password controls.

Solutions Fast Track

Configuring Strong Passwords

☑ Passwords protect access to the database. Because of this, they are a prime target for attackers. There are literally hundreds of freely available password cracking tools out there, many of which can be used to attack Oracle databases.

- ☑ Weak passwords are those that are easy to guess, either for a human or a computer. Passwords are weak if they are dictionary words, usernames, sports teams, dates, or patterns.
- ☑ Use a password management tool to generate and store strong passwords for all your systems. If you want to try to generate your own strong passwords and remember them, set up a password rule set.

Password Controls Using Oracle Profiles

- ☑ Enable Oracle's account lockout feature by configuring the FAILED_LOGIN_ATTEMPTS parameter in every profile in the database. This feature stops password guessing attacks by locking the account after a set number of failed logon attempts.
- ☑ Use the PASSWORD_LIFE_TIME profile feature to automatically expire users passwords, forcing them to choose a new password. The new password must meet all the criteria enforced by Oracle's password controls.
- ☑ Use the PASSWORD_VERIFY_FUNCTION to check the strength of new passwords. Oracle provides a default function that can be used or easily modified to provide additional protections. There are even better sample functions available on the internet.

OS Authentication

- ☑ Oracle has native support for accepting connections to the database that are authenticated by the OS rather than the database. The OS-authenticated connections are a good thing, but must be configured properly to avoid security problems.
- ☑ Be sure to disable remote OS authentication on all your databases. When set to TRUE, this configuration allows Oracle to accept connections that are authenticated by a remote OS rather than the database server. Since it is impossible to trust any remote OS, this represents a huge risk.
- ☑ Configure an OS authentication prefix to help identify accounts that are set up for OS authentication, and to block simple attacks where an attacker uses OS authentication to connect to Oracle using an account that was meant to be accessed via Oracle authentication only.

Automated Scanning for Weak Passwords

☑ Manually searching for weak passwords is an impossible task, even in a single database. There is simply no mechanism to decipher the password hash values stored in the database to determine the original password.

☑ Automated tools that scan Oracle databases for weak passwords are widely available. There are both freeware scanners and commercial products available that target databases. The free software is better suited for small environments run by tech-savvy users. The commercial products are better for scanning large numbers of systems with an easy-to-use interface.

☑ Oracle password controls only impact new passwords, they do not go back and flag existing passwords that don't meet the new standards. It's important to scan all systems for weak passwords on a regular basis, and to move aggressively to have them changed.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to www.syngress.com/solutions and click on the "Ask the Author" form.

Q: Can you explain how to configure Oracle's account lockout feature?

A: Set the FAILED_LOGIN_ATTEWMPTS parameter in a profile (or all profiles) to a numeric value (the value UNLIMITED disables account lockout). Associate any or all users with this profile. You can modify this value in the built-in DEFAULT profile, which will cover all users that have not been explicitly associated with a different profile. Here is an example:

```
alter profile DEFAULT limit FAILED_LOGIN_ATTEMPTS 3;
alter user SCOTT profile DEFAULT;
```

Q: Do I really need to configure strong passwords on my non-production systems? It's much easier to set easy to remember passwords.

A: Yes! Many non-production systems hold sensitive data. This may be a copy of production data being used in development or for testing, or it may be information about the production schema and configuration. An attacker is just as happy to steal your secrets from your development lab as she is to steal them from your fully locked down production data fortress. Furthermore, if an attacker can take over the non-production server via the database (lots of attacks allow this to happen), they can use it as an internal point to gain reconnaissance and launch attacks against other internal systems. It's important to configure strong passwords on all your Oracle systems.

Q: I've seen a few Oracle attack demos where the attacker is trying to get a copy of the usernames and passwords from the DBA_USERS view. Aren't these passwords secured with some kind of encryption?

A: The passwords stored in DBA_USERS (actually stored in SYS.USER$) are stored securely. They are not exactly encrypted; they are hashed, which is a cryptographic operation similar to encryption except that a hash is a one-way function. There is no way to take an Oracle password hash from DBA_USERS and work backwards to the user's password. There is, however, a simple means to take a whole lot of passwords and work forwards to create an Oracle password hash. Compare the hash you calculate with the value stored in the database. If you find a match, you know the user's password. There are many freely available tools out there that automatically perform this kind of password cracking on Oracle password hashes. Make sure to keep any non-DBA users out of Oracle's list of database passwords.

Q: Do you have any tricks to use for remembering strong passwords?

A: The best trick is to use a password management tool; there are a few good ones out there that you can get for free. If you want to remember them yourself, find something you can easily remember and use it to build a strong password. For example, take a phrase such as "I was born in Boston, MA in February of '76," and use the first letter of each word, plus all the numbers and punctuation to make a strong password: IwbiB,MiFo'76.

Q: What is a Password Verify Function and how do I use it?

A: The Password Verify Function is Oracle's means of providing controls on password complexity. It allows an administrator to specify a PL/SQL function that will be used to check the complexity of every new password set in the database. The function must conform to a defined application program interface (API), and must exist in the SYS schema. Otherwise, it can do whatever you please. Oracle packages a sample function in the file *utlpwdmg.sql*, (found in *$ORACLE_HOME/rdbms/admin*). Run this script to install the default *verify_function*. This provides some basic complexity checks, which you can easily improve or tailor to your organizations needs.

Q: What is the OS_AUTHENT_PREFIX and why is it important?

A: OS_AUTHENT_PREFIX is a configuration parameter that allows you to specify a value that will be automatically prepended to the usernames that authenticate via the OS. By default, OS_AUTHENT_PREFIX is set to *OPS$*. Therefore, for an account to connect to the database with OS authentication, that account name with the proper prefix must exist as a user in the database. Setting OS_AUTHENT_PREFIX to null creates a security hole where no prefix is added to OS usernames logging into the database. Since Oracle trusts the OS to check the users password, all it does is verify that the proper username exists in the database. If you create an OS user named SYSTEM and the OS_AUTHENT_PREFIX was null, that OS user could log in as the database user SYSTEM without any password checking done by Oracle. It's really important to set OS_AUTHENT_PREFIX to a non-null value.

Q: How often should I scan my production systems for weak passwords? What about my non-production systems?

A: It's a good practice to scan all of your production databases for weak passwords on a monthly basis. Non-production systems should also be scanned regularly, ideally just as often as production. Having a commercial database vulnerability scanner that can automatically run large scan jobs across many databases makes it easy to scan all systems with nearly any frequency that you desire. At a minimum, be sure to scan your non-production systems for weak passwords on a quarterly basis.

Q: How often should I check the password control settings on my databases?

A: Ideally, you would only need to check them once. Get them set correctly, and only re-check them when and if your database monitoring system detects a change to one of the settings. This actually is possible, but you'll need to purchase a complete database security platform to get it. More realistically, check the password controls on all of your systems at least once a quarter; it's even better to check once a month.

Chapter 8

Database Activity Monitoring

Solutions in this chapter:

- Database Intrusion 101
- Detecting Known Attack Patterns
- Detecting Suspicious Activity
- Tracking the Attacker
- Adhering to Government and Industry Regulations

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Monitoring database activity serves many purposes. Detecting intruders is common, but you will also want to be alerted to suspicious transactions that step outside of the bounds of what you would consider "normal," which might indicate that an authorized user was acting in an unauthorized way. For example, a database administrator could be stealing credit card numbers. Monitoring such actions helps toward fulfilling several government and industry regulations such as the Sarbanes-Oxley Act in the United States or the European Union's Data Protection Directive.

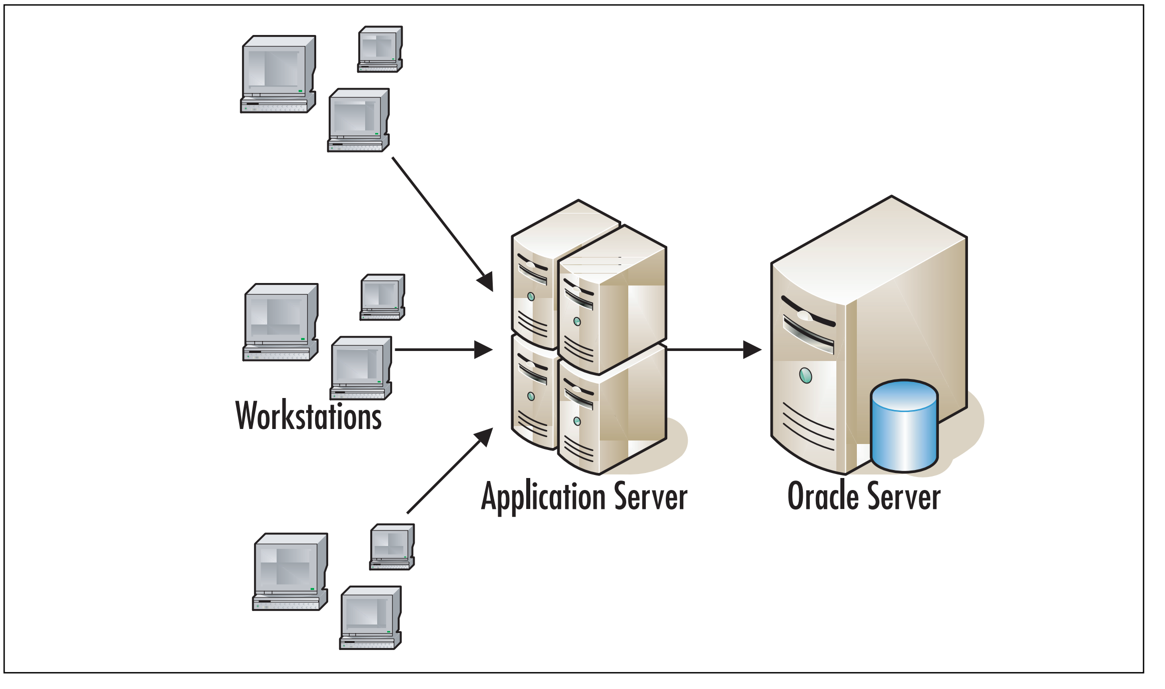
Forrester Research estimates that 70 percent of all database breaches are internal.[1] Knowing *who* is accessing your data and *how* they are doing so is as important as ever. It is also important to know *when* the access happened so that you are able to correlate it with other suspicious activity across your enterprise. Even better, if you know of an attack as it occurs, you will be in a position to taken defensive action against it, such as turning off a user's account before he or she is able to steal your entire data repository.

A real-time activity monitor for your database is like a security alarm for your home. It lets you know when something is happening. You set the areas of your property you wish it to observe and you set its sensitivity. Monitoring the street in front of your house would let you know about cars driving up to your house as well as about those just passing by. Monitoring every spec of dust that floats through the air would yield more information than is practical, but desensitizing the system to ignore common movement enables viewing abnormalities more clearly. Your database monitoring solution should be able to perform similar functions for your database, notifying you of activity that is interesting and ignoring that which is not.

Database Intrusion 101

Let's start by reviewing some basic concepts from the attacker's perspective. This will give you enough background to understand how database intrusion detections systems (IDS) work and how they interact with your environment. Figure 8.1 illustrates a basic three-tiered application having multiple clients, an application or Web server in the middle, and a database as the backend.

Figure 8.1 Three-Tiered Application



Injecting SQL

Clients communicate with the middle tier using a protocol such as Hypertext Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP), or some Remote Procedure Call (RPC) technology. Whatever the mechanism used in your particular application, the clients send data and the application server uses that data as part of a database command. One class of database attacks called Structured Query Language (SQL) Injection refers to the ability of a client-side user to influence the construction of that database command. Very often the client is a Web browser lying outside of the corporate firewall, so this type of attack can be executed by outsiders or insiders alike. Let's look at some sample code that constructs a query for information based on the user's ID and password. If there is a match, the client will display the name and address of the user, and if no results match, the client will display a message indicating the username or password was invalid:

```
String sql = "SELECT name, address FROM Users WHERE id = '" + id + "' "
             "AND password = '" + password + "'";
```

Notice anything wrong?

On the surface it looks fine, but if the value of *id* comes straight from the user and you imagine the possible values he or she may use, you'll start to see where a hacker could get their hooks into your system. Let's say I'm user *zach* but I want to see the data for *jessica*. To do this, I simply send my username as the following string:

```
jessica' --
```

The application server will construct the following SQL command and send it to the database, thus returning *jessica*'s, not *zach*'s, information:

```
SELECT name, address FROM Users WHERE id = 'jessica'
--' AND password = 'password'
```

The above indicates a way to steal someone's identity, but it has other variants that give an attacker even more access to your data. Let's look at a different value for the *id* field:

```
' union select banner, '' from $version --
```

This yields the following SQL, which lets the attacker know exactly what patchset of software your Oracle server uses, which in turn lets him or her know what vulnerabilities are potentially open:

```
SELECT name, address FROM Users WHERE id = ''
union select banner, '' from $version --' AND password 'password'
```

Attacks can continue from there, injecting whatever the attacker wants. To detect such attacks, look for key indicators such as an odd number of quotes in the query string. Additionally, to save network bandwidth and server parsing time, most applications do not send SQL comments. So, to catch a potential intruder, also look for commands that contain comments. The two most common indicators of SQL Injection attacks are unclosed quotes and the presence of SQL comments. Be sure your IDS software can detect these. If your database regularly receives queries containing comments, your IDS will need to have a mechanism to allow you to ignore the commands you expect and catch the ones you do not.

HTTP Server

The Web server that comes with Oracle offers users convenient access to the server without requiring them to install Oracle client software. The interface requires a username and password and then offers a freeform text field permitting the full range of Oracle database commands. The downside of this convenience is that sometimes external attackers will use Google to find such Web sites that are exposed to the

Internet. Your lockdown plan should include either shutting off the service or putting it behind a firewall.

WARNING

Firewalls are designed to permit only desirable traffic into your network. Generally, the default is to permit everything with the exception of Web requests. If the underlying implementation is to permit common HTTP/Hypertext Transfer Protocol Secure (HTTPS) ports (e.g., 80, 8080, and 443) regardless of the destination's IP address, then the Web server sitting in front of your Oracle database may be exposed. Be sure your firewall is configured to permit only requests for sites that you want open to the public.

Either way, it is a good practice to monitor HTTP traffic having a Uniform Resource Locator (URL) pattern that indicates a request to Oracle's Web server. If the service is turned off, your users cannot use it. So, if anyone attempts to use it this may indicate an attack. If on the other hand the service is turned on and it is behind a firewall, you should enable monitoring for requests coming from any client or subnet not in your list of approved client machines, or using a username that is not in your list of approved users of the HTTP server.

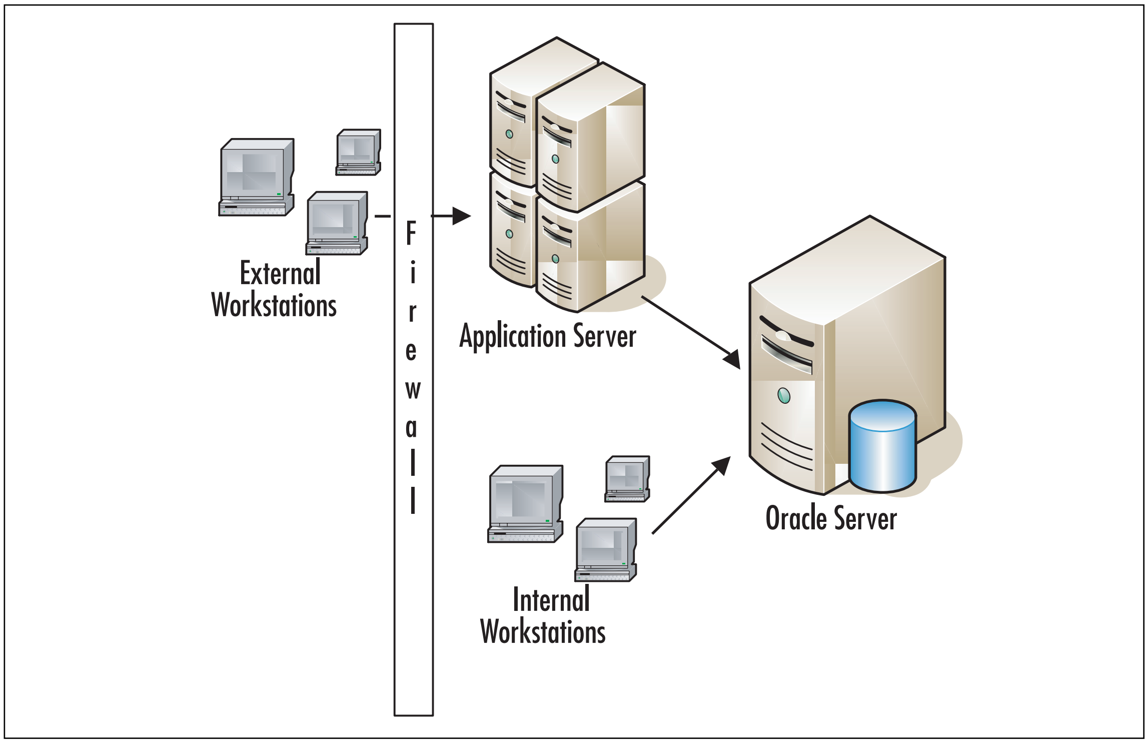
TIP

As the administrator of a database IDS, generate a list of approved users, the hosts or subnets from which they may authenticate, and the list of applications they may use (e.g., the Oracle HTTP server). Configure your IDS to flag any traffic that originates from any source that is not authorized.

Direct Database Access

The same network access to the database that is available to the Web/application server in Figure 8.1 is often also available to many people within an organization, as illustrated in Figure 8.2. To execute many attacks, rogue insiders need only freely available Oracle client software, but with a bit more effort they don't even need that.

Figure 8.2 Insider Access



Pretend for a moment that you are a malicious user working for some large corporation and you want to manipulate your salary. You do not have an account on the database, so first you will need to find one. Let's try the most well known user and password to attempt to access the human resources database:

```
SQL> connect scott/tiger;
Connected.
```

Excellent! You're in. Now let's see if you are in the "dba" role.

```
SQL> set role dba;
set role dba
*
ERROR at line 1:

ORA-01924: role 'DBA' not granted or does not exist
```

Hmmm, you do not have DBA privileges. However, because you are a savvy user and you keep up on security issues, you know of an attack that will execute a command of your choosing as CTXSYS, a DBA:

```
SQL> begin ctxsys.driload.validate_stmt('grant dba to scott'); end;
  2  /
begin ctxsys.driload.validate_stmt('grant dba to scott'); end;
*
ERROR at line 1:
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at "CTSYS.DRILOAD", line 42
ORA-01003: no statement parsed
ORA-06512: at line 1

SQL> set role dba;

Role set.
```

Bingo! You now "own" the database because you have just given yourself DBA rights.

NOTE

This example is very real, but intentionally very old. The particular vulnerability exploited is the same one used in the Voyager worm in early 2005. It affects only Oracle 8i and 9i and a patch has since been released.

Oracle Listener

The Oracle listener is a service that runs on the database host and receives requests from Oracle clients. Sometimes the listener forwards requests to the database server (Figure 8.3), and sometimes it redirects the client somewhere else where the database server listens for dedicated connections (Figure 8.4). In either case, it is the first point of contact for the client.

Figure 8.3 Shared Sever Process

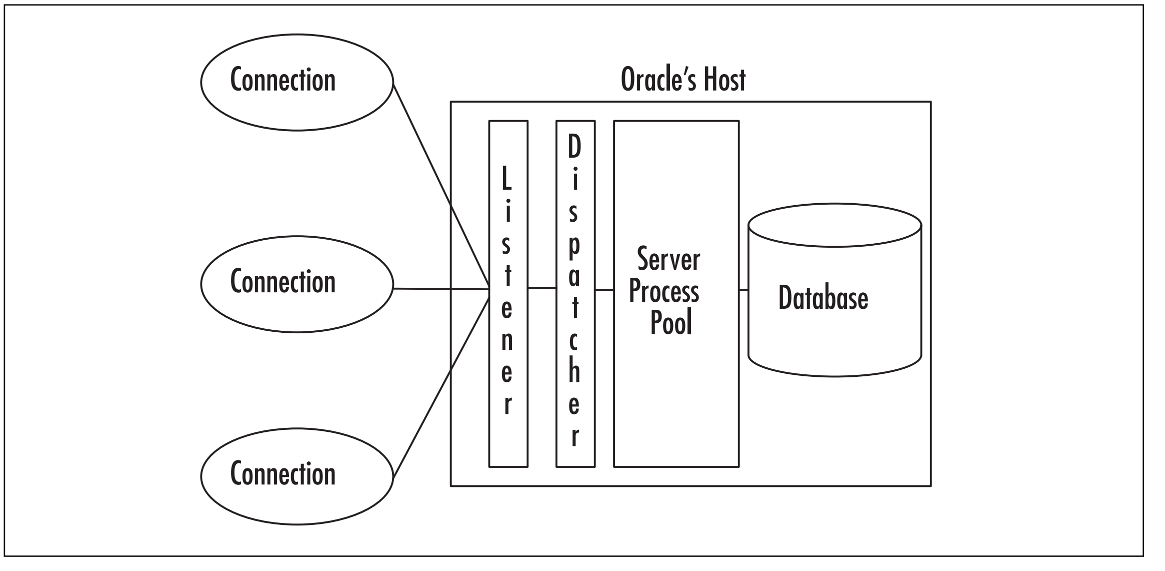
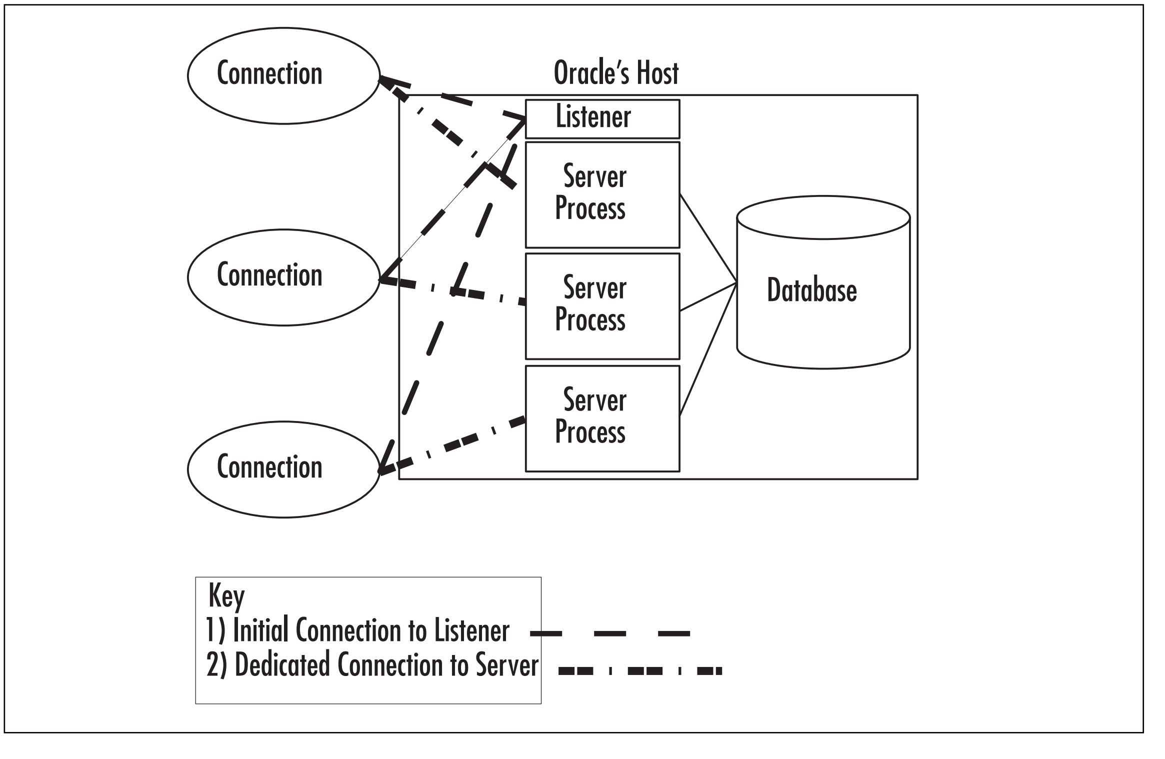


Figure 8.4 Dedicated Server Processes



The easiest attack by far for an insider is against a listener that doesn't have a password. Executing listener commands such as "stop" is trivial if you have a copy of Oracle's listener control software, lsnrctl:

```
lsnrctl stop 192.168.1.2
```

TIP

The Oracle listener is the front door to your database. Lock the door by configuring it to require a password. Before 10g it was unlocked by default, so be sure to check your older database servers.

Also consider configuring your listener to only accept connections from trusted sources. The *tcp.validnode_checking*, *tcp.invited_nodes*, and *tcp.excluded_nodes* parameters in the *sqlnet.ora* configuration file allow you to do that. See Oracle's documentation for details: http://oraclesvca2.oracle.com/docs/cd/B12037_01/network.101/b10775/profile.htm - i485056

Detecting Known Attack Patterns

Intrusion detection at the database level is an important step forward as you safeguard your critical systems. Oftentimes, "the devils are inside the walls"[2] meaning that employees, contractors, and others with access to your internal networks are the attackers in a majority of cases, and they might have direct access to your databases without a firewall standing in their way. It is of utmost importance to monitor activity to and from your database. Looking for known attack patterns is an easy solution because commercial products exist that can do this for you. The best solutions require intimate knowledge of the hacker community and an ability to update its attack signature library, much like a virus detection product knows about viruses and updates itself with the latest data.

WARNING

Hackers who do not already have a valid username and password will often attempt to find one as a first step toward breaking in. Being able to authenticate to the database is the first hurdle that must be overcome for most breaches to be successful. Blank or default passwords, especially those associated with default accounts which Oracle's installer creates, are easy to try.

Your database security solution should be able to detect the usernames being used during authentication (some can detect passwords, too), and it should be able to audit your *sys.user$* table for easily guessed passwords, default user accounts in use, and related settings that impact the security of your system.

Some of the attacks that take advantage of software flaws can escalate privileges of an existing user, and some of them can give an attacker SYS access without authenticating. The good news is that a majority of attacks of this type are against published vulnerabilities. Security researchers can deduce attack signatures with very little information and add them to their IDS solutions. As we discussed earlier in this book, be sure to know what vulnerabilities your database has even if you are unable to fix them. This gives you the ability to monitor for attacks against those problems and escalate them to a higher risk level if they occur. An example of this might be checking for a large number of characters used as a parameter of a stored procedure known to be vulnerable to a buffer overflow attack.

Another type of attack seeks not database access, but access to Oracle's host. For example, use of the *UTL_FILE* package gives access to the file system. If a certain file has a database administrator's password in clear text[3], it would be easy for a regular user to get the password. Your IDS should be able to flag use of *UTL_FILE* and other means of interacting with the host operating system.

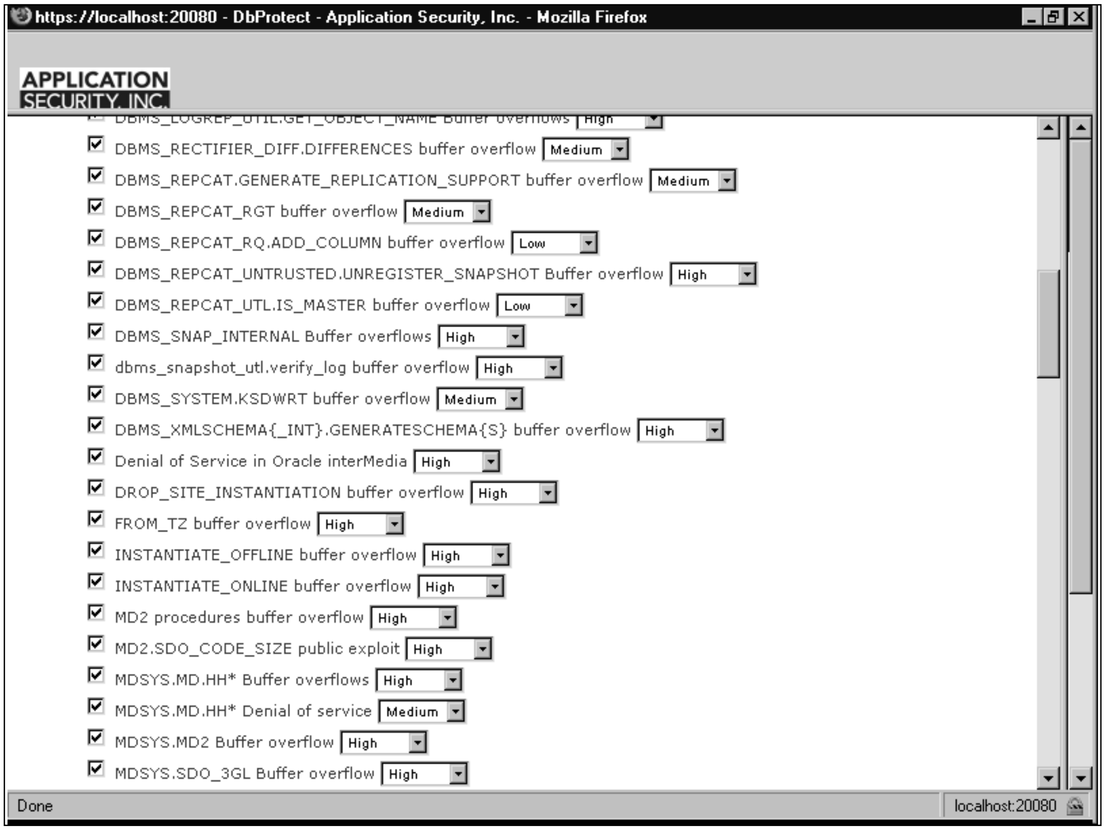
One more type of attack doesn't require knowledge of software flaws. It simply uses a dictionary of usernames and passwords and tries them in what is called a "brute force" authentication attack. A single failed login attempt may not be a security issue, but a series of them in a limited timeframe should raise a red flag. A slightly elevated volume of failed logins is probably someone manually trying a bunch of passwords, but a significant volume of them is most likely a script. The latter is more dangerous for two reasons. First, it has a higher chance of successfully finding a valid username/password pair. Second, it can lock out your valid accounts by using too many bad passwords for a single user. This prevents the real user from accessing the database.

Here is a list of several well-defined types of attacks for which you should monitor:

- Exploits of database software flaws
- Monitor for exploits of vulnerabilities whether you have fixed them or not. This will alert you to an attack even if it is unsuccessful.
- Exploits of client application software flaws
- SQL Injections
- Cross-site scripting
- Accessing operating system resources (e.g., file system, network)
- Password attacks
- Denial of Service (DoS) attacks
- Failed authentication attempts to lock out an account
- Massive number of open connections which can drain resources
- Exploits of configuration flaws
- Use of default username
- Use of blank password
- Use of services or packages that have not been turned off

DbProtect from Application Security, Inc. has a detailed list of exploits it detects. The following screenshot indicates a sample monitoring policy that contains rules to watch for known attack patterns against Oracle database servers shown in Figure 8.5.

Figure 8.5 DbProtect



https://localhost:20080 - DbProtect - Application Security, Inc. - Mozilla Firefox

APPLICATION SECURITY, INC.

☑ DBMS_LOGREP_UTIL.GET_OBJECT_NAME Buffer overflows	High
☑ DBMS_RECTIFIER_DIFF.DIFFERENCES buffer overflow	Medium
☑ DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT buffer overflow	Medium
☑ DBMS_REPCAT_RGT buffer overflow	Medium
☑ DBMS_REPCAT_RQ.ADD_COLUMN buffer overflow	Low
☑ DBMS_REPCAT_UNTRUSTED.UNREGISTER_SNAPSHOT Buffer overflow	High
☑ DBMS_REPCAT_UTL.IS_MASTER buffer overflow	Low
☑ DBMS_SNAP_INTERNAL Buffer overflows	High
☑ dbms_snapshot_utl.verify_log buffer overflow	High
☑ DBMS_SYSTEM.KSDWRT buffer overflow	Medium
☑ DBMS_XMLSCHEMA{_INT}.GENERATESCHEMA{S} buffer overflow	High
☑ Denial of Service in Oracle interMedia	High
☑ DROP_SITE_INSTANTIATION buffer overflow	High
☑ FROM_TZ buffer overflow	High
☑ INSTANTIATE_OFFLINE buffer overflow	High
☑ INSTANTIATE_ONLINE buffer overflow	High
☑ MD2 procedures buffer overflow	High
☑ MD2.SDO_CODE_SIZE public exploit	High
☑ MDSYS.MD.HH* Buffer overflows	High
☑ MDSYS.MD.HH* Denial of service	Medium
☑ MDSYS.MD2 Buffer overflow	High
☑ MDSYS.SDO_3GL Buffer overflow	High

Done localhost:20080

Tools & Traps…

Assessing Your Security Posture

A major part of database security is taking an assessment of the vulnerabilities open in your systems. A good approach to addressing them is to sort them by risk and then schedule fixes. However, it is also a good practice to take a step back and assess any patterns of problems. Seeing a long list of things like "FROM_TZ buffer overflow" and "CREATE VIEW privilege escalation" may not mean anything at first glance, but when you consider that both require software patches, you can start to see a pattern. Putting vulnerabilities into categories that generically describe the fix rather than the problem will help you identify larger issues in your enterprise. Here are the categories I typically use:

Continued

- Oracle vulnerabilities/software flaws (e.g., a bug fixed in a patchset, a CPU or security alert)
- Misconfigurations (e.g., a user account that was not turned off or a file with promiscuous permissions)
- Password problems (e.g., a blank or easily guessed password)
- Application vulnerabilities/software flaws (e.g., an Hypertext Markup Language [HTML] field is open to a SQL injection attack)

If a lot of your problems are misconfigurations, this indicates that your database administrators may need training in security procedures related to installing and maintaining your databases. Likewise, problems in custom applications that use the database may indicate you need to train your software developers to account for SQL injection attacks and hackers creating specially crafted application requests. Any recurring patterns may indicate that you need to reassess your security posture to put more focus on particular areas of concern.

Detecting Suspicious Activity

Detecting known attacks is only one part of the puzzle for a database security monitoring solution. It also needs intelligence to detect general attack patterns much like a network IDS. An increased volume of failed login attempts, for example, could indicate someone is trying to guess a user's password. On the other hand, it could indicate that a new system was recently brought online and everyone's password expired for the first time all at the same time. The former is bad behavior while the latter is not. A good monitoring solution will give you enough information about the suspicious activity to investigate it yourself to determine if it is acceptable. More advanced products will also let you know what it believes the threat of the activity is so you can more easily assess the situation.

Exploits of known software flaws are generally well-defined, but suspicious activity is not. In addition to password attacks, there are a number of other types of activity that you may consider suspicious. Remember that insiders have more access than external hackers, and so the range of commands they can run are less limited by firewalls and network IDSes. When monitoring, choose the ones from this list that apply to your environment. Modify them or create some of your own rules to ignore good behavior and to notify you of bad behavior such as:

- Activity outside of business hours
- Unusually high number of records affected by a query or update
- Sudden reappearance of a user who has not been on the system for an extended period of time
- Activity from someone who is no longer with your company
- An unauthorized client application
- An unauthorized client machine
- An authorized user from an unusual machine or using an unusual application
- Usual queries from unusual sources (e.g., users, machines or applications). You generally want all traffic from SQLPlus monitored, but you may not care about user "weblogic" logging in from your BEA WebLogic server.

NOTE

Define signatures for SQL that you normally expect to occur and make sure it is only executed by authorized user-host-application combinations. Log all else. This will take a while to get correct. Start with something that is too permissive and whittle it down to be more restrictive as time permits. If you try to get this exactly right the first time, you will find that you are flooded with alerts about normal activity such as the backup job for which you forgot to create a SQL signature.

- Network traffic leaving the server destined for an unusual machine or port
- Unusual queries from usual sources
- Unusually high number of executions of some query (e.g., "SELECT creditcardnum FROM mytable WHERE creditcardnum > [the max credit card number selected last time] and ROWNUM <=10"). In this case, the number of records isn't high, but the attacker may execute it over and over again to get a large number of credit card numbers. Be careful to look for the general form of the query, not an exact match. In this example, the credit card number will change.

- Unusual growth rate of Oracle logs. If a particular attack will log an error to Oracle's error log, the attacker may cause a large volume of some other error to mask the one he does not want you to notice.
- Failed access to objects in the database
- Vulnerability scan being executed
- Any listener command. These typically are not executed very often, so any execution should be considered suspicious.

Notes from the Underground…

Sweeney Attack

A *Sweeney Attack*, named for Latanya Sweeney, PhD, Associate Professor of Computer Science, Technology and Policy at Carnegie Mellon University, is one where an attacker will associate seemingly innocuous data from one source with data from another source and combine them to deduce specific data points.

Substantiating her research, Sweeney successfully identified health records of former Massachusetts state governor William Weld. The data started with health insurances records of state employees. The data had several personally identifiable data points removed, including names and addresses. However, the "cleaned" data still contained each patient's five-digit zip code, gender, and date of birth. Sweeney's research claims that this is enough to identify 87 percent of the population of the United States—all that is needed is a name to go with those data points. To prove her point, she obtained publicly available voter registration data, correlated the health records, and accurately deduced which record belonged to the governor.

Large organizations often have data warehouses with aggregate data of customers, patients, and so forth. Sometimes aggregation points (the columns of a "group by" clause) may only represent a small number of records from the original data. If one of your intentions when aggregating is to hide data, take care to aggregate in such a way that one cannot discover details by comparing the aggregate with another data source. Choose a solution that can monitor your entire enterprise and correlate activity across multiple servers. This will enable you to discover abnormal patterns that span more than just one database.

Tracking the Attacker

Remember that as with any security policy, a "least privileges" activity monitoring policy permits only what is required and forbids all else. Be as precise as you can when defining what activity your monitoring system will accept as not being suspicious. Continuously adjust your policy to account for newly acceptable behavior, and expire rules defining behavior that is no longer in use. If this proves to be too difficult in your environment, this section will give you pointers on what well-defined, seemingly normal activity you should log just in case it was executed by an attacker. For example, any command that creates a user or grants privileges may be normal activity, but it may also be an attacker creating a backdoor so that he can execute commands at a later time.

It is important for you to be able to see an audit trail of activity if at some point in the future you realize you have been hacked. A complete history of everything that ever happened is too much to analyze, so use the list below as a guide of what activity is important to monitor.

- Privileged users (DBA's, sys, system)
- Ideally, your monitoring system will dynamically detect whether a user has advanced permissions and monitor him or her automatically
- Privileged actions regarding authorizations including granting/revoking permissions, creating/altering/dropping users and roles, changing ownership of objects
- Creating, changing, or dropping tables, columns, views, stored procedures, triggers, schemas, and so forth
- Configuration changes to your database, particularly those relevant to security. These include but are not limited to password expiration time, maximum number of failed login attempts before being locked out, password complexity.
- Configuration changes to and restarts of your activity monitoring solution
- Changes to an administrator password
- Truncation of redo logs, clearing of error logs, or deleting of audit trails

- All logins, successful or unsuccessful. Be sure to record not just the database user, but also the real person and the machine from which he or she connected.
- The real person may be indicated by the operating system user on the client side of the connection, but a multi-tier architecture will make the real user harder to detect. In this case, very often you will need to instrument your database monitoring solution with some knowledge of your application. For example, your application login procedure may include the query "*select account_num, last_login from AppUsers where user_id = ? and password = ?.*" Customize your monitoring solution to detect this signature and flag it as an authentication attempt.
- Changes to data related to Oracle's operations
- Copying of data from a production system to another machine whether via a backup, an export/import, or one of several forms of remote access
- Database start/stop
- One type of attack typically initiated by someone with access to the server room is to shut down the operating system and boot off of the attacker's CD containing another copy of the operating system. Using an operating system on the CD rather than the one on the hard drive gives the attacker root/administrator access to the file system. This means he or she can readily steal database files. A database stopping unexpectedly and then restarting later may be an indication of this.

TIP

Supplement your monitoring of suspicious activity with monitoring of DBA-like activity and of access to sensitive data. If you detect an intrusion, you will want to go back and investigate how it happened and what damage, if any, was done. This may include identity theft, changing financial data, or altering the host operating system. In all cases, record the exact command, the time it was executed, who did it, the source machine, and any other detail that will help you identify the person and track his or her activity.

Adhering to Government and Industry Regulations

One of the main drivers of database activity monitoring products today is the need to adhere to government and industry regulations. The general theme of the regulations we discuss below is that data must be protected whether it is financial data, personally identifiable information, or medical patient records. These are representative examples of the many standards that exist.

When building your audit policy, pay specific attention to anomalies because, speaking from experience, reviewing millions of records of expected database activity does not help you keep your data secure. For example, tracking that your application server is doing its job does not provide you any useful information regarding a data breach. In fact, making it so difficult to find anomalies helps to mask the attack. Use the guidelines above to define "expected database activity" and then log anything else that is required by the particular regulation below. Don't forget to include database servers used for development and testing purposes. They often include a copy of production data. To help performance and storage requirements, use an auditing solution that allows you to filter activity at the source before collecting it to disk.

The Sarbanes-Oxley Act

The Sarbanes-Oxley Act (SOX) revolves around the integrity of corporate financial statements. Senior executives are accountable for their accuracy and for the effectiveness of controls that guarantee this. Independent, external auditors must also review the controls periodically. They must check for areas where misstatements can occur and places where fraud can happen. From an information technology perspective, they are particularly concerned with monitoring privileged users such as database administrators.

Your activity monitoring solution must be able to detect changes to sensitive financial data as well as be able to detect all privileged users and privileged activity. The latter includes all Data Definition Language (DDL) statements, backups/restorations, and any significant configuration changes. The audit trail must identify the person who made the changes and a separate change management system should agree that those changes were permitted. Very often an internal auditor will attach to the audit trail an identifier defined by the change management system to indicate such permission.

Intrusion detection is also a key factor in protecting sensitive data and to this end, SOX explicitly requires fraud detection controls. Monitoring for attacks as described in this chapter goes a long way toward meeting this goal. Since SOX also requires reasonable measures to prevent fraud, look into getting a solution that combines activity monitoring and vulnerability assessment to yield a single SOX compliance report. The regulation goes on to say that management overrides of the controls (e.g., the administrator of the monitoring tool changing its configuration) must also be tracked.

The Gramm-Leach-Bliley Act

The Gramm-Leach-Bliley Act (GLBA) requires financial institutions to safeguard their customers' private information. The three key factors in this are risk assessment, monitoring access, and verification of the effectiveness of these controls. You will need to specify specific tables (e.g., *customer_accounts*) in the database for which database commands should be monitored. These commands include standard Database Markup Language commands (i.e., *select*, *insert*, *update*, and *delete*) as well as the use of any view, stored procedure, trigger, and so forth, which may read from or write to such tables. Monitor DDL and configuration changes to your database and to your monitoring tool.

California Senate Bill 1386

Identity theft is an enormous problem costing hundreds of millions of dollars every year. In reaction to this, California Senate Bill 1386 was created in 2003 requiring companies to disclose when they become aware that personally identifiable information has been leaked. To meet this requirement, monitor specific tables just like you do for GLBA, but in this case only *selects* are important. Include any objects such as views and stored procedures that could query those tables. Like all of the regulations in this section, monitor privileged users, privileged activity such as DDL commands, import/export (including backups), database configurations, and activity monitoring configurations.

The Health Insurance Portability and Accountability Act

The Health Insurance Portability and Accountability Act (HIPAA) includes a Privacy Rule that took effect in 2003. This ensures the security and privacy of Protected

Health Information (PHI), which includes, among other things, the patient's name, address, date of birth, social security number, medical data, payment history, account number and doctor's name. Track all unexpected access to this information regardless of it being a read or a write. Breaches incur a fine per record, so be sure to record the number of records affected by database commands. As usual, track DBAs, DDL commands, backups/restorations, and configuration changes.

The Payment Card Industry Data Security Standard

The Payment Card Industry (PCI) developed a Data Security Standard (DSS) to secure consumer information and to help prevent fraud. Any company using credit card numbers from any of the major payment card providers (American Express, Visa, MasterCard, and others) is required to adhere to this standard. Of particular interest on the topic of activity monitoring is DSS Requirement 10: "Track and monitor all access to network resources and cardholder data." Quoting directly from the PCI standards document:

> "Logging mechanisms and the ability to track user activities are critical. The presence of logs in all environments allows thorough tracking and analysis if something does go wrong. Determining the cause of a compromise is very difficult without system activity logs."

Monitoring for attacks is a critical part of this, and of course logging what else the attacker is doing is crucial, whether commands succeeded or not. Unlike any of the other standards though, PCI requires "all individual user accesses to cardholder data" to be recorded. Like the other regulations, you need to record administrator activity, configuration changes, and so forth. It also requires a daily review of audit logs, but this can be achieved by a proactive alerting system that reads your log for you. Additionally, the DSS makes specific requirements about the integrity of the audit log. It must be secure and you must monitor all access to it. This includes using change detection software such as cryptographic hashes or signatures.

Summary

There are many reasons to monitor database activity beyond just looking for known attacks. Your organization is likely to come under the auspices of one or more regulations that will require it. You may also want to use it as part of your workflow to ensure your database administrators do not stray outside the bounds of approved activity, particularly on your production systems.

The concept of "separation of duties" is not new. Best practices in accounting assert that the person who accepts money is different than the person who counts it. Likewise, database administrators who configure and control the database should not audit their own activity. Similarly, auditors should not have access to configure the database or the audit trail.

Attackers can come from many places, but recent history suggests that more likely than not they will be internal to your organization. They may be employees, contractors, or anyone else with access. Internal people even without administrative privileges generally have more access than outsiders, so it is important to monitor all database activity at its source. Watch for known attacks, detect suspicious activity, and keep track of the entirety of any session that exhibits these behaviors. Many government or industry bodies require your organization to do so in one way or another.

Solutions Fast Track

Database Intrusion 101

☑ Applications sitting in front of the database often have their own vulnerabilities, which enable malicious users to alter the SQL commands sent to the database.

☑ Oracle's HTTP Server should either be turned off or access to it should be monitored closely. It offers a freeform field to execute any database command. Be particularly careful to firewall it from the outside world.

☑ Direct database access is available to people inside the corporate firewall. Carefully monitor any unexpected database activity.

Detecting Known Attack Patterns

- ☑ Software flaws present a recurring problem in any server application. When the vendor, in this case Oracle, publishes fixes to these vulnerabilities, hackers will immediately try to exploit unpatched systems.
- ☑ Use an activity monitoring solution such as DbProtect from Application Security, Inc. to detect hackers trying to execute known attacks.
- ☑ Prepare your internal processes to be able to respond in real time to threats. This may include disabling a user, installing a fix, or blocking access from a particular client host.

Detecting Suspicious Activity

- ☑ 70 percent of attacks come from internal sources
- ☑ A "least privileges" security policy defines acceptable behavior to be specific combinations of user, client application, client machine, and SQL. Flag any activity that is not explicitly permitted as being suspicious.
- ☑ It is often too difficult to define a "least privileges" policy, so instead monitor common indicators of an attack such as off-hours activity, unusually large result sets, unusually high traffic rate, uncommon client hosts, and uncommon client applications.

Tracking the Attacker

- ☑ Monitor the username, the real person, the time of the attack, the client application, the client host, and any other contextual information relevant to your particular application.
- ☑ Monitor typical targets of attacks such as DDL statements, clearing audit trails and error logs, changes to Oracle's configuration, and access to sensitive data.

Adhering to Government and Industry Regulations

- ☑ Many government and industry regulations exist. Each has different requirements for database activity monitoring

☑ Regulations generally detect fraud and/or theft in some form. This may be altering financial statements, stealing credit card numbers, identity theft, or unauthorized access of medical records.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: I already have a network IDS. Do I need a separate one for databases?

A: Yes, network IDSes and database IDSes are complimentary technologies. Having both working for you is an ideal combination to detect generic network attacks and attacks specifically targeted at your databases which a network IDS would not detect.

Q: An Oracle NET protocol network packet sniffer cannot detect users who Telnet or SSH to the server. How do I monitor them?

A: One common solution is to install a small application that runs on the server alongside the database. There are many mechanisms to monitor a process from another process on the same machine.

Q: Will an application installed on the database's host affect performance?

A: That depends on the technical details of the solution. Consult its creators (your in-house engineers, a software vendor, and so forth) for details. Always test in a non-production environment having production traffic (i.e., the same commands and traffic volume).

Q: My manager is telling me I need to monitor everything, but either I do not have enough storage space, or it takes too much time to report on such a large volume of data, or my activity monitoring cannot keep up the pace. How do I solve this problem?

A: To my knowledge, no regulation requires you to monitor everything. Take a step back from the problem you have been asked to solve and determine why you need to solve it. You will discover that you do not need to monitor everything, but you do need to detect intruders, to audit administrators, or to track unauthorized access to sensitive data. Assuming this is true, use an activity monitoring tool that can filter traffic before it stores it.

Q: The SOX auditor at my company is asking for different things than the auditor at my friend's company. Why is that?

A: The SOX is vague in its technical requirements. In recent years a standards body called the Public Company Accounting Oversight Board (PCAOB) has begun to create technical guidelines for auditors, but they are still in the early phases of this work[4].

1 "Enterprise Databases Need Greater Focus to Meet Regulatory Compliance Requirements" by Noel Yuhanna, Forrester Research, January 24, 2007.

2 www.imsdb.com/scripts/Harry-Potter-and-the-Goblet-of-Fire.html

3 The sysman password was in clear text in a world-readable file in earlier versions of 10g: www.ngssoftware.com/advisories/oracle23122004D.txt

4 http://www.pcaob.org/Rules/Docket_021/2007-05-24_Release_No_2007-005.pdf

Chapter 9

Implementation Guide

Solutions in this chapter:

- Getting Started
- Implementing Basic Security
- Implementing Best Practices
- Locking Down Your Database

☑ Summary

☑ Solutions Fast Track

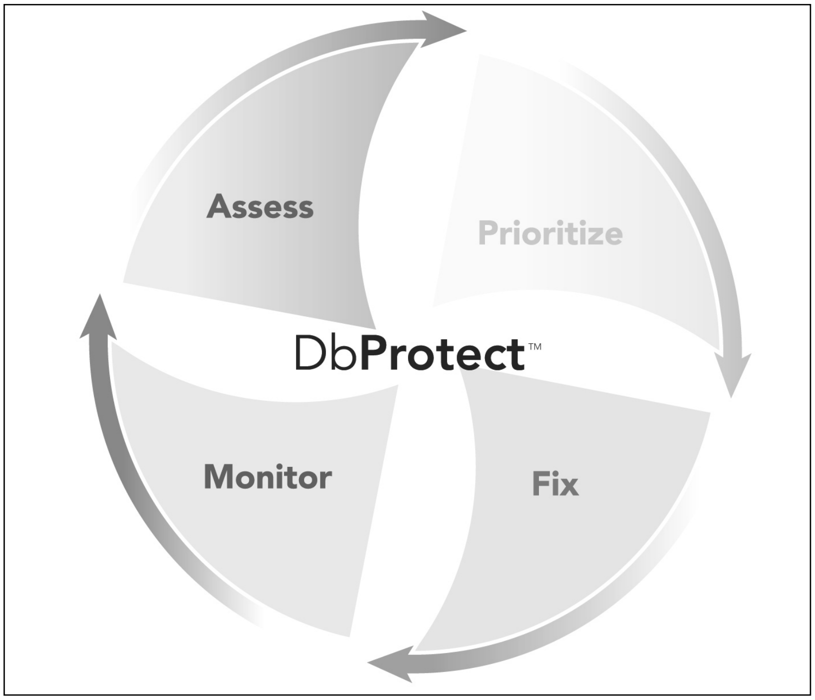
☑ Frequently Asked Questions

Introduction

The solutions presented in the book are a lot to digest. Whether you need to lock down one database or ten thousand, it is a daunting job to get all of the right people involved, consider all of the variables and develop a plan. This chapter is dedicated to the latter: developing a plan to secure your databases and to keep them secure.

The core tasks in this effort start with identifying all of your databases, assessing their vulnerabilities, monitoring them for intrusions especially if you know your systems are vulnerable, selecting which fixes come first, and then implementing them. Once you have done this cycle of work, repeat it to discover new systems on your network, to find new vulnerabilities, etc. Figure 9.1 summarizes these concepts:

Figure 9.1 The Database Security Lifecycle (Used with Permission from Application Security, Inc.)



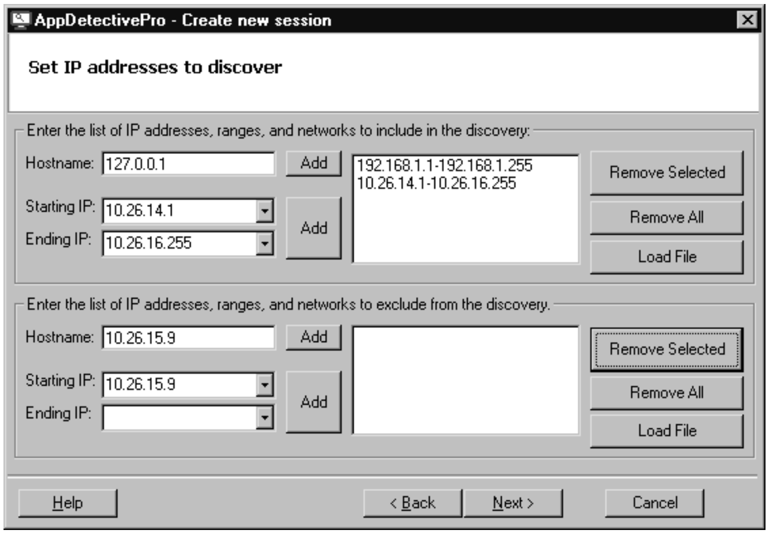
In the following sections, we will review a three-stage evolution of your security posture starting from Basic Security to Best Practices and finally to Lock Down. The guiding principle behind these stages is to start with the low-hanging fruit; the steps that are easiest to implement and provide the most value. We will then move on to best practices used by organizations which have integrated database security into their overall Information Technology effort. These enterprises have gained experience with the most effective ways to improve security. Finally, we will discuss steps that lock down your data to give you optimal security, generally requiring additional implementation effort.

Getting Started

As with any large undertaking, the first step is to assess your present situation so you can plan to work from there toward your goal. If you are saving money for a new home, you will want to know how much you have already saved. With database security, start with listing all of your databases and the problems they have.

There are several methods for generating a list of databases. Of course you can do so manually—you may already know the databases for which you are responsible. Expect however that database servers will seemingly crop up out of nowhere, especially in large enterprises. Maybe a developer needed to try something on a clean installation of Oracle or maybe your "business continuity" group just brought online a new disaster recovery site. Sometimes desktop applications come with a lightweight backend database that the application's installer does not properly secure. I suggest investing in an automated port scanning tool, particularly one designed to detect Oracle, not just any open port. Using it will yield a list of databases, probably more than you anticipated. As an example, Figure 9.2 shows the database discovery screen in AppDetective Pro.

Figure 9.2 AppDetective Pro's Database Discovery


AppDetectivePro - Create new session

Set IP addresses to discover

Enter the list of IP addresses, ranges, and networks to include in the discovery:

Hostname: 127.0.0.1
Starting IP: 10.26.14.1
Ending IP: 10.26.16.255

192.168.1.1-192.168.1.255
10.26.14.1-10.26.16.255

Enter the list of IP addresses, ranges, and networks to exclude from the discovery.

Hostname: 10.26.15.9
Starting IP: 10.26.15.9
Ending IP:

Once you have a list of all of your databases, prioritize them in order of value to your company. Generally start with production systems that would have the most impact to your business if they were to be hacked. Don't forget non-production systems though. Development and test machines serve the basis for production systems.

WARNING

At this stage in the lockdown process, make no assumptions about how your database might be hacked or whether or not it can be hacked. Prioritize your systems assuming they are all equally vulnerable. You have not done a vulnerability assessment yet so you must assume the worst. In my experience, database administrators are either not aware of the possibility of internal hackers or they do not believe they are a real threat. They will prioritize a database that serves as the backend of an externally facing application before an internal database. This method leaves the latter system vulnerable without regard to its importance to the business.

The next two steps can be merged or done separately at your option: check for vulnerabilities and fix them. This is the core of database security. Start with your top priority databases and follow the Basic Security plan below. Most of the steps are straightforward, such as removing access to particular files. There are implied steps for some of them though. For example, "lock user accounts that are not in use," implies determining which accounts are in use, but once that's done, the rest is a simple script to lock those accounts or to use an impossible password hash.

Once your important systems are at the first stage of security, you can opt to continue securing them or move on to less important systems to get them to basic compliance. This is your decision but be careful not to leave even the lowest priority database untouched for long. Oftentimes, non-production systems have security holes that yield access to production systems. An all too common example of this is using the same password in a live environment as was used while developing the application. This implies your developers have access to your real data.

Recheck for vulnerabilities periodically. Just because you locked down your system once does not mean it stays locked down. Software updates such as patchsets and Critical Patch Updates may change your settings. DBAs, application developers or operating system administrators may change things just to get around some problem. Oftentimes whatever solves a problem first is the solution that gets implemented …

without regard to its security impact. Also, you may add new steps to your lockdown plan or add new, more accurate techniques to check for existing problems.

TIP

Use an automated assessment tool if you have a lot of databases or you do not have time or resources to check manually for security issues.

Any change you make to your database requires testing; even the slightest change can possibly disrupt the proper functioning of Oracle in your environment. A seemingly innocuous file permission fix that is a clear security requirement may break some important script that for some reason relies on insecure permissions.

Implementing Basic Security

Basic Security involves the easiest fixes that give you optimal results. The idea here is to get your database to a reasonably secure state without much effort or risk. The most egregious problems luckily are the easiest to fix. These fixes will block the most common attacks and the reconnaissance leading up to attacks. Install the latest patchset, set a listener password and change default passwords and you are halfway there!

The steps listed below summarize concepts found elsewhere in this book. Use this list as a guide to implementing Basic Security and refer to other chapters for more information:

- Install the latest patchset
    - A patchset includes security fixes from CPUs released since the previous patchset so this is a good first step when you are getting started with security.
- Restrict access to configuration files
    - Examples of these files are listener.ora, sqlnet.ora, tnsnames.ora, and control files
    - Only the Oracle user on the host should have access; remove all access from other users

  - Note if you have local users on the host, they can have their own copy of tnsnames.ora and client software. This allows you to lock them out of the server's copy thus reducing risk.
- Turn off unneeded services, e.g. ext proc, http server
  - Unnecessary services are unnecessary risks. Also, services that are not used tend to be the least secure because no one is paying attention to them.
- Assign a password to the listener and lock down access to it according to your database security policy
  - For example, you may want to restrict network access to administrative commands, but the flip side of this is that database administrators must be given operating system access to run them.
- Lock accounts that are not in use
  - … or even better, use an *impossible password*, a password hash that cannot possibly be constructed from a valid password. See the "Passwords and Password Controls" chapter for more information.
- Revoke permissions to PUBLIC that are not explicitly required
  - To start, ensure PUBLIC is not a member of any role.
  - PUBLIC should generally not have DDL access, but this varies based on each database's requirements
- Remove unneeded permissions to roles and to users
- Change default passwords
  - This is almost always the first attack vector. Personally, just for fun I always try logging in as **scott/tiger** if I have network access to a database. It is really amazing how often that works. Then I can use any of a number of privilege escalation hacks to give myself DBA rights.
- Remove access to Oracle executables
  - Only the Oracle user should have read, write and execute permissions to most executables.
  - You may want to leave on read and execute permissions on some executables such as sqlplus, but do so only for the DBA group, not for everyone.

    - See the "File System" chapter for more information.
- Remove all access to executables that are not in use, even from Oracle
    - "oracle0" and other backup copies of executables are prime examples of this.
    - Often extproc and other Oracle features are not used, so turn off all access to them as well.
- Implement minimal permissions to important directories such as audit_dest_dir and user_dump_dest. See the "File System" chapter for more information.
- Remove all access to datafiles and redo logs from users other than Oracle
- Set minimal permissions on backups and exports
- Set umask appropriately ("File System" chapter)
- Error and trace logs should only be accessible to database administrators

Implementing Best Practices

Best Practices reflect proven security processes in real world environments. These steps enhance Basic Security by making it even more difficult for an attacker to penetrate your system, it helps track user activity and it implements processes that make security inherently better:

- Follow the password safety recommendations mentioned in the "Passwords and Password Controls" chapter
    - This step includes locking out accounts with too many failed login attempts. Be careful with this one though. If a middle tier application server uses a pool of connections all logged in as "bob" and then bob gets locked out, your system effectively goes down.
    - This also includes using a password verification function to validate adequate password complexity. Be careful with this one, too, for two reasons. First, requiring new passwords to be complex does not imply old ones must be. Use the tools mentioned in the password chapter to find easily guessed passwords. Second, too much security can backfire. You may require passwords to be so complex that your users cannot remember

them. You don't want to walk around the office and see a sticky note with a DBA's password on his/her monitor.

- Turn off access to OS resources
  - Packages such as UTL_FILE, UTL_HTTP, UTL_TCP, UTL_UDP and UTL_SMTP give access to the host's operating system including the file system and network. An attacker can use UTL_SMTP for example to send himself an email with data.
- Encrypt data at rest
  - Backups, exported data and laptops contain important data that is not controlled by your secure database. Use encryption to protect it from prying eyes.
- Audit privileged users and known attack patterns ("Database Activity Monitoring" chapter)
- Separation of duties: An auditor should monitor privileged users such as DBAs, and DBAs should not have access to the audit trail. A third party should verify this process is in place to ensure, for example, that the auditor is a different person from the DBA ("Database Activity Monitoring" chapter)
  - A relatively new Oracle feature called Data Vault helps to implement this requirement
- Install all CPUs and security alerts
  - Because CPUs are released quarterly, you will need a predefined testing and deployment process in place to be able to keep up the pace. Use as much automation as possible to speed this process along. CPU release dates are available a year in advance and the effort to evaluate them, to test them and to deploy them is predictable. This enables you to plan resources (people, hardware, software) effectively and schedule time every quarter for this effort.

NOTE

I have seen companies get the CPU implementation process down to a week. I have even heard of companies that do this in 24 hours across the enterprise, but I have not witnessed this personally.

Locking Down Your Database

Lock Down is the final stage of database security. Beyond best practices, these steps go the extra mile to monitor the most remote threats and to prevent even accidentally revealing private data. The steps here are advanced, they require training and research, and they will generally affect your operating environment. Expect significant lead time before being able to deploy them:

- Don't permit direct access to host, even for DBAs, unless necessary. Monitor all activity where the client is local to the server.
  - This follows the standard secure practice of "least privileges." Most DBA activity can be done via a remote client and does not require local access. Local access implies access to the file system which is more permissive than is necessary for most tasks. An operating system administrator can perform many of the tasks necessary, and to prevent his/her access to the data, use encryption.
- Oracle Advanced Security (OAS) encrypts data at rest and in transit thus maintaining confidentiality and integrity of your data.

WARNING

Be aware that many third-party auditing products are implemented by sniffing the network. Depending on which product you choose and how it is configured, using network encryption may impede your ability to audit. Also, the standards which require network encryption generally only require this for external networks. Virtual Private Networks (VPNs) typically take care of this requirement nicely.

- Build on your Best Practice auditing. Include suspicious activity, access to sensitive data and access to the auditing tool. See the "Database Activity Monitoring" chapter for details.
  - Activity monitoring at this level requires a commitment to analyze its results. Suspicious activity, for example, may or may not be a problem. You will need to implement a workflow where someone investigates and acknowledges security alerts.
  - Expect advanced auditing of this type to generate a significant volume of results. Plan storage space accordingly.
- Restrict access to individual records within tables
  - Virtual Private Databases and Label Security are technologies related to access control on a row-by-row basis. You set access privileges to particular rows or to rows meeting a rule that you define. Using technologies built into the database is more secure than doing so at the application layer.
- Use centralized authentication
  - Separate the data from the mechanism of accessing the data, i.e. use LDAP, Kerberos or some other authentication process that integrates with Oracle. The TJX hack mentioned in the first chapter was enabled in part because encryption keys were stored in the database. This step serves to remove the function of access control from the database administrators and puts it into the hands of a separate security administrator.
  - Storing passwords in clear-text in application server configuration files is a common security problem which can be solved by using centralized authentication.
  - Single Sign On (SSO) features are very often linked to centralized authentication. This will help you detect the real user behind an Oracle connection. Seeing only a connection pool's proxy user is often a problem in database auditing.
- Hire hackers to try to break into your systems. Some organizations choose to have specially trained internal staff for this and some of them outsource. Either way, you cannot be sure your security posture is working unless you test it.

- Review error logs daily for anomalies. This is a requirement for Payment Card Industry (PCI) Data Security Standards (DSS) compliance.
- Configure the listener to validate hosts that attempt to access the database. Generally, the client machines which are allowed to access a database are known in advance so disallow all others.
- Generate strong cryptographic hashes of the executables and other static files on the host, especially those belonging to Oracle. Periodically verify that the hashes have not changed.

Summary

The key points to take away from this chapter are 1) tackle the easy fixes that have a major impact first, and 2) set major milestones that concisely show the state of database security in your enterprise. When a Chief Security Officer or Chief Information Officer asks you for status you can give a clear response if you have well-defined milestones. You can report, for example, that of 6,218 databases, 95% have cleared a Basic Security audit, 27% have gone further with Best Practices implemented and 3% are fully Locked Down from intrusion. Then you can break this down by priorities such as "Top Tier," for high-priority databases, "Middle Tier" and "Bottom Tier" giving the percent of each tier that is at each of the three security stages.

Some organizations choose to alter the definition of the three stages slightly: anything that requires extra effort in Basic Security gets moved to Best Practices. For example, accounts that are clearly unused get locked in the former stage, and then once the rest have been evaluated they will be locked in the latter. It is your choice how you organize your plan. Use the steps in this chapter as a starting point and customize them as you feel necessary.

Solutions Fast Track

Getting Started

- ☑ Start with a list of databases to secure
- ☑ Seek out databases of which you may not be aware
- ☑ Prioritize based on business value
- ☑ Lock down each in turn, starting with Basic Security. Consider preferring Basic Security for more systems over fully locking down fewer systems.
- ☑ Test any change
- ☑ Recheck for problems periodically

Implementing Basic Security

- ☑ Set proper file permissions

- ☑ Remove components that are not being used
- ☑ Set a listener password
- ☑ Set proper user privileges
- ☑ Update to the latest patchset

Implementing Best Practices

- ☑ Implement password controls
- ☑ Disable all access to operating system resources
- ☑ Audit privileged user activity
- ☑ Monitor for known attack patterns
- ☑ Separate the various duties of the traditional DBA into a read-only auditor and a user with read/write access to everything except the audit trail
- ☑ Keep up with Critical Patch Updates
- ☑ Encrypt data at rest, e.g. backups and data exports

Locking Down Your Database

- ☑ Restrict access to the host as much as possible. Monitor everything the user does if host access is granted.
- ☑ Use centralized password management
- ☑ Use Virtual Private Database to restrict access to records at the database level.
- ☑ Restrict listener access to only approved hosts
- ☑ Review error logs daily
- ☑ Hire database security professionals on a periodic basis to attempt to penetrate your defenses
- ☑ Generate and check file hashes to ensure they have not changed over time

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How long will it take me to implement Basic Security on a database?

A: The answer to this question depends on several factors. First, determine how much work is left to meet the requirements of Basic Security. This assessment can take a matter of minutes with an automated tool that is configured properly, e.g. it knows which users should exist and what permissions they should have. If your database already meets the requirements, then there is no further work to do. Fixing the problems should take a few hours and then testing can take as little or as long as you like.

Q: My experience is that Oracle's own auditing methods impact database performance. What are some alternatives?

A: There are many auditing technologies available. A popular one is to read network packets via a separate machine with special access to the database's network. Another is to read Oracle's System Global Area (SGA) for SQL commands.

Q: I want to adhere to a policy of "least privileges," but I do not have a complete list of users and the objects to which they need access. How do I create this list?

A: One way is to consult with all of the stakeholders in the server and get each of them to tell you what access they need with the understanding that anything they do not mention will be disabled. Another method is to turn on auditing for some period of time, let's say a month, a watch who is accessing the system and what they are accessing. Get sign off from the stakeholders that this list is accurate.

Index