Mining Information from the Listener Log - Part 1

by Arup Nanda

Part 1 | Part 2 | Part 3

An Oracle database infrastructure has several components — the database, the listener, and the clustering components (CRS, CSS, ONS, and so on, if it's a RAC database). All these components produce extensive logs to let you know what's happening behind the scenes. These logs show information vital to understanding the working of the database. Perhaps the most popular and most commonly used is the database alert log, which offers a running commentary on the operation of the database. You may find many utilities and tools, including the Grid Control and Database Console interfaces from Oracle itself, to parse the alert log and reve valuable information.

However, a very useful source of information is often overlooked — the listener log. The listener log shows some information that is not available anywhere else (for example, the service names used by the clients). Some of the information can also be obtained by other means, such as via the IP address of the clients recorded in audit trails. But even in such cases, the listener log provides a non-intrusive source for which you don't have to place instrumentation inside the database, as you must do when turning on auditing. In additio listener logs also record the listener operations, both successful and unsuccessful, which can show attacks against the listener. Since listener is usually the target of many database attacks, this information can reveal valuable clues and help you build better defenses. In summary, listener logs provide far too much valuable information to be ignored.

In this article, you will learn how to build an infrastructure to process the listener log and to use it to unearth information on your database operations. The information can be used in different scenarios, as you will see later.

Standard Disclaimer

- This article is based on my independent research into the composition of the listener log, and no from extracts from any other source, including Oracle manuals. While I have made every attemp to derive and present accurate information, there is no guarantee that the format of listener log will not be changed in some Oracle version, or will not have been changed already for some platforms. Therefore, I'm not making any kind of statement guaranteeing the accuracy of the findings on this paper.
- The results and output are from an actual production database infrastructure, but the identifying information such as IP Address, Host Name, usernames, and so on, have been masked to hide their identities. If it resembles any actual infrastructure, it is purely coincidental.

Building the Interface

The listener log file is a simple text file, so searching for specific information inside is easy; however, in its ra form, it's difficult to extract *collated* information. To make it easier, you must first build an interface to searc and extract information in any manner you want. The best way to do that is a widely used, and powerful, interface, the humble SQL*Plus and the powerful language SQL. But how could you use the SQL language to extract and summarize the information from the listener log? Simple — by using an external table; by presenting the listener log as an external table, you can easily create powerful SQL statements to extract usef pieces of information.

Full listener log

Let’s get down to business. First, let’s create a table from the listener log, with one record per line in the log. Before you do that, you must create a directory object on the location of the listener log:

```
create directory LISTENER_LOG_DIR
as '/u01/app/oracle/10.1/db1/network/log'
/
```

Of course, you should change the location of the listener log file to whatever your system uses. If you are not sure, you can find find this information by using the listener control utility:

```
prolin01.oraprol:/u01/app/oracle/dbaland/tools # lsnrctl status
listener_prolin01

LSNRCTL for HPUX: Version 10.1.0.4.0 - Production on 11-NOV-2005 20:24:37

Copyright (c) 1991, 2004, Oracle.  All rights reserved.

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=proprd1.proligence.com)(PORT=1521)(IP
=FIRST)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER_PROLIN01
Version                   TNSLSNR for HPUX: Version 10.1.0.4.0 - Production
Start Date                10-NOV-2005 22:34:10
Uptime                    0 days 21 hr. 50 min. 26 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/10.1/db1/network/admin/listener.ora
Listener Log File
/u01/app/oracle/10.1/db1/network/log/listener_prolin01.log
```

Note the line that shows “Listener Log File,” which shows the directory of the listener log file. Now, you are ready to build the external table on the listener log, using the following script:

```
create table full_listener_log
(
  line varchar2(4000)
)
organization external (
  type oracle_loader
  default directory LISTENER_LOG_DIR
  access parameters (
     records delimited by newline
     nobadfile
     nologfile
     nodiscardfile
  )
  location ('listener_prolin01.log')
)
reject limit unlimited
/
```

Replace the log file name mentioned in the location parameter with whatever is the name of the real log file. Once this is done, you can select data from this external table using simple SQL. The following is a partial output from the file without any filtering:

```
SQL> select * from full_listener_log;
LINE
--------------------------------------------------------------------------------
Trace level is currently 0
Started with pid=18597
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.20.188.29)(PORT=1521)
))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.20.188.28)(PORT=1521)
))
TIMESTAMP * CONNECT DATA [* PROTOCOL INFO] * EVENT [* SID] * RETURN CODE
16-OCT-2005 04:58:06 * (CONNECT_DATA=(SID=proprd1)(CID=(PROGRAM=)(HOST=__jdbc__)
(USER=))) * (ADDRESS=(PROTOCOL=tcp)(HOST=10.20.191.93)(PORT=43829)) * establish
* proprd1 * 12505
TNS-12505: TNS:listener does not currently know of SID given in connect descript
or
16-OCT-2005 04:58:06 * (CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin01)(USER=oraprol
))(COMMAND=status)(ARGUMENTS=64)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760))
* status * 0
```

You can now use this to find specific pieces of information. For instance, you can find out the log files the listener produces by searching the lines that start with a pattern:

```
SQL> select * from full_listener_log
  2  where line like 'Log messages written%'
  3  /
```

The output (partial) is:

```
LINE
--------------------------------------------------------------------------------
Log messages written to /u01/app/oracle/10.1/db1/network/log/listener_prolin01.l
og

Log messages written to /u01/app/oracle/10.1/db1/network/log/listener_prolin01.l
og

Log messages written to /u01/app/oracle/10.1/db1/network/log/listener_prolin01.l
og

Log messages written to /u01/app/oracle/10.1/db1/network/log/listener_prolin01.l
og
```

You can, of course, search for any string you want using this procedure. Typical search strings may be the IP addresses the listener uses to listen on, the errors it encounter, and so on.

Detailed table

This external table shows the entire listener log, one record per line in the file. This is useful when you want t see the records without any kind of formatting. However, remember that the original intent was to extract *useful* information from the log; given this, the raw format is not very useful. You need to extract information from these logs in a decipherable format.

So, let's build another table that shows the contents of the log file by breaking the lines into individual fields. To do this this, you must first understand the structure of the listener log. Most of the lines in the listener log follow a format with the fields shown as follows:

- The date and timestamp of the log entry
- The connect string used by the client

- The protocol related information (TCP/IP, port, etc.) used by the client
- The action by the client, e.g. status, establish connection, etc.
- The service_name used in the client's connect string
- The return code of the action. If the return code is 0, then the action was successful; otherwise the error code is shown.

The fields are separated from each other by "*" character. Note: Not all the log entries follow this format, although most of them do. Here are some examples of the log entries, when the listener comes up:

```
TNSLSNR for HPUX: Version 10.1.0.2.0 - Production on 14-SEP-2005 20:20:51

Copyright (c) 1991, 2004, Oracle.  All rights reserved.

Log messages written to /proprd/oracle/products/10.1/db1/network/log/listener_o
dssdb01.log
Trace information written to /proprd/oracle/products/10.1/db1/network/trace/lis
tener_prolin01.trc
Trace level is currently 0

Started with pid=3134
```

These messages do not follow a pattern of many fields; all information is contained in just one line. For the summarized information, you may ignore them, since they don't contain data of interest. If needed, these can be extracted from the table FULL_LISTENER_LOG.

Some other messages follow a pattern, but some fields are missing. For example, when a database registers itself with the listener and updates its load information, the following line appears:

```
16-SEP-2005 01:22:58 * service_update * PROPRD1 * 0
```

In this example, there are only four fields — Timestamp, Action, Service Name, and Return Code. Even thoug there are only four fields, not six, you can still parse them, but be aware of the contents of each field.

Another instance of a message being generated that is missing some fields occurs when the user issues a STATUS command from the LSNRCTL prompt. A line as shown below appears in the log:

```
16-SEP-2005 01:27:26 * (CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin01)(USER=oraods
s))(COMMAND=status)(ARGUMENTS=64)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760)
) * status * 0
```

This line also has only four fields — Timestamp, The Connect Data used by the client, the action, and the return code. However, the interpretation of the fields is different in this case; as long as you understand what information each field contains, you should be able to "slice and dice" the listener log any way you want. Mos lines will follow the six-field approach, and those lines are the ones that contain the information you are seeking. Therefore, building a standard table is not that difficult.

The following shows how to build the table:

```
create table listener_log
(
   log_date date,
   connect_string varchar2(300),
   protocol_info varchar2(300),
   action varchar2(15),
   service_name varchar2(15),
   return_code number(10)
)
organization external (
   type oracle_loader
   default directory LISTENER_LOG_DIR
   access parameters
   (
      records delimited by newline
      nobadfile
      nologfile
      nodiscardfile
      fields terminated by "*" lrtrim
      missing field values are null
      (
          log_date char(30) date_format
          date mask "DD-MON-YYYY HH24:MI:SS",
          connect_string,
          protocol_info,
          action,
          service_name,
          return_code
      )
   )
   location ('listener_prolin01.log')
)
reject limit unlimited
/
```

Needless to say, you must change the name of the listener log file in the location parameter. When created, th structure of the table looks like this:

```
LOG_DATE           DATE
CONNECT_STRING     VARCHAR2(300)
PROTOCOL_INFO      VARCHAR2(300)
ACTION             VARCHAR2(15)
SERVICE_NAME       VARCHAR2(15)
RETURN_CODE        NUMBER(10)
```

Voila! The table is ready to be mined for the valuable data it contains. Before you start, let's exsplore how to interpret the fields.

Description of Fields

Since you have now divided the lines to different parts, you can see what information they contain. The fields show information in nested groups enclosed by parentheses. The following is a partial extract from the field CONNECT_STRING:

```
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin01 .......
```

Each of these fields may have subfields nested inside within a second pair of parentheses. For instance, in the line previously noted, the field CONNECT_DATA has a nested column — CID — which, in turn, has nested fields PROGRAM, CID, and so on. Let's see how these fields are structured.

Connect String

Connect String has the following components nested inside:

1. **SID** — the Oracle SID. This is populated only if the user connects using the SID; otherwise it's NULL.
2. **CID** — a further nested field. This has the following subfields:
   a. **PROGRAM** — the name of the program issued by the client
   b. **HOST** — the host name from which it came. In some cases, the host name is now shown if the client is a JDBC clone program.
   c. **USER** — the Operating System UserID of the user that issued the command.
3. **SERVER** — Shows the type of the connection the client used - DEDICATED or SHARED
4. **SERVICE_NAME** — If the client used a service name, instead of SID, it comes here.
5. **COMMAND** — the actual command issued by the user, or issued by some other process. For instance, the user may have issued "status" to see the status, and pmon process may have issued "service_update" to update the listener with its load data. The COMMAND argument can be one of:

```
services
status
stop
service_register
service_update
service_died
```

6. **SERVICE** — this is only present when the listener control utility commands are given, e.g STATUS, SERVICES, etc. It's nested with the following elements:
   a. DESCRIPTION
      i. ADDRESS
         1. PROTOCOL
         2. HOST
         3. PORT
7. **FAILOVER_MODE** — this is present only is the client used a failover enabled connect string to connect. It's also a nested field. It has four sub-fields:
   a. TYPE — the type of the failover, e.g. BASIC
   b. METHOD — the method of failover, e.g. PRESELECT
   c. RETRIES — the number of retries the client makes
   d. DELAY — the delay between the connection attempts

Not all of the fields are relevant in a listener log; for example, when the user does not use SID to connect, but uses SERVICE_NAME, the SID field is irrelevant. When a field is not relevant, the field itself could be completely absent or it could be null; in the case of (SID=), for instance, there is no value after the "=" sign. You have to understand how to interpret information in both scenarios.

Protocol Information

The field Protocol Information has the following subfields:

PROTOCOL — the protocol that the client has used to connect, such as TCP.

HOST — the IP address of the client machine.

PORT — the port number established by the listener. (Note: It's not the port number to which the listener is listening, so this is not especially interesting to us.)

Enhancing the Interface

Now, you not only know that the fields CONNECT_STRING and PROTOCOL_INFO may have single values, but they may have strings containing sub-columns as well. It’s vital that you extract the values of these sub-columns as accurately as possible. This can be done by parsing the field to search for specific strings such as HOST and extracting the value after the “=” sign. What could be better than a function that does all f that? Th following is a function that accepts both:

- The string that needs to searched for the parameter names
- The parameter name that needs to be searched inside the string passed above

That function has been shown as follows:

```
 1  create or replace function parse_listener_log_line
 2  (
 3      p_in varchar2,
 4      p_param in varchar2
 5  )
 6  return varchar2
 7  as
 8      l_begin     number(3);
 9      l_end       number(3);
10      l_val       varchar2(2000);
11  begin
12      if p_param not in (
13              'SID',
14              'SERVICE_NAME',
15              'PROGRAM',
16              'SERVICE',
17              'HOST',
18              'USER',
19              'PROTOCOL',
20              'TYPE',
21              'METHOD',
22              'RETRIES',
23              'DELAY',
24              'PORT',
25              'COMMAND'
26      ) then
27          raise_application_error (-20001,'Invalid Parameter Value
'||p_param);
28      end if;
29      l_begin := instr (upper(p_in), '('||p_param||'=');
30      l_begin := instr (upper(p_in), '=', l_begin);
31      l_end := instr (upper(p_in), ')', l_begin);
32      l_val := substr (p_in, l_begin+1, l_end - l_begin - 1);
33      return l_val;
34* end;
```

Let’s see how this function works.

Line#	**Description**
3	The input string inside which you need to search
4	The string that needs for which you need to search
12-26	You define the valid values of strings that show up as a parameter inside the listener log; this limits mistakes that can made during the search
29-32	You search for the presence of this parameter, note the position of the parentheses and the equality sign, and determine the position of the value of the paramet

Now that this function is complete, let's see how it works. The following is a sample of the values in the colum connect_string:

```
CONNECT_STRING
--------------------------------------------------------------------------------
(CONNECT_DATA=(SID=proprd1)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin01)(USER=oraprol))(COMMAND=status)(ARG
UMENTS=64)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760))
(CONNECT_DATA=(SID=proprd1)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
(CONNECT_DATA=(SID=proprd1)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
```

To see the value of the parameter USER in the previous values, notice a few interesting things:

- The first few lines are null, probably because the lines couldn't be parsed properly, and hence, were discarded. There is no mention of any parameters there, let alone USER.
- The first non-empty line has the parameter USER, but the string is (USER=), which indicates that the value of the parameter is NULL. The same is true for some following line as well.
- The second line has the value of USER as oraprol from the string (USER=oraprol).

So, if you parse the lines for the value of the parameter UER, what should happen? To show null values, you can use the "?" character.

```
SQL> set null ?
SQL> select parse_listener_log_line(connect_string,'USER')
  2  from listener_log;
PARSE_LISTENER_LOG_LINE(CONNEC
-------------------------------------
?
?
?
?
?
?
oraprol
?
?
```

Note that the value was extracted only when it was a *valid* value. The function safely parses the input string and reports a value only if it's present. This function will be our tool to extract information from the listener log lines.

Get Down to Mining

Now that you have a basic understanding of how you can interpret the listener log, let's discuss some of the basic information you can extract, then focus on the more advanced commands.

Listener stop

To find the entries that occurred when the listener was stopped, you can search for the parameter COMMANI inside the CONNECT_STRING, which should have been "stop." This is a very important piece of information If someone killed the listener process from the UNIX prompt instead of issuing a "lsnrctl stop" command, the the line showing (COMMAND=stop) will not appear in the log; this tool can help determine how many times, and when, the listener was gracefully stopped.

```
col host format a20
col l_user format a20
col service format a15
col logdate format a20
select to_char(log_date,'mm/dd/yy hh24:mi:ss') logdate,
parse_listener_log_line(connect_string,'HOST') host,
       parse_listener_log_line(connect_string,'USER') l_user,
       parse_listener_log_line(connect_string,'SERVICE') service
from listener_log
where parse_listener_log_line(connect_string, 'COMMAND') = 'stop';
```

Here is the output:

```
LOGDATE              HOST             L_USER            SERVICE
-------------------- ---------------- ----------------- ---------------
10/16/05 05:35:41    prolin01         oraprol           LISTENER_PROLIN01
10/27/05 21:04:50    prolin01         oraprol           LISTENER_PROLIN01
```

There it is. From the output, you know that on 10/16/05 5:35:41 AM, the user “oraprol” issued a stop command to the listener named “LISTENER_PROLIN01” on the host prolin01.

Program usage

The code noted previously is useful, but you are looking for more interesting information, something you can use to make your database operation better. One question that comes up frequently is, what kind of tools are people using to access the database. How can you find out?

The answer lies in the parameter PROGRAM in the listener log, under the field CONNECT_STRING. Here is query that’s used to show what programs users are using:

```
col program format a70
col cmt format 999,999
select parse_listener_log_line(connect_string,'PROGRAM') program,
       count(1) cnt
from listener_log
group by parse_listener_log_line(connect_string,'PROGRAM');
```

The output is shown as follows:

```
C:\InstalledPrograms\Quest Software\TOAD\TOAD.exe                    1
C:\Oracle\product\10.1.0\Client_1\jdk\jre\bin\java.exe              15
C:\Program Files\Actuate7\Server\operation\fctsrvr7.exe         25,796
C:\Program Files\Embarcadero\DBA700\DBArt700.exe                    53
C:\Program Files\Informatica PowerCenter 7.1\Client\pmdesign.exe     1
C:\Program Files\Microsoft Office\OFFICE11\EXCEL.EXE                20
C:\Program Files\Microsoft Office\Office10\MSACCESS.EXE              4
C:\Program Files\Oracle\jre\1.1.8\bin\jrew.exe                       9
C:\Program Files\PLSQL Developer\plsqldev.exe                       19
C:\Program Files\Quest Software\Quest Central\QuestCentral.exe       2
C:\Program Files\Quest Software\TOAD\TOAD.exe                      846
C:\Program Files\Quest Software\Toad for Oracle\TOAD.exe            32
C:\Programs\TOAD\TOAD.exe                                           13
C:\opt\TOAD\TOAD.exe                                                 5
C:\opt\toad\toad.exe                                                 1
C:\oracle\ora10g\BIN\sqlplusw.exe                                   11
```

```
C:\oracle\ora81\bin\sqlplus.exe                                      2
C:\oracle\ora92\bin\sqlplusw.exe                                    44
C:\oracle\product\10.1.0\Client_1\BIN\sqlplusw.exe                  26
C:\oracle\product\10.2.0\db\bin\sqlplus.exe                         77
F:\Programs\Quest Software\TOAD\TOAD.exe                            16
c:\9I_CLIENT\bin\sqlplus.exe                                         5
exp@odsddb01                                                         2
odbcad32.exe                                                         1
oracle                                                              31
oracle@stcdwhdd                                                      4
sqlplus                                                             20
                                                               402,752
```

This shows a pretty interesting picture; let's do some analysis. Most of the sessions (402,752 of them) include the column NULL, indicating that the users may have been using some other access mechanism. A large number of the sessions (25,796 of them) use the program "C:\Program Files\Actuate7 \Server\operation\fctsrvr7.exe," which seems to be Actuate, a reporting program. The number sounds unusually high since this is an OLTP database, but then again, this may be justified. Either way, knowing this information keeps you better informed.

Note the following lines:

```
C:\Program Files\Quest Software\TOAD\TOAD.exe                      846
C:\Program Files\Quest Software\Toad for Oracle\TOAD.exe            32
C:\Programs\TOAD\TOAD.exe                                           13
C:\opt\TOAD\TOAD.exe                                                 5
C:\opt\toad\toad.exe                                                 1
```

The previous example shows various TOAD applications being used. You can use this to track TOAD users (and any issues associated with these users), and later, you'll know how to see the host names and IP address from their point of origination:

Note the following lines:

```
C:\Program Files\Microsoft Office\OFFICE11\EXCEL.EXE                20
C:\Program Files\Microsoft Office\Office10\MSACCESS.EXE              4
```

The first two lines show that someone accessed the database using Microsoft Excel and Microsoft Access, perhaps using ODBC drivers. Do you want to allow such a practice? That's your call, but you are better prepared to answer this if you are well informed.

Finally, note the following line:

```
C:\Program Files\PLSQL Developer\plsqldev.exe                       19
```

This shows some program named "plsqldev.exe," which has made 19 connections. This is not exactly familiar software. It may be interesting to track this user down and ask about the software.

Similarly, the log may show a number of other interesting facts about the types of programs users are using to access the database.

Conclusion

In this article, I showed you how to build this powerful tool and just two examples in which I used the tool to enhance my knowledge of the database system and how it is used by others. I provided these two examples to show the power of this tool; they are, by no means, the defining boundaries. In part two of this series, I will show you how to use this tool to solve more complex and real-life issues that require deeper mining.

--

Arup Nanda has been an Oracle DBA for more than 11 years with a career path spanning all aspects Oracle database design and development — modeling, performance tuning, security, disaster recovery, real application clusters and much more. He speaks frequently at many events such as Oracle World, IOUG Live and writes regularly for publications like Oracle Magazine, DBAZine and Select Journal (the IOUG publication). Recognizing his accomplishments and contributions to the user community, Oracle honored hir with the DBA of the Year award in 2003.

Typo in Create External Table

There has to be a whitespace between log_date and char(30). Very good article though!

Regards,
martin

Replies to this comment

Fixed! (Posted by kstone at 2006-05-05 11:28 AM)

USER and HOST is not always related

When you read the description of HOST and USER it states that the HOST is from where it came and the USER is the Operating System UserID of the user that issued the command.
However I just discovered that the USER does not always come from the HOST.
Scenario:
When you connect from a CLIENT to DATABASE A on HOST 1 and then from DATABASE A goes through a db-link to DATABASE B - still on HOST 1, then the listener log for DATABASE B shows the HOST of DATABASE A and the USER from the client.
I must admit that I expected to see the USER running DATABASE 1. The logged user does not exist on the UNIX server and NIS is not configured.