

Unraveling the Sweater

Oracle Database Security (Part 2)

Tim Gorman – Principal, SageLogix, Inc.

Overview

Some of the most experienced database administrators in the world leave their systems open to casual hacking. Hackers aren't only lonely 13 year olds with bad skin – they could be a co-worker just trying to get his/her job done without getting tangled up in the bureaucratic red tape of change management or data security. It could also be a more malicious co-worker who likes to know things about other co-workers, customers or patients.

The purpose of Part 1 of this article (published last quarter) was to show how simple it is to break into most Oracle databases. The point is not to alarm you or alert you to how scary and dangerous I am, but rather to provide justification to take some simple steps to close the easiest security loopholes. In *Unraveling the Sweater, Part 1*, we discussed how unguarded default accounts and the failure to use even the basic password protections could make unauthorized access to the database easily obtained.

There are password-hacking scripts for Oracle databases freely available all over the Internet. Go to a search engine like <http://www.google.com> and search using keywords like “hack oracle password” to find scores of simple yet powerful password cracking tools. This will also provide you some insight into both the legitimate (a.k.a. “white hat”) as well as the illegitimate (a.k.a. hacker or “black hat”) security community.

For a more complete discussion of all levels of security in an Oracle database, I strongly recommend the *Oracle Security Handbook* (Oracle Press, 2001) by Marlene Theriault and Aaron Newman.

Unraveling the sweater – finding the thread to pull...

As mentioned in the previous article, there is an old saying amongst both burglars as well as police:

Secrets are like a pretty woman's sweater – find a loose thread, start pulling and don't stop pulling until the sweater is gone.

Well, the actual saying is a lot more profane, but hopefully you get the general idea: find a flaw (any flaw) and patiently exploit it.

Most successful burglars (and police investigators) are patient, willing to watch carefully, notice any anomalies and learn enough to take the next step to move closer to their objective. In this case, find a loose thread. Even though it may not lead anywhere, gently start pulling. You never know, it might lead to something productive!

Securing the edge of the database

Part 1 focused on the presence of *unused default accounts* and the relatively unused *strengthened passwording* features of Oracle. Taking the few simple steps recommended there will prevent the most common methods of unauthorized access to the database and its services.

Moving up and away from the database server, the TNS (variously known as SQL*Net, Net8 or Oracle Net) layer of software provides both access to the database and security. This is the *edge* of the database, the *gatekeeper* if you will. As a separate service from the database itself, TNS has its own separate *authorization* and *authentication* for configuration and control. In my own informal polling, I would estimate that only a very small number of database administrators utilize any of the safeguards surrounding TNS configuration and control, and even fewer utilize the features of the Advanced Security Option, such as:

- Database authentication from external services (i.e., LDAP, Kerberos, etc.)
- Data protection using encryption (i.e., DES, 3DES)
- Data integrity-checking and validation using check-summing (i.e., MD5)

This article will not address the Advanced Security Option and these features. These features pertain to clearly identified security concerns and are nontrivial in their deployment, maintenance, and troubleshooting.

Rather, we will simply concentrate on the more mundane TNS issues:

- Authentication as a TNS administrator
- Authorization to perform startup, shutdown, display options and change configuration of TNS services

By default, TNS is wide open; allowing anybody with the TNS control programs the ability to do whatever he or she pleases.

Attitudes

Of course, one prime reason for the lack of utilization of the basic TNS security features is that the database and the TNS layer are *protected* by a firewall, walled off from the outside world. This is an excellent argument, as you can reasonably ignore the vast majority of imaginable threats to your property by locking the door to your house. But, continuing the analogy of locking the doors to your house, do you believe for a minute that that alone is sufficient?

Suppose, as a homeowner, you own guns, jewelry or fine art – any expensive insured possessions. Depending on how valuable these items are, you have to assume that, one way or another, someone is going to get into your house, locked doors or not. They might defeat the locks and the alarm system. But, they might also be houseguests or workers admitted to perform work. In other words, they might either defeat the safeguards you have installed or they might betray your trust. Regardless of the *motive* and *method*, you should consider how to protect yourself from the *opportunity*.

Personally, I have nothing of real value in my home (except for my family, of course, which is a separate discussion). So, our safeguards on our possessions are casual. Should someone take it into

their head to steal something, there is little we can do to prevent them. When such persons gain entry, our feeling is that not much damage can be done that cannot be easily replaced or repaired. Things are things.

You and your organization may have the same attitude about your Oracle database. But please think carefully about that attitude. As a homeowner and member of an independent family unit, I personally don't have to explain to anybody why our television and computer were stolen, lost through our own lack of countermeasures. But you, your supervisor and your executive management may have to do some fast explaining to shareholders if strategic data is stolen or damaged, resulting in tangible financial loss. Moreover, if a security breach violates one of the many new laws centering on privacy, then you might face criminal as well as civil charges. This may sound over the top, but the casual old attitude of "Who would do such a thing?" may soon be as rare as a car without seatbelts.

The TNS Listener service

Unlike other database engines, Oracle separates its network services from the database service. These network services can be operated entirely independent of the database, allowing a great deal of flexibility from an operations standpoint.

To correct a popular misconception, Oracle's networking software exists mainly as a software layer in both the *client* as well as the *server*. There are two major client-side application programming interfaces (APIs): the Oracle Call Interface (OCI) and JDBC Thin.

JDBC Thin is a reduced-functionality implementation using TCP network sockets through Java. It is designed to enable direct database access from *thin* clients on the Internet, primarily browser-based Java JDBC applets connecting directly to an Oracle database. In contrast, OCI is a full-functionality mechanism used from all other programs:

- JDBC OCI (designed for access from application-server-based Java JDBC servlets and beans)
- PRO* Precompilers for C, C++, Cobol, Fortran, Pascal, PL/1
- SQLJ Precompiler for Java
- ODBC
- DBD::Oracle for Perl DBI

The TNS *Listener* process is only a small part of this software layer. It is an independent program designed merely to establish connections to the Oracle database. It monitors one or more network access points (i.e., known TCP sockets, UNIX domain sockets, etc.), validates any connection requests, spawns Oracle *server* processes, and redirects the *client* to the *server* process. Once the connection is established, the role of the TNS Listener is complete and it resumes listening for more connection requests. In this respect, the TNS Listener works exactly like other network services such as *telnetd* for Telnet sessions, *ftpd* for FTP sessions, *sshd* for SSH sessions, etc.

TNS Listener Control

While most of the TNS software is built-in to Oracle client- or server-processes as a software layer, the TNS Listener operates as an independent process. It is controlled by the *lsnrctl* program and can also be configured by default by the same program.

While configuration information is stored within the running TNS Listener process, persistent configuration information is stored in a file named *listener.ora*, which resides in a file-system on the database server. When the TNS Listener program is started using the *lsnrctl start* command, configuration information is read from the *listener.ora* file:

```
$ lsnrctl start

LSNRCTL for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003 19:30:43
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Starting /opt/app/oracle/product/8.1.7/bin/tnslsnr: please wait...

TNSLSNR for Solaris: Version 8.1.7.4.0 - Production
System parameter file is /opt/app/oracle/admin/sqlnet/listener.ora
Log messages written to /opt/app/oracle/admin/sqlnet/listener.log
Listening on:
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=prd.acme.com)(PORT=1521)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=PROD)))

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=prd.acme.com)(PORT=1521)))
STATUS of the LISTENER
-----
Alias                LISTENER
Version              TNSLSNR for Solaris: Version 8.1.7.4.0 - Production
Start Date           16-MAR-2003 19:30:43
Uptime                0 days 0 hr. 0 min. 0 sec
Trace Level           off
Security              OFF
SNMP                  OFF
Listener Parameter File /opt/app/oracle/admin/sqlnet/listener.ora
Listener Log File    /opt/app/oracle/admin/sqlnet/listener.log
Services Summary...
  PLSExtProc          has 1 service handler(s)
  PRD1                 has 1 service handler(s)
The command completed successfully
```

If the configuration file is modified while the TNS Listener process is running, then the new configuration can be loaded into the running process using the *lsnrctl reload* command.

```
$ lsnrctl reload

LSNRCTL for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003 19:36:18
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=prd.acme.com)(PORT=1521)))
The command completed successfully
```

The status of a running TNS Listener process can be obtained by running the *lsnrctl status* command, while much of the details of the configuration of the process can be obtained using the *lsnrctl services* command.

```
$ lsnrctl status

LSNRCTL for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003 19:37:49
```

(c) Copyright 1998 Oracle Corporation. All rights reserved.

```
Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=prd.acme.com)(PORT=1521)))
STATUS of the LISTENER
-----
Alias                LISTENER
Version              TNSLSNR for Solaris: Version 8.1.7.4.0 - Production
Start Date           16-MAR-2003 19:30:43
Uptime                0 days 0 hr. 7 min. 6 sec
Trace Level          off
Security              OFF
SNMP                  OFF
Listener Parameter File /opt/app/oracle/admin/sqlnet/listener.ora
Listener Log File    /opt/app/oracle/admin/sqlnet/listener.log
Services Summary...
  PLSExtProc          has 1 service handler(s)
  PRD1                 has 1 service handler(s)
The command completed successfully
```

Of course, the TNS Listener process can be terminated using the *lsnrctl stop* command.

```
$ lsnrctl stop

LSNRCTL for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003 19:40:03

(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=prd.acme.com)(PORT=1521)))
The command completed successfully
```

Besides the basic control functions of *start*, *reload*, *status* and *stop*, configuration operations can also be performed from the *lsnrctl* program:

- SERVICES
- SAVE_CONFIG
- TRACE
- SPAWN
- CHANGE_PASSWORD
- SET and SHOW
 - PASSWORD (not available for SHOW command)
 - RAWMODE
 - DISPLAYMODE
 - TRC_FILE, TRC_DIRECTORY, and TRC_LEVEL
 - LOG_FILE, LOG_DIRECTORY, and LOG_STATUS
 - CONNECT_TIMEOUT
 - STARTUP_WAITTIME
 - USE_PLUGANDPLAY
 - SAVE_CONFIG_ON_STOP

There is another program named *tnsping* used for testing the functionality of a TNS Listener process:

```
$ tnsping tst1
```

```
TNS Ping Utility for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003  
19:49:50
```

```
(c) Copyright 1997 Oracle Corporation. All rights reserved.
```

```
Attempting to contact (ADDRESS=(PROTOCOL=TCP)(HOST=tst.acme.com)(PORT=1521))  
OK (10 msec)
```

If provided with a TNS name-string, then *tnsping* will resolve the ADDRESS portion of the specification to locate a responding TNS Listener process. However, a full TNS name-string is strictly necessary; the verbose TNS specification itself can also be passed to *tnsping*:

```
$ tnsping "(ADDRESS=(PROTOCOL=TCP)(HOST=tst.acme.com)(PORT=1521))"
```

```
TNS Ping Utility for Solaris: Version 8.1.7.4.0 - Production on 16-MAR-2003  
19:49:50
```

```
(c) Copyright 1997 Oracle Corporation. All rights reserved.
```

```
Attempting to contact (ADDRESS=(PROTOCOL=TCP)(HOST=tst.acme.com)(PORT=1521))  
OK (10 msec)
```

So, *tnsping* is a versatile tool and can be used on the fly.

Between them, *tnsping* can be used to locate Oracle database services on a server and *lsnrctl* can be used to uncover information about those services. There is really no way to disable these commands, other than to block them with a firewall.

Password on the TNS Listener service

Of the commands performed by the *lsnrctl* program, only the START and STATUS commands cannot be configured to require a password. Presumably, this might be because these two commands are considered relatively benign, but personally I would disagree. For example, the *lsnrctl status* command provides more information about my Oracle configuration than I would be willing to give away. And starting an invalid TNS Listener service can potentially be as harmful as stopping a valid one.

But all other *lsnrctl* passwords can be configured to require a password.

Sounds straightforward? Simply password the listener and all security problems disappear?

Yes. But...

The password for the listener is stored in the *listener.ora* configuration file. This password can be stored either in clear text or in a hash format. Admittedly, even the hash format can be broken easily, so it becomes important to ensure that only the owner account (i.e., "oracle") can read this file. Even then, this does not prevent someone from seeing the clear text password or the hash value from being noted visibly when it is displayed on a screen.

How the password value is stored is dependent on how it is set. There are two ways to set the TNS Listener password:

- Editing the *listener.ora* file using a text editor
- Using the CHANGE_PASSWORD command in the *lsnrctl* program

The former method requires storage of the password in clear text. The latter method stores a hash password value. The two methods are mutually exclusive.

One thing to remember about using the CHANGE_PASSWORD command: it must be followed by a SAVE_CONFIG command to explicitly save changes to the configuration file. If you tend to forget to execute SAVE_CONFIG, then you can set the SET SAVE_CONFIG_ON_STOP command to automatically save any configuration changes when the *lsnrctl* program is exited.

However you set the password or store the password, you will henceforth need to use the SET PASSWORD command at the command prompt inside the *lsnrctl* program to run any commands.

The ADMIN_RESTRICTIONS_<listener-name> parameter

As previously mentioned, the *lsnrctl* program is used for both control and configuration of the TNS Listener process. However, most database administrations use *lsnrctl* for control purposes – startup, shutdown and status check. Relatively few database administrators use *lsnrctl* for configuration purposes because editing the *listener.ora* file and then stopping/restarting or reloading the TNS Listener process is simpler and easier.

This is a good practice, but nevertheless, the *lsnrctl* program is capable of changing the listener configuration. This lesser-known capability, intended as a convenience, thus represents a potential risk.

There is a way to disable The ADMIN_RESTRICTIONS_*listener-name* parameter. When it is set to the value of ON or TRUE in the *listener.ora* configuration file, the ability of the *lsnrctl* program to change TNS Listener configuration settings will be disabled. This leaves *lsnrctl* with only its control capabilities, to start up, shut down and check the status of the TNS Listener service. By default, this parameter is set to OFF or FALSE.

The default name of a TNS Listener is “LISTENER,” so the parameter will commonly be named ADMIN_RESTRICTIONS_LISTENER, but be aware that the last phrase in the parameter’s name is the name of the actual listener itself. If you had configured a TNS Listener named “PROD,” then the parameter would be named ADMIN_RESTRICTIONS_PROD.

By forcing all configuration specifications to be made by editing the *listener.ora* configuration file on the server, we are relying on the security of the server itself. Someone wishing to change the TNS Listener configuration must first log into an appropriate administrator account on the server (i.e., the “oracle” account) instead of issuing commands over the network from a remote *lsnrctl* program.

Additionally, using the *lsnrctl* program over the network is dangerous. The *lsnrctl* program is quite primitive and does not encrypt any of its commands, not even its password. It has well-documented features that enable hackers to acquire significant information about the topology of your systems, using

a technique called *buffer overflow*. The only protection in this situation is to limit the number of times this password is sent over the network or avoid it altogether. Using the `ADMIN_RESTRICTIONS_listener-name` parameter to disable the ability to issue configuration commands over the network takes away many reasons for doing so. In contrast, a remote connection to the database server to edit the `listener.ora` file that uses SSH is much safer altogether.

The `tnsprobe.sh` script

I've posted a UNIX korn-shell script named `tnsprobe.sh` online at <http://www.EvDBT.com/tools.htm> that uses the `tnsping` and `lsnrctl` commands to find the following information about Oracle database services on a specific host server:

- If a TNS Listener process is present and active
- If the TNS Listener process has a password to execute privileged control and configuration operations
- What database instances the TNS Listener is establishing connections for

Provided with this information, a curious (or malicious) person could delve deeper, attempting to guess passwords, for instance. So it should be obvious that a script like `tnsprobe.sh` can be used as the starting point for an assault on database security.

The script expects either the IP address or IP symbolic hostname of a database server and works as follows (in conjunction with the `oraprobe.sh` script discussed in last quarter previous article):

1. Use the UNIX `ping` command to verify the validity of the specified IP address or hostname, and also to confirm that the server is up, available and responsive.
2. Starting from port number 1025 and continuing all the way to port 65535:
 - a. Generate a TNS address specification using `PROTOCOL=TCP`, the `HOST` specified and the `PORT`;
 - b. Pass the generated TNS address as a parameter to the `tnsping` command;
 - c. If `tnsping` returns success, then save the generated TNS address specification to a private `listener.ora` file, identified by the name "T_<host>_<port>":
 - i. Run the command `lsnrctl services T_<host>_<port>`.
 - ii. If the `lsnrctl services` command returns success, then parse the output to identify database instances:
 1. For each identified database instance, create TNS "CONNECT_DATA" specifications to append to the known TNS "ADDRESS" specification. Save these full TNS connect-string specifications to a private `tnsnames.ora` file.
 2. Call the `oraprobe.sh` script for each of the generated TNS connect-strings to try to find unguarded passwords in the database.

So, by pinging every available port on the specified database server, the `tnsping` program will find any active TNS Listener processes. By running the `lsnrctl services`, we can determine whether the control functions (i.e., `services`, `stop`, `reload`) or configuration functions (i.e., `set`, `show`, etc.) can be executed by anybody at all. If `lsnrctl services` fails, then a password has been configured previously and is needed for the `SERVICES` command to succeed. If `lsnrctl services` succeeds, then there is no password in place

and everything is wide open. What's more, if *lsnrctl services* succeeds, then we can generate fully specified TNS connect-strings to pass to the *oraprobe.sh* script or any other password hacking utility.

In a single jump, using the *tnsprobe.sh* script, which calls the *oraprobe.sh* script, one goes from only knowing the name of a database server all way to knowing some passwords in one or more databases on that server.

Executing the script looks like this:

```
$ tnsprobe.sh dbserver1
```

It will produce output like the following:

```
$ tnsprobe.sh prd.acme.com
Setting TNS_ADMIN=test_tnsprobe_9955 for duration of run...
starting at port 1025...
...port 1100...
...port 1200...
...port 1300...
...port 1400...
...port 1500...
Oracle TNS Listener detected on "prd.acme.com" at port 1521
TNS Listener on "prd.acme.com" on port 1521 not passworded.
connect dbsnmp/dbsnmp@T_1521_PRD1 is valid
connect load/load@T_1521_PRD1 is valid
connect monitor/monitor@T_1521_PRD1 is valid
connect perfstat/perfstat@T_1521_PRD1 is valid
connect scott/tiger@T_1521_PRD1 is valid
connect pthompson/pthompson@T_1521_PRD1 is valid
...port 1600...
...port 1700...

...and so on until...

...port 65500...
Completed.
```

The script created a directory named *test_tnsprobe_9955* in which to place private *listener.ora* and *tnsnames.ora* configuration files generated from information found during execution. During this execution, the server PRD.ACME.COM was probed. One TNS Listener process was found at port 1521. Since the command *lsnrctl services* succeeded, we know that the listener does not have a password. Using information returned by the *lsnrctl services* command, the script was able to determine that a database instance named PRD1 resided on that server. It then invoked the *oraprobe.sh* script to attempt to hack into the PRD1 database, which it was able to do for six accounts. Please note that only three of those accounts were “default” accounts, which were unfortunately permitted to retain their default passwords. The *oraprobe.sh* script found the other three accounts by querying the *ALL_USERS* view and then attempting to crack all the other accounts existing in the database.

Even if the *oraprobe.sh* script is not able to actually find unguarded database accounts and guess their password, we still know that we can start up or shut down the TNS Listener service at will. That itself is a risk, which is easily corrected.

Summary

Neither this paper nor the *tnsprobe.sh* script is intended to encourage malicious behavior. It is intended to point out some extremely simple security loopholes that can exist in all versions of Oracle.

Very simple measures are available to safeguard your database from even the most casual probes, such as described here.

Inside your database, please use the safeguards described in Part 1 of this article:

- Lock any unused or seldom-used accounts
- Enable the restriction on failed login attempts
- Enable the restrictions on password complexity

At the edge of your database, please limit the ability of savvy hackers to learn more about your services:

- Password your TNS Listener
- Disable TNS Listener configuration to edits on the *listener.ora* configuration file
 - ...and prevent non-authorized users from viewing or editing that file...

These few simple steps should greatly improve the stability and security of your systems.

Biography

Tim Gorman works for SageLogix, a consulting and managed-services company focusing on Oracle database servers and related I/O and backup subsystems (<http://www.SageLogix.com>). He is a database administrator with a focus on Oracle internals, performance tuning, security, troubleshooting, and high availability. He has been a DBA since a dark and stormy night in 1993, an Oracle developer since 1990, and a "C" programmer since 1983. He is co-author, with Gary Dodge, of *Essential Oracle8i Data Warehousing* (Sept 2000) and *Oracle8 Data Warehousing* (March 1998), both published by John Wiley and Sons. He also has a personal website with papers and tools at <http://www.EvDBT.com>.