



APPLICATION
SECURITY, INC.

Protecting Oracle Databases

White Paper

APPLICATION SECURITY, INC.

WEB: WWW.APPSECINC.COM

E-MAIL: INFO@APPSECINC.COM

TEL: 1-866-9APPSEC • 1-212-420-9270

INTRODUCTION

One of the more recent evolutions in network security has been the movement away from protecting the perimeter of the network to protecting data at the source. The reason behind this change has been that perimeter security no longer works in today's environment. Today, more than just your employees need access to data. It's imperative that partners and customers have access to this data as well. This means that your database cannot simply be hidden behind a firewall.

Of course, as your databases become more exposed to Internet, it is imperative that you properly secure it from the threats and vulnerabilities of the outside world. Securing your database involves not only establishing a strong password policy, but also establishing adequate access controls. In this paper, we will cover various ways databases are attacked, and how to prevent them from being “hacked.”

CURRENT ORACLE SECURITY ENVIRONMENT

It is very easy in the security community to create an air of fear, uncertainty, and doubt (FUD). As both security and Oracle professionals, it's important to see through the FUD, determine the actual risks, and investigate what can be done about the situation. The truth is that most Oracle databases are configured in a way that they can be broken into relatively easily. This is not to say that Oracle cannot be properly secured – only that the information to properly lock down these databases has not been made available, and that the proper lockdown procedures have not been taken.

On the other hand, the number of Oracle databases compromised so far has not been nearly on the scale that we have seen web servers being attacked and compromised. The reasons for this are several.

- There are less Oracle databases than web servers.
- Knowledge of Oracle security has been limited.
- Getting a version of Oracle to learn and test on was difficult.
- Oracle was traditionally behind a firewall.

All of the above have changed significantly over the past year.

First, there is an increasing interest for databases in the Black Hat hacker community. The number of talks on database security has grown significantly over the past two years at the infamous Defcon (<http://www.defcon.org/>) and Black Hat (<http://www.blackhat.com/>) conferences. The number of exploits reported on security news groups such as SecurityFocus™ (<http://www.securityfocus.com>) has increased tenfold over the last year.

Downloading Oracle's software has also become much simpler. The latest version is available for download from the Oracle web site for anyone with a fast enough Internet connection, and the installation process has become increasingly simple in design.

The increasing threats against Oracle databases are not going to cause the end of the world. However, it is necessary for us to start taking database security seriously by taking a proactive approach to understand the risks and lockdown procedures.

WHY SHOULD I CARE ABOUT ORACLE SECURITY?

The most common point of network attack is the web server, and other devices connected directly to the Internet. Usually these programs do not store a company's most valuable assets. The biggest issue from a defaced web site is usually the publicity, and "loss-in-trust" of the company's customers.

A hacked database is an entirely different story. Databases store a company's most valuable assets – credit card information, medical records, payroll information, and trade secrets. If your database is compromised, it could have serious repercussions on the viability of your business.

Security is also about the weakest link. Your network is only as secure as the weakest computer on the network. If you have a secure network with an insecure database, the operating system or other devices on the network can be attacked or compromised through the database. Databases should not provide a point of weakness.

Databases are easy targets. Database security is not nearly as robust as operating system security. They have all the most powerful features, such as certificates, LDAP integration, token card access, etc., but the protocol stack has not been closely examined. They are still vulnerable to simple attacks and have many weaknesses.

Also, Oracle databases have become the backbone of many web server applications. They are becoming more Internet-enabled, which means that they are open to the world of "bad guys." This is especially the case with Oracle9i Application Server, which is being pushed heavily by Oracle.

UNDERSTANDING VULNERABILITIES

In order to understand vulnerabilities, we should start by describing the various classes of vulnerabilities:

- Vendor bugs
- Poor architecture
- Misconfigurations
- Incorrect usage

VENDOR BUGS

Vendor bugs are buffer overflows and other programming errors that result in users executing the commands they are show not be allowed to execute. Downloading and applying patches usually fix vendor bugs. To ensure you are not vulnerable to one of these problems, you must stay aware of the patches, and install them immediately when they are released.

POOR ARCHITECTURE

Poor architecture is the result of not properly factoring security into the design of how an application works. These vulnerabilities are typically the hardest to fix because they require a major rework by the vendor. An example of poor architecture would be when a vendor utilizes a weak form of encryption.

MISCONFIGURATIONS

Misconfigurations are caused by not properly locking down Oracle. Many of the configurations options of Oracle can be set in a way that compromises security. Some of these parameters are set insecurely by default. Most are not a problem unless you unsuspectingly change the configuration. An example of this in Oracle is the REMOTE_OS_AUTHENT parameter. By setting REMOTE_OS_AUTHENT to true, you are allowing unauthenticated users to connect to your database.

INCORRECT USAGE

Incorrect usage refers to building applications utilizing developer tools in ways that can be used to break into a system. Later in this paper, we are going to cover one example of this – SQL Injection.

LISTENER SERVICE

A good place to start delving into Oracle security is the Listener service - a single component in the Oracle subsystem. The listener service is a proxy that sets up the connection between the client and the database. The client directs a connection to the listener, which in turn hands the connection off to the database.

One of the security concerns of the listener is that it uses a separate authentication system. It is controlled and administered outside of the database, and runs in a separate process under the context of a privileged account such as 'oracle'. The listener accepts commands and performs other tasks besides handing connections to the database.

LISTENER SECURITY IS NOT DATABASE SECURITY

Why is the separation of listener and database security a potential problem? There are a few reasons.

First, most people do not even realize that a password must be set on the listener service. The listener service can be remotely administered just as it can be administered locally. This is not a feature that is clearly documented, and is not well known by most database administrators.

Secondly, setting the password on the listener service is not straightforward. Several of the Oracle8i versions of the listener controller contain a bug that causes it to crash when attempting to set a password. You can manually set the password in the "listener.ora" configuration file, but most people do not know how, nor do they have any idea that they should. The password itself is either stored in clear text or as a password hash in the listener.ora file. If it's hashed, setting the password in the listener.ora file manually cannot be performed. If it is in clear text, anyone with access to read the \$ORACLE_HOME/network/admin directory will be able to read the password.

KNOWN LISTENER PROBLEMS

So what are the known problems with the listener services? To investigate these problems, let us pull up the listener controller, and run the 'help' command. This gives us a list of the commands that we have at our disposal.

To start the listener controller from UNIX, enter the following command at a UNIX shell:

```
$ORACLE_HOME/bin/lsnrctl
```

To list the commands available from the listener controller, run the following command at the listener controller prompt:

```
LSNRCTL for 32-bit Windows: Version 8.1.7.0.0 - Production on 04-JUN-2001  
10:42:14
```

```
(c) Copyright 1998 Oracle Corporation. All rights reserved.  
Welcome to LSNRCTL, type "help" for information.
```

```
LSNRCTL> help  
The following operations are available  
An asterisk (*) denotes a modifier or extended command:
```

start	stop	status
services	version	reload
save_config	trace	db snmp_start
db snmp_stop	db snmp_status	change_password
quit	exit	set*
show*		

Notice within the above example, two of the commands with the asterisks after them – set and show. We can list the possible extended commands for these commands as well:

```
LSNRCTL> help set
```

password	rawmode	displaymode
trc_file	trc_directory	trc_level
log_file	log_directory	log_status
current_listener	connect_timeout	startup_waittime
use_plugandplay	save_config_on_stop	

Note the command 'set password'. This command is utilized to log us onto a listener. There are a couple of problems with this password:

- ❑ There is no lockout functionality for this password
- ❑ The auditing of these commands is separate from the standard Oracle audit data
- ❑ The password does not expire. Basically, there are no password management features for the listener password.

This means that it is not very difficult to write a simple script to brute force this password even if a strong password is being utilized.

Another problem is that the connection process to the listener is not based on a “challenge-response” protocol. Basically, whatever you send across the wire is in clear text. Of course, if you look at the traffic you might notice that a password hash is sent across the wire, but this password hash is actually a password equivalent - and knowledge of it is enough to login.

So what can a hacker accomplish once they have the listener password? There is an option to log the data sent to the listener to an operating system file. Once you have the password, you can set which file the logging data is written, such as .profile, .rhosts, or autoexec.bat. Below is an example command sent to the listener service:

```
CONNECT_DATA=(COMMAND=ping)
```

Instead a hacker can send a packet containing a maliciously constructed payload such as below.

- ❑ "+ +" if the log file has been set to .rhosts
- ❑ "\$ORACLE_HOME/bin/svrmgrl" followed by "CONNECT INTERNAL" and "ALTER USER SYS IDENTIFIED BY NEW_PASSWORD" if the log file has been set to .profile.

Oracle released a patch for this issue, which basically provides a configuration option you can set which will not allow parameters to be reloaded dynamically. By setting the option, you disable a hacker's ability to change the log_file value. Of course if you do not set this option, this problem is not fixed. By default, this option is not set. It is essentially the database administrator's responsibility to recognize and fix this problem.

TNS LEAKS DATA TO ATTACKER

Another problem with the listener service is that it leaks information. James Abendschan first made this problem public. A full description can be found at:

<http://www.jammed.com/~jwa/hacks/security/tnscmd/tns-advisory.txt>

The format of a listener packet resembles the following:

```
TNS Header - Size of packet - Protocol Version - Length of Command - Actual Command
```

If you create a packet with an incorrect value in the 'size of packet' field, the listener will return to you any data in its command buffer up to the size of the buffer you sent. In other words, if the previous command submitted by another user was 100 characters long, and the command you send is 10 characters long, the first 10 character will be copied over by the listener, it will not correctly null terminate the command, and it will return your command plus the last 90 characters of the previous command.

For example, a typical packet sent to the listener looks like the following:

```
.T.....6.,.....:.....4.....(CONNECT_DATA=.)
```

In this case we are sending a 16-byte command – (CONNECT_DATA=.) . One of the periods is actually the hex representation of the value 16, which indicates the command length. Instead we can change 16 to 32 and observe the results. Below is the response packet:

```
....."....(DESCRIPTION=(ERR=1153)(VSNNUM=135290880)(ERROR_STACK=(ERROR=(CODE=1153)(EMFI=4)(ARGS='(CONNECT_DATA=.)ervices))CONNECT'))(ERROR=(CODE=303)(EMFI=1)))
```

The return packets indicate that Oracle does not understand our command, and the command it does not understand is returned in the ‘ARGS’ value seen in the above example. Notice that the ‘ARGS’ value is as follows:

```
(CONNECT_DATA=.)ervices))CONNECT
```

The ‘ARGS’ value returned our command plus an additional 16 characters. At this point it is not clear what the last 16 bytes are. So we then try to “up the lie” and tell the listener our command is 200 bytes long. Below is the return value we get from the listener:

```
.....">.H.....@(DESCRIPTION=(ERR=1153)(VSNNUM=135290880)(ERROR_STACK=(ERROR=(CODE=1153)(EMFI=4)(ARGS='(CONNECT_DATA=.)ervices))CONNECT_DATA=(SID=orcl)(global_dbname=test.com)(CID=(PROGRAM=C:\Oracle\bin\sqlplus.exe)(HOST=host123)(USER=user1))) (ERROR=(CODE=303)(EMFI=1)))
```

Notice this time the ‘ARGS’ parameter is a little longer:

```
(CONNECT_DATA=.)ervices))CONNECT_DATA=(SID=orcl)(global_dbname=test.com)(CID=(PROGRAM=C:\Oracle\bin\sqlplus.exe)(HOST=host123)(USER=user1))
```

Now it is a bit clearer what is being returned – previous commands submitted by other users to the database. Notice that the HOST and USER name of the other user is displayed in this buffer.

This information is useful to an attacker in several ways. It can be used to gather a list of database usernames. An attacker can continually retrieve the buffer, and in a matter of a few days, retrieve a list of all the users that have logged in during that time. Now imagine if the database administrator logs into the database using the listener password of which you can retrieve from the buffer.

This problem has been fixed in the latest patch sets (patchset 2 for Oracle version 8.1.7). It is also a good idea to deal with this problem by limiting access to connect to Oracle using a firewall or another packet filtering device.

BUFFER OVERFLOW IN LISTENER

Using the same techniques from the previous vulnerability, we can send a large connection string to the listener. If the packet contains more than 1 kilobyte of data, the listener crashes. Using a connection

string of 4 kilobytes results in a core dump. The following is an example of what this packet would look like:

```
.T.....6.,.....:.....4..... (CONNECT_DATA=  
XXXXXXXXXXXXXXXXXX<snip>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/0x12/0x54/0x5/0x34/0x12/0x  
54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x  
54/0x5/0x34/0x12/0x54/0x5/0x34)
```

In the above example, we have clipped most of the “Xs”. The funny characters at the end of the command are opcodes. Opcodes are low-level machine commands used by the hacker to inject commands that will run on the database. By overflowing the stack with all the Xs, an attacker can cause the execution of arbitrary code by manipulating the SEH (Structured Exception Handling) mechanism.

This vulnerability caused quite a stir in June and July of 2001. Patches were not released immediately for all platforms, and not at all for “non-terminal” release versions of Oracle. This means a lot of databases are still very vulnerable.

TUNNELING ORACLE THROUGH A FIREWALL

One common question that we hear is the following: “How do I tunnel Oracle connections through a firewall?” In addressing this question, we will discuss the following:

- Why is it so difficult to get Oracle traffic through a firewall?
- Why would you want to do this?

After we address these points, we will then ultimately describe how to tunnel Oracle through a firewall.

It is difficult to tunnel traffic through a firewall because Oracle handles traffic much like FTP does. You make a request for a connection to port 1521, the listener receives the connections, negotiates with the database to listen on a new port greater than 1024, and then replies back to the client with the new port number. The client then connects directly to the database over the new port.

The challenge with tunneling Oracle traffic is that Oracle connections are redirected to a second port after making the initial connection to the listener. By default, connections are made to port 1521. The listener receives a connection request, matches up the SID you are looking for with one of the SIDs it is listening for. It then picks a new port for the database and the client to connect on, after which it requests the database to start listening on the new port. The new port then returns to the client. The client then connects directly to the database on that new port. The complication is that the new port is dynamically chosen for any of the ports over 1024. Opening the firewall to any port over 1024 defeats the purpose of even having a firewall in place.

Note that not all versions of Oracle communicate in the fashion. Some versions of Oracle on UNIX use port 1521 for the entire process. In this case, you will not have a problem tunneling through a firewall, and all you have to do is simply open up port 1521. This is not the case for Oracle on Windows or Oracle configured in Multi-threaded server mode on UNIX.

YOU HAVE BEEN WARNED!

One strong recommendation is that you SHOULD NOT expose Oracle directly to the Internet. There are some very powerful arguments on why you should not do this. Still, when the business reasons outweigh the risk, you may need to expose Oracle to the Internet. In this case, you can mitigate the risk by selectively opening access. For instance, if you must open port 1521 for a partner to connect to a specific database, the firewall administrator can open up only the ports for the specific IP destinations and for the source IP address of the partner.

The reasons not to expose Oracle directly to the web are as follows:

- The TNS protocol is vulnerable to a wide array of attacks
- Oracle listener and database passwords can be brute-forced

A better solution to expose your database to the network is to integrate your database into a web application, such as Oracle9i Application Server, with a web front end. Place the Web server and the database in the DMZ, and block connections from outside the DMZ to the database. Specifically set the database to only accept connections from the web server.

OPENING A HOLE IN YOUR FIREWALL

Several firewall vendors support transparent proxying of the Oracle protocol. Proxying works by understanding the Oracle protocol, watching the listener return the new port to connect on, and temporarily opening that port.

The vendors that support this functionality are:

- Cisco PIX
- Symantec's Raptor Firewall
- CheckPoint Firewall One

There are other ways to pipe Oracle connections through a firewall. Connection manager is an Oracle application that translates from one protocol to another. It can also be used to get connections through the firewall. Connection manager listens by default on port 1610. If you configure your clients to use the connection manager, place the connection manager behind the firewall, and then cause the connection manager to forward the TNS protocol to the database, the problem with redirection is eliminated. The port redirection then occurs between the database and the connection manager all behind the firewall.

There are also some options to use in Oracle that can stop the redirection process. One possibility is to set the `USE_SHARED_SOCKET` parameter in the registry on Windows NT or Windows 2000. Another possibility is to force the multi-threaded server to use a range of specific ports. This is achieved by configuring the `MTS_DISPATCHER` parameter in the "init.ora" file. The last option is to set the `SERVER=DEDICATED` option in the client "tnsnames.ora" file. This last option is not recommended since it hurts the scalability of Oracle.

Another solution is to tunnel all Oracle traffic through SSH. There is an interesting article on this on the Database Specialists website, <http://www.dbspecialists.com/>. The advantage of this strategy is that the protocol is also encrypted in this fashion.

SQL INJECTION

Just because your database is behind a firewall does not mean that you do not need to worry about it being attacked. There are several other forms of attacks that can be executed through the firewall. The most common of these attacks today is SQL Injection. SQL Injection is not an attack directly on the database. SQL Injection is caused by the way web applications are developed. Unfortunately, since you are trying to protect the database, you need to be aware of these issues, know how to detect them, and fix the problems.

SQL Injection works by attempting to modify the parameters passed to a web application to change the SQL statements that are passed to the database. For instance, you may want the web application to select from the ORDERS table for a specific customer. If the hacker enters a single quote into the field on the web form, and then enters another query into the field, it may be possible to cause the second query to execute.

The simplest way to verify whether or not you are vulnerable is to embed a single quote into each field on each form and verify the results. Some sites will return the error results claiming a syntax error. Some sites will catch the error, and not report anything. Of course, these sites are still vulnerable, but they are much harder to exploit if you do not get the feedback from the error messages.

This attack works against any database, not just Oracle. How this attacks works varies slightly from database to database, but the fundamental problem is the same for all.

SQL INJECTION SAMPLE1

So how does the exploit work?

SQL Injection is based on a hacker attempting to modify a query, such as:

```
Select * from my_table where column_x = '1'
```

to:

```
Select * from my_table where column_x = '1' UNION select password from DBA_USERS where 'q'='q'
```

In the preceding example, we see a single query being converted into 2 queries. There are also ways to modify the 'WHERE' criteria to update or delete rows not meant to be updated or deleted. With other databases you can embed a second command into the query. Oracle does not allow you to do this. Instead an attacker would need to figure out how to supplement the end of the query. Note the 'q' = 'q' at the end. This is used because we must handle the second single quote that the ASP page is adding onto the end of the page. This clause simply evaluates to TRUE.

Here is an example of a Java Server Page that you might typically find in a web application. Here we have the case of a typical authentication mechanism used to login to a web site. You must enter your password and your username. Using these two fields we get a SQL statement that selects from the tables where the username and password match the input. If a match is found, the user is authenticated. If the recordset in our code is empty, then an invalid username or password must have been provided and the login is denied. Of course, a better idea would be to use the authentication built into the web server, but this form of "home grown" authentication is very common:

```
Package myserverlets;  
<...>  
String sql = new String("SELECT * FROM WebUsers WHERE Username='" +  
request.getParameter("username") + "' AND Password='" +  
request.getParameter("password") + "'")  
stmt = Conn.prepareStatement(sql)  
Rs = stmt.executeQuery()
```

Exploiting the problem is much simpler if you can access the source of the web page. You should not be able to see the source code, however there are many bugs in most of the common web servers that allow an attacker to view the source of scripts, and we assume that there are still many that have not been discovered.

The problem with our ASP code is that we are concatenating our SQL statement together without parsing out any single quotes. Parsing out single quotes is a good first step, but it's recommended that you actually use parameterized SQL statements instead.

For the following web page, I set the username to:

Bob

I also set the password to:

Hardtoguesspassword

The SQL statement for these parameters resolves to:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Hardtoguess'
```

Now what if an attacker enters some letters together with using a single quote to end the string literal, and then inserts another Boolean expression in the where clause instead of using a regular password. Obviously, this boolean expression is TRUE which returns all the rows in the table. For instance, if an attacker instead enters the password as:

```
Aa' OR 'A'='A
```

The SQL statement now becomes:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Aa' OR 'A'='A'
```

As you can see, this query will always return all the rows in the database, with the attacker convincing the web application that a correct username and password was passed in. The kicker is that when the recordset contains the entire set of users, the first entry in the list will typically be the Administrator of the system, so there is a good chance the attacker will be authenticated with full administrative rights to the application.

SQL INJECTION SAMPLE2

Various twists on SQL Injection can also be performed. An attacker can select data other than the rows from the table being selected from by using a UNION. Here's another example of how to pull data back from other tables that are not directly involved in the current query. The best way to exploit this issue is to find a screen that contains a dynamic list of items, such as a list of open orders or the results of a search.

The trick here for the attacker is to make the single query into two queries and UNION them. This is somewhat difficult because you must match up the number of columns and column types. However, if the server provides you the error messages, the task is relatively simple. The error returned will resemble the following message:

```
Number of columns do not match
```

or:

```
2nd column in UNION statement does not match the type of the first statement.
```

This time we will look at a sample Active Server Page that might typically be found in an application.

```
Dim sql
Sql = "SELECT * FROM PRODUCT WHERE ProductName='" & product_name & "'"
Set rs = Conn.OpenRecordset(sql)
' return the rows to the browser
```

Once again, we will assume we have access to the source code. An attacker does not really need the source code, but it does make our lives easier for demonstration purposes. Once again we are not using parameterized queries, but instead are concatenating a string to build our SQL statement.

We try entering valid input by setting the product_name to:

```
DVD Player
```

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='DVD Player'
```

An attacker would instead want to get a copy of the password hashes from your databases. Once he has these hashes, he can start cracking them. The hacker would set the product_name to:

```
test' UNION select username, password from dba_users where 'a' = 'a
```

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='test' UNION select username,  
password from dba_users where 'a'='a'
```

Instead of entering a single word, the attacker used a single quote to end the string literal. The attacker then adds a UNION command and a second statement. Notice at the end that he must still handle the fact that the code will place another single quote at the end, so we end our second SQL query with:

```
'a'='a
```

This last clause evaluates to TRUE causing all rows to be returned from the dba_users table.

PREVENTING SQL INJECTION

Preventing SQL injection from happening is simple once you understand the problem. There are really two strategies you can use to prevent the attacks:

- Validate user input
- Use parameterized queries

Validating user input involves parsing a field to restrict the valid characters that are accepted. In most cases, fields should only accept alphanumeric characters.

Also you can escape single quotes into two single quotes although this method is riskier since it is much easier to miss parsing input somewhere.

Using parameterized queries involves binding variables rather than concatenating SQL statements together as strings.

The biggest challenge will be reviewing and updating all the old CGI scripts, ASP pages, etc... in your web applications to remove all instance of this vulnerability. It is also suggested that you setup a programming guideline for web programmers that includes emphasis on using parameterized queries and not constructing SQL by concatenating strings with input values.

CONCLUSION

The truth is, there are not a lot of resources out there to keep up with Oracle security. There are a few simple tasks that can be performed to reduce your security risk to a reasonable level.

- Stay patched. To download patches go to:

<ftp://oracle-ftp.oracle.com/server/patchsets>.

- Stay aware of Oracle security hole. You can subscribe to a list to receive an email when a new security alerts is publicized by going to:

<http://www.oraclesecurity.net/resources/maillinglist.html>.

- To ask questions on Oracle security, check out:

<http://www.oraclesecurity.net/cgi-bin/ubb/ultimatebb.cgi>.

- Explore possible third-party security solutions.

By staying informed and aware of security vulnerabilities relating to your Oracle database, you should be able to keep the risks to a minimum.

ABOUT APPLICATION SECURITY, INC. (APPSECINC)

AppSecInc is the pioneer in designing, developing, and managing application security solutions for the enterprise. AppSecInc products proactively secure enterprise applications by discovering, assessing, and protecting the database against rapidly changing security threats. We give organizations the confidence to extend business with customers, partners and suppliers across networks and the Internet. Our security experts, combined with our strong support team, deliver the most up-to-date application safeguards to minimize risk and eliminate impact on business. Please contact us at 1-866-927-7732 to learn more, or visit us on the web at www.appsecinc.com.