Mining Information from the Listener Log - Part 3

by Arup Nanda

Part 1 | Part 2 | Part 3

In the first two parts of this article series, you have learned how to build a tool to mine information from the listener log, which contains valuable information, yet is usually ignored in database analysis. If you haven't already done so, please take a take a moment to go through part 1 and part 2 of this series. (Part 1 contains th description of this tool.) In this third, and last article of the series, you will see how to extract enhanced bits o information from the listener log and also how to tie it in with security analysis.

Standard Disclaimer

- This paper is solely based on my independent research into the composition of the listener log and not an extract from any other source, including Oracle manuals. While I have mad every attempt to derive and present accurate information, there is no guarantee of its accuracy if the format of listener log will be changed in some Oracle version, or has not bee 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; however, th 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.

JDBC Connections

Earlier, we saw that the HOST parameter in the CONNECT_STRING shows __jdbc__ when the client connects to the database using the JDBC thin driver. In this case, the real host name is shown in the HOST parameter in the PROTOCOL_INFO field. Using that knowledge, we can pull up a report on what service name is used from which client. The following query accomplishes this:

```
select
   parse_listener_log_line(connect_string,'SERVICE_NAME') SN,
   parse_listener_log_line(protocol_info,'HOST') host,
   count(1) cnt
from listener_log
where
   parse_listener_log_line(connect_string,'HOST') = '__jdbc__'
group by
   parse_listener_log_line(connect_string,'SERVICE_NAME'),
   parse_listener_log_line(protocol_info,'HOST');
```

The output is:

```
SN              HOST                                          CNT
--------------- ---------------------------------------- --------
                10.14.105.19                                   80
                10.20.191.76                                  798
                10.20.191.77                                  150
                10.20.191.78                                  160
                10.20.191.80                                  396
                10.20.191.82                                   99
```

	10.20.191.91	274
	10.20.191.93	43,839
	10.20.194.57	96
	10.20.199.60	1
	10.20.199.67	6
	10.14.104.105	6
	10.14.104.122	8
	10.14.104.203	9
	10.14.105.105	2
	10.20.214.170	15,869
OMT	10.20.191.60	84
OMT	10.20.191.80	72
SLC	10.14.104.99	81
SLC	10.20.191.209	15
SLC	10.20.214.170	260,928
omt	10.20.191.60	1
PNAT	10.14.104.105	18
PNAT	10.20.191.116	1,056
PNAT	10.20.191.117	200
PCAT	10.20.191.60	325
PCAT	10.20.191.80	572
ADHOC	10.20.191.91	325
ADHOC	10.23.35.233	3
PROLO	10.23.35.37	4
PROLO	10.20.191.78	114
PROLO	10.20.191.82	178
PROLO	10.14.105.175	35
adhoc	10.14.32.8	18
adhoc	10.14.32.34	66
adhoc	10.14.32.76	9
adhoc	10.20.170.35	236
PROMSG	10.14.32.22	16
PROMSG	10.20.191.76	1,688
PROMSG	10.20.191.80	2
PROMSG	10.20.191.88	1,263
PROMSG	10.20.191.91	306
PROSVC	10.20.191.76	120
PROSVC	10.20.191.80	117
PROSVC	10.20.191.88	142
PROSVC	10.20.191.89	166
PROSVC	10.20.191.90	150
PROSVC	10.20.191.91	244
PROSVC	10.20.191.93	1,971
proprd	10.20.194.57	128
PROSVC	10.20.191.82	1
PROSVC	10.20.191.91	5
BOOKING	10.20.210.21	649
BOOKING	10.20.210.23	1,546
PROPEDIA	10.20.195.20	8
PROCOMM	10.20.191.76	326
PROCOMM	10.20.191.80	367
PROCOMM	10.14.105.101	1,595
PROSRCH	10.23.35.6	2
PROSRCH	10.20.191.48	236
PROSRCH	10.20.191.62	61
PROSRCH	10.20.191.63	5
PROSRCH	10.20.191.86	9
PROSRCH	10.20.191.87	8
PROSRCH	10.23.35.169	3
PROSRCH	10.23.35.233	6
PROSRCH	10.14.104.251	42
PROSRCH	10.20.191.125	7,166
proprd1	10.20.191.91	1
proprd1	10.20.194.57	8
PROMEET	10.14.32.67	3
PROMEET	10.14.104.58	116
PROMEET	10.20.218.193	424
PROMEET	10.20.218.194	433
RESPONSE	10.20.191.78	76
RESPONSE	10.20.191.82	105
response	10.20.191.78	475
PROPRD_ADHOC	10.14.105.175	1

Now it's time to match up client IP addresses to make sure they are correctly pointing to the right service

name. Some client IP addresses use multiple service names since they run multiple applications. While there no way to differentiate between applications, we can at least eliminate the possibility that an appserver uses a wrong service name.

SID or Service Name Breakup

Now that you have seen how clients use both service names and SIDs while connecting, a consolidated report will help you to understand how clients connect, and will help us fix two potential issues — clients using SIDs instead of service names, and using wrong service names. We will use this information for JDBC thin clients only, because that's where most of our applications are.

You can use this query to see the breakup of the SID/Service Names:

```
select
   parse_listener_log_line(connect_string,'SID') SID,
   parse_listener_log_line(connect_string,'SERVICE_NAME') SN,
   parse_listener_log_line(protocol_info,'HOST') host,
   count(1) cnt
from listener_log
where
   parse_listener_log_line(connect_string,'HOST') = '__jdbc__'
group by
   parse_listener_log_line(connect_string,'SID'),
   parse_listener_log_line(connect_string,'SERVICE_NAME'),
   parse_listener_log_line(protocol_info,'HOST')
```

The output is:

```
SID        SN              HOST                     CNT
---------- --------------- -------------------- --------
           OMT             10.20.191.60               84
           OMT             10.20.191.80               72
           SLC             10.14.104.99               81
           SLC             10.20.191.209              15
           SLC             10.20.214.170         261,029
           omt             10.20.191.60                1
           PNAT            10.14.104.105              18
           PNAT            10.20.191.116           1,056
           PNAT            10.20.191.117             200
           PCAT            10.20.191.60              325
           PCAT            10.20.191.80              572
           ADHOC           10.20.191.91              325
           ADHOC           10.23.35.233                3
           PROLO           10.23.35.37                 4
           PROLO           10.20.191.78              114
           PROLO           10.20.191.82              178
           PROLO           10.14.105.175              35
           adhoc           10.14.32.8                 18
           adhoc           10.14.32.34                66
           adhoc           10.14.32.76                 9
           adhoc           10.20.170.35              236
           PROMSG          10.14.32.22                16
           PROMSG          10.20.191.76            1,688
           PROMSG          10.20.191.80                2
           PROMSG          10.20.191.88            1,263
           PROMSG          10.20.191.91              306
           PROSVC          10.20.191.76              120
           PROSVC          10.20.191.80              117
           PROSVC          10.20.191.88              142
           PROSVC          10.20.191.89              166
           PROSVC          10.20.191.90              150
           PROSVC          10.20.191.91              244
           PROSVC          10.20.191.93            1,972
```

```
           proprd          10.20.194.57              128
           PROSVC          10.20.191.82                1
           PROSVC          10.20.191.91                5
           BOOKING         10.20.210.21              649
           BOOKING         10.20.210.23            1,546
           PROPEDIA        10.20.195.20                8
           PROCOMM         10.20.191.76              326
           PROCOMM         10.20.191.80              367
           PROCOMM         10.14.105.101           1,595
           PROSRCH         10.23.35.6                  2
           PROSRCH         10.20.191.48              236
           PROSRCH         10.20.191.62               61
           PROSRCH         10.20.191.63                5
           PROSRCH         10.20.191.86                9
           PROSRCH         10.20.191.87                8
           PROSRCH         10.23.35.169                3
           PROSRCH         10.23.35.233                6
           PROSRCH         10.14.104.251              42
           PROSRCH         10.20.191.125           7,166
           proprd1         10.20.191.91                1
           proprd1         10.20.194.57                8
           PROMEET         10.14.32.67                 3
           PROMEET         10.14.104.58              116
           PROMEET         10.20.218.193             424
           PROMEET         10.20.218.194             433
           RESPONSE        10.20.191.78               76
           RESPONSE        10.20.191.82              105
           response        10.20.191.78              475
           PROPRD_ADHOC    10.14.105.175               1
PROPRD                     10.14.105.105               1
PROPRD1                    10.20.191.77              150
PROPRD1                    10.20.199.60                1
PROPRD1                    10.20.199.67                6
PROPRD1                    10.14.104.105               6
PROPRD1                    10.20.214.170          15,869
PROSRCH                    10.14.105.105               1
proprd1                    10.14.105.19               80
proprd1                    10.20.191.76              798
proprd1                    10.20.191.78              160
proprd1                    10.20.191.80              396
proprd1                    10.20.191.82               99
proprd1                    10.20.191.91              274
proprd1                    10.20.191.93           43,839
proprd1                    10.20.194.57               96
proprd1                    10.14.104.122               8
proprd1                    10.14.104.203               9
```

If the SID column is NULL, then the client has used the Service Name, which is displayed in the next column. This output shows relatively good news. Most of the clients are using Service Name instead of SIDs. This output serves two purposes — (1) you can target the IP addresses shown in the lower portion of the output to change them to service names whenever possible, and (2) you can check the Service Names used by the IP addresses in the upper half if they are accurate.

Mining for Security

So far, we have collected information on the database connections that are legitimate. Listener logs also contain information on unsuccessful attempts for connection. Even though not all unsuccessful attempts are attacks, a pattern might emerge from the attempt to show a potential attack. Using the listener mining tool, w can reveal a lot of those issues.

Listener Password

When you set a password for the listener, the user must supply the correct password before issuing some damaging commands such as stopping the listener. **Note**: this behavior is different across Oracle versions. In

Oracle 9i and earlier, a password, if set, applies to any user trying to manipulate the listener. In Oracle 10*g* an later, the Oracle software owner without a password can manipulate the listener. So, if a user other than the software owner tries to manipulate the listener, he has to supply the correct password, else he gets the following error:

```
TNS-01190: The user is not authorized to execute the requested listener command
```

And this message also finds its way to the listener log file such as the following line:

```
06-NOV-2005 13:45:06 * (CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin01)(USER=ananda
))(COMMAND=stop)(ARGUMENTS=64)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760)) *
 stop * 1190
TNS-01190: The user is not authorized to execute the requested listener command
```

We can mine this information from the listener log using our tool. Note an important difference, however. Th line has just four fields, not the usual six. Therefore, the field ACTION will show the last field on this line — tl return code, i.e., 1190.

```
col l_user format a10
col service format a20
col logdate format a20
col host format a10
col RC format a5
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,
       action RC
from listener_log
where parse_listener_log_line(connect_string, 'COMMAND') = 'stop';
```

The output is:

```
LOGDATE              HOST       L_USER     SERVICE              RC
-------------------- ---------- ---------- -------------------- -----
10/16/05 05:35:41    prolin01   oraprol    LISTENER_PROLIN01    0
10/27/05 21:04:50    prolin01   oraprol    LISTENER_PROLIN01    0
11/06/05 13:45:06    prolin01   ananda     LISTENER_PROLIN01    1190
11/06/05 13:46:00    prolin01   ananda     LISTENER_PROLIN01    0
```

Read the lines of the previous example carefully. On one occasion, on 11/06/05 13:45:06, the user “ananda” issued the stop command to the listener LISTENER_PROLIN01, without supplying the right password. Does this indicate an attack? The answer lies in the next line. About a minute later, at 13:46, the user probably realized the mistake in the password and supplied the right one and started the listener properly, as shown by the Return Code of “0.” However, if we had seen a number of lines with Return Code 1190, then we would hav suspected a possible attack. In addition, we would also have verified that the UNIX user “ananda” is actually a DBA, mapped to a physical person, and upon questioning, we’d have found out that the user was indeed tryin to stop the listener, but it failed due to a bad password first time. It all fits together.

Here is another example:

```
LOGDATE              HOST       L_USER     SERVICE              RC
```

```
------------------- ---------- ---------- -------------------- -----
10/16/05 05:35:41    prolin01   oraprol    LISTENER_PROLIN01    0
10/27/05 21:04:50    prolin01   oraprol    LISTENER_PROLIN01    0
11/06/05 13:45:06    prolin01   ananda     LISTENER_PROLIN01    1190
11/06/05 13:45:37    prolin01   ananda     LISTENER_PROLIN01    1190
11/06/05 13:46:01    prolin01   ananda     LISTENER_PROLIN01    1190
11/06/05 13:46:41    prolin01   ananda     LISTENER_PROLIN01    1190
11/06/05 13:47:05    prolin01   ananda     LISTENER_PROLIN01    1190
```

This time, we see that the user “ananda” made several attempts to supply the correct password, each within seconds of the other. This could indicate an explainable and benign situation — the user forgot the password and was trying to enter all possible commands. It also indicates a potentially malignant situation, which the user “ananda” was actually trying to break the password and stop the listener illegally. This should warrant attention and further investigation.

Log File Redirection

One of the breaches comes from the exploit available in the listener code, in which case a hacker might chang the log directory to something other than the default, and then use that to gain valuable information about th listener, the services, the database, and so on. In a more serious exploit, the hacker might direct certain commands to be placed in the trace files that creates a user and grants it a DBA role. These commands are then placed in the glogin.sql file, which is executed automatically every time someone on the server connects to the database using SQL*Plus. When the DBA logs in, the code is also executed, which creates this Trojan horse user. To prevent such an exploit, you should place a password on the listener. When the user tries to modify these values, the correct password must be specified. If the wrong password is supplied, the user gets TNS-1190 error, which also goes to the log file. Here are two sample entries in the log file, when an incorrect password was issued:

```
06-NOV-2005 13:52:33 *
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin02)(USER=ananda))(COMMAND=log_file)(AR
GUMENTS=4)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760)(VALUE=/tmp/a)) *
log_file * 1190
06-NOV-2005 14:01:45 *
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=prolin02)(USER=ananda))(COMMAND=log_director
y)(ARGUMENTS=4)(SERVICE=LISTENER_PROLIN01)(VERSION=168821760)(VALUE=/tmp)) *
log_directory * 1190
```

This is also something we can successfully mine from logs using our listener log mining tool. Note that this ha only four fields, not six, as in case of a regular listener log line. These four map to the first four fields on the listener log external table, even though the actual column names may be different. So, the meanings of some o the external table columns are different now as shown in the following:

Column	New Meaning
LOG_DATE	Log Date
CONNECT_STRING	The Connect string, meaning is same
PROTOCOL_INFO	The command given by the user (log_file), not the same meaning as protocol_info
	The return code (1190), not the

ACTION	

Using this understanding, we can write the query to find out who issued the “log_file” command and what wa the result:

```
col l_user format a10
col service format a20
col logdate format a20
col host format a10
col RC format a5
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,
       action RC
from listener_log
where parse_listener_log_line(connect_string, 'COMMAND') = 'log_file';
```

This shows the output:

```
LOGDATE              HOST       L_USER     SERVICE              RC
-------------------- ---------- ---------- -------------------- -----
11/06/05 13:52:33    prolin01   ananda     LISTENER_PROLIN01    1190
```

This shows that at the specified time, the user “ananda” tried to change the log file of the listener without supplying the correct password. He must have received the TNS-1190 error, which is what we see in the listener log. This could be an honest mistake, but is definitely worth an investigation.

Admin Restrictions

As I explained previously, one of the most common attacks against the database come through the listener by changing the log file to the glogin.sql in the directory $ORACLE_HOME/sqlplus/admin and placing some commands there. However, what if you could restrict the ability to issue the command from the LSNRCTL prompt? You can do so by placing the RESTRICT_ADMIN option in the listener.ora file and restarting the listener. Once this is place, the only way to change the log_file, log_directory, trc_level, and so on, is to chang them in the listener.ora file, then reload the listener. If you want to change them online, you will get this erro

```
TNS-12508: TNS:listener could not resolve the COMMAND given
```

The following line will appear in the listener log file:

```
06-NOV-2005 14:10:20 * trc_level * 12508
TNS-12508: TNS:listener could not resolve the COMMAND given
```

If you search for this error, you can determine whether anyone has attempted to change them online. Note th this has only three fields, not six, as in case of a regular listener log line. These three map to the first three fields on the listener log external table, even though the actual columns names are different. So, meanings of the external table columns may differ as shown:

Column	New Meaning
LOG_DATE	Log Date
CONNECT_STRING	The command given by the user (trc_level)
PROTOCOL_INFO	The return code (12508)

We can then write the query to extract the correct information:

```
Col command format a20
Col return_code format a15
Set pages 3400
select
       log_date,
       connect_string       command,
       protocol_info        return_code
from listener_log
where connect_string in
(
        'password',
        'rawmode',
        'displaymode',
        'trc_file',
        'trc_directory',
        'trc_level',
        'log_file',
        'log_directory',
        'log_status',
        'current_listener',
        'inbound_connect_timeout',
        'startup_waittime',
        'save_config_on_stop',
        'start',
        'stop',
        'status',
        'services',
        'version',
        'reload',
        'save_config',
        'trace',
        'spawn',
        'change_password',
        'quit',
        'exit'
)
/
```

This returns the following:

```
LOG_DATE  COMMAND              RETURN_CODE
--------- -------------------- ---------------
06-NOV-05 change_password      0
06-NOV-05 save_config          0
06-NOV-05 log_file             0
06-NOV-05 trc_level            12508
06-NOV-05 save_config_on_stop  12508
06-NOV-05 log_directory        12508
06-NOV-05 log_directory        12508
06-NOV-05 stop                 1169
06-NOV-05 stop                 1169
```

```
06-NOV-05 services             1169
06-NOV-05 status               1169
06-NOV-05 reload               1169
06-NOV-05 status               1169
06-NOV-05 stop                 1169
06-NOV-05 status               1169
06-NOV-05 stop                 1169
```

The results are interesting. On several occasions, we see that someone has issued the commands without restarting the listener. Some of them may be really benign — they may have been entered by mistake. But that's something you can easily verify.

To verify further, you will need to find out all other details about these commands — the OS userid, the extat time, and so on. You can create a generic query as the following:

```
select log_date,
       parse_listener_log_line(connect_string,'USER') l_user,
       protocol_info command,
       action return_code
from listener_log
where
       parse_listener_log_line(connect_string,'COMMAND')
       in
       (
               'start',
               'stop',
               'status',
               'services',
               'version',
               'reload',
               'save_config',
               'trace','spawn',
               'change_password',
               'quit',
               'exit'
       )
and action != '0'
/
```

The partial output is:

```
LOG_DATE  L_USER          COMMAND         RETURN_CODE
--------- --------------- --------------- ---------------
06-NOV-05 ananda          stop            1190
...
...
```

It appears that the user "ananda" issued a stop command, and received an error 1190, as shown in RETURN_CODE. The full error message the user must have got is:

```
TNS-01190: The user is not authorized to execute the requested listener command
```

Again, this could have been an honest mistake; but it warrants an investigation, nonetheless. If you see a pattern, this may lead to a possible attack.

Combining with Auditing

So far, we have seen how to mine these valuable pieces of information from the listener log alone. There is another source of vital connection information — the database audit trail. This is not turned on by default; yo must explicitly enable it by setting the initialization parameter audit_trail to DB_EXTENDED or DB and the restarting the database. When the database is up, issue the following statement:

```
AUDIT CREATE SESSION;
```

This will enable the auditing of all logons and logoffs, but not of any specific statements. After the database h been running for a while, you can see the audit trails in the view DBA_AUDIT_TRAIL. The audit trails are recorded with the timestamp as well; however, the timestamps are made when the database connections are made, and the listener log timestamp is made when the listener gets the request. There is a difference betwee the two — usually one or two seconds, which we will have to accommodate in the query.

You can perform the following query to combine the audit trails and listener log entries:

```
select username, ses_actions, logoff_time, comment_text, sql_text
from dba_audit_trail a, listener_log l
where parse_listener_log_line(connect_string,'SERVICE_NAME') = 'DBA'
and   parse_listener_log_line(connect_string,'HOST') = 'STSCOTIGERAT42'
and l.log_date between a.timestamp - (2/24/60/60) and a.timestamp
and a.terminal = 'STSCOTIGERAT42'
/
```

The output is as follows:

```
USERNAME                       SES_ACTIONS         LOGOFF_TI
------------------------------ ------------------- ---------
COMMENT_TEXT
--------------------------------------------------------------------------------
-
SQL_TEXT
--------------------------------------------------------------------------------
-
SCOTIGERA                                          02-NOV-05
Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=10.14.1
04.72)(PORT=1365))
```

This shows that the user connected as the Oracle userid SCOTIGERA, even though she used the service_nam as DBA, so we can rest easy.

Now it is time to connect the JDBC connections with service names and userids. The audit trails contain the username, but not the service name, while listener logs have service names, but not usernames. So, we need t combine them to get the complete picture. Remember, from the discussion on tracking client machines, that the host with IP address 10.20.214.170 had the maximum number of connections using thin JDBC drivers? Let's find out how many service names are used from that client.

First, we need to find out the hostname of the server with that IP Address. We can try to get it by nslookup, provided it's in the DNS:

```
c:\Work\Orascripts>nslookup 10.20.214.170
Name:    stcdelpas01.starwoodhotels.com
```

```
Address:  10.20.214.170
```

The name of the client is “stcdelpas01,” which is what appears in the Audit Trails. So, we can combine the Listener Log records with the Audit Trail ones to track the service names used by this client.

```
select
   parse_listener_log_line(connect_string,'SERVICE_NAME') SN,
   a.username,
   count(1) cnt
from
   listener_log l,
   dba_audit_trail a
where
   parse_listener_log_line(connect_string,'HOST') = '__jdbc__'
and
   parse_listener_log_line(protocol_info,'HOST') = '10.20.214.170'
and
   l.log_date between a.timestamp - (2/24/60/60) and a.timestamp
and
   a.userhost = 'stcdelpas01'
group by
   parse_listener_log_line(connect_string,'SERVICE_NAME'),
   a.username
/
```

From the output, we can ascertain how the usernames and service names are connected from the hostname. This is just an example of how you can combine audit logs and listener logs.

Conclusion

As you can see in the previous sections, the listener log contains several valuable pieces of information that ca offer insights into the way users connect to the database and how to diagnose common existing, and potential issues, allowing you to be proactive or at least to gain in-depth knowledge about the database usage.

These articles are an introduction to the concept of building this tool, they do not show the full capabilities of this tool or the concept. From this foundation, you can build your own enhancements, with all the sophistication you want. For instance, you can build a scheduler job that polls this table and intelligently send a server alert if a certain condition is satisfied (e.g., if someone tried to stop the listener several times with a wrong password or to change the log_file, even though the listener is under ADMIN RESTRICT condition, indicating some potential security breach). You can build forms in HTMLDB to present these in some user-friendly format, or even persuade some friendly developer to develop a Web page that shows the data in a nic formatted manner for all to examine. The possibilities are endless, and I hope I have tickled your imagination

Good luck mining and enhancing this tool. If you do add enhancements to this tool, I would highly appreciate it if you could drop me a line or post a comment to this article. After all, knowledge grows by sharing.

--

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.com, 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.