Mining Information from the Listener Log - Part 2

by Arup Nanda

Part 1 | Part 2 | Part 3

In part one of this article, I showed you how to build a tool to mine information from a valuable, yet often overlooked, source of database activity — the listener log file. You also saw two basic examples of instances in which the tool was used. In this second part of the article series, I will show you how to use the tool in a more advanced manner. (If you haven't read part 1 yet, please read it before continuing; how to build the tool has been described there.)

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.

Service Name Usage

If you have a RAC database, you may have introduced a few features to support some of the advanced functionalities of RAC, e.g., load balancing across nodes and making sessions fail over to a surviving node when a node dies. These things will occur only if users are employing the proper connect string, utilizing SERVICE_NAME and not SID. Here is one way to use connect strings, utilizing service name PROMEET connecting to the RAC database running on three nodes – prolin1, prolin2 and prolin3.

```
PROPRD =
   (DESCRIPTION =
     (LOAD_BALANCE = on)
     (FAILOVER = on)
     (enable = broken)
     (ADDRESS = (PROTOCOL = TCP)(HOST = prolin1)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = prolin2)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = prolin3)(PORT = 1521))
     (CONNECT_DATA =
       (SERVER = DEDICATED)
       (SERVICE_NAME = PROMEET)
       (FAILOVER_MODE =
         (TYPE = SELECT)
         (METHOD = BASIC)
         (RETRIES = 120)
         (DELAY = 2)
       )
     )
   )
```

An example of the connect string using SID is shown as follows:

```
PROPRD =
   (DESCRIPTION =
     (ADDRESS = (PROTOCOL = TCP)(HOST = prolin1)(PORT = 1521))
     (CONNECT_DATA =
       (SID = PROPRD1)
     )
   )
```

In this case, if the node prolin1 fails, the session will not fail over to the other surviving node. If the client attempts to make the connection again, it will fail, and users should understand this. How can you proactively capture these users, still utilizing the SID? You can't get that information from the dynamic performance view V$SESSION; the listener log records that data.

When the users employ a connect string using the SID, the listener log entry shows SID as a nested parameter in the field CONNECT_STRING. If they use SERVICE_NAME, the SID is not present or NULL. You can then parse the field for the parameter and check the value.

```
col sid format a15
select parse_listener_log_line(connect_string,'SID') sid, count(1) cnt
from listener_log
group by parse_listener_log_line(connect_string,'SID');
```

The output is:

```
SID                     CNT
--------------- ----------
PROPRD                   1
PROPRD1              16292
PROPRD2                  1
PROSRCH                  1
proprd1              64601
proprd2               3242
                     349780
```

This produces some interesting results. We see that 349,780 of them (where the SID value is NULL) have used SERVICE_NAME, but not the others. A total of 64,601 sessions used proprd1 as the SID, and 16,292 used PROPRD1 as the SID name. It will be a good idea to go after those users and inquire why they are not using the SERVICE_NAME. Regardless of the reason, you can make them aware that their sessions will not fail over in case of failure in the instance to which they are connected.

Besides RAC failover, Service Names are also used in a variety of other ways, such as to control resource usage through Resource Manager, to track SQL performance, and track database usages such as CPU and IO. If you are planning to enforce Service Name adoption in your database, this is a great way to start, and later it will be a tool to track the progress of its adoption.

There is an interesting fact in the output here. Note the line:

```
PROSRCH                  1
```

There is no SID named PROSRCH for which this listener is paying attention. So, how come this listener appears in the above output? The presence in the listener log indicates that the user has tried to connect; but may not have been successful. Let's confirm that:

```
SQL> select return_code
  2  from listener_log
  3  where parse_listener_log_line(connect_string,'SID') = 'PROSRCH'
  4  /
RETURN_CODE
-----------
      12505
```

The return code is 12505. The client must have received the error TNS-12505 while making this connection attempt. To see what the error means, you can use the “oerr” utility:

```
prolin01.oraprol:/u01/app/oracle/10.1/db1/network/log # oerr tns 12505
12505, 00000, "TNS:listener does not currently know of SID given in connect
descriptor"
// *Cause:  The listener received a request to establish a connection to a
// database or other service. The connect descriptor received by the listener
// specified a SID for an instance (usually a database instance) that either
// has not yet dynamically registered with the listener or has not been
// statically configured for the listener. This may be a temporary condition
// such as after the listener has started, but before the database instance
// has registered with the listener.
// *Action:
//  - Wait a moment and try to connect a second time.
//  - Check which instances are currently known by the listener by executing:
//    lsnrctl services <listener name>
//  - Check that the SID parameter in the connect descriptor specifies
//    an instance known by the listener.
//  - Check for an event in the listener.log file.
```

The mystery is clear now. This connection was accepted by the listener; but then the listener immediately refused the connection with a TNS-12505 error since there was no SID named PROSRCH to which it was listening.

Now let’s be a little more helpful. Why did this user connect using an invalid SID? There are many possible explanations — one of them being that, perhaps, we told them to use that SID? No, that’s not possible; we know the SID well enough to tell it correctly. A more likely reason could be that we told them PROSRCH was the service name to use, but they misunderstood and thought it was the SID. Now, the connection didn’t work and we may have a disgruntled and confused user.

Let’s put on our “good neighbor hat” and proactively seek this user out to correct the mistake. The best way to do so is by selecting all the columns of the table for that record:

```
SQL> select return_code
  2  from listener_log
  3  where parse_listener_log_line(connect_string,'SID') = 'PROSRCH'
  4  /
```

The output is shown below in a vertical format to make the values more readable.

```
LOG_DATE         : 18-oct-2005 11:30:28
CONNECT_STRING   :
(CONNECT_DATA=(SID=PROSRCH)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
PROTOCOL_INFO    : (ADDRESS=(PROTOCOL=tcp)(HOST=10.14.105.105)(PORT=3164))
ACTION           : establish
SERVICE_NAME     : PROSRCH
RETURN_CODE      : 12505
```

The output shows that the user connected from the IP address 10.14.105.105, using TCP connection. The HOST value shows __jdbc__, which typically indicates a JDBC thin client. We can find out which applicatior group owns this appserver and tell them about the error. We can feel the tinkle in their eye when we tell them "hey Joe, on October 18th, at 11:30 AM, your appserver 10.14.105.105 tried to connect to the database but failed with a TNS-12505 error; and we know exactly why". Won't we look like heroes?

Do We Have Too Many Service Names

You have diligently defined several service names inside the database, which makes it easier to track individu groups of users. However, even after defining all the services, there is no guarantee that the users have selecte these service names. How can you ensure they have been using these service names? Have you defined servic names that are not being used? Checking the SERVICE_NAME column from the dynamic view V$SESSION does not work because the row shows a session occuring at the current time only; once the user disconnects, the session disappears. But just because you don't see a service name being used does not mean that it has no been used. It may have been used before. So, how can you be sure which service names have never been used

Again, the listener log comes to rescue. To find this information, use the following query:

```
select name from v$services
minus
select distinct parse_listener_log_line(connect_string,'SERVICE_NAME')
from listener_log
/
```

The output shows the following:

```
NAME
----------------------
AFCP
DJ
SARATOGA
PVO
SYS$BACKGROUND
SYS$USERS
PROCOMM
PROMEET
```

This is pretty enlightening. Note, the service names SYS$BACKGROUND and SYS$USERS are not defined by the user; they are default service names. So it is easy to understand why they would show up. What about the others? To find the answer, we will need to check the service names used by clients from the listener log:

```
SQL> select distinct parse_listener_log_line(connect_string,'SERVICE_NAME')
  2  from listener_log
  3  /
PARSE_LISTENER_LOG_LINE(CONNEC
--------------------------------------------------------------------------------
ADHOC
PNAT
BOOKING
DBA
DWETL
PROPEDIA
PCAT
PROCOMM
PROMEET
PROPRD
```

```
PROPRD_ADHOC
OMT
PROLO
REPORT
PROMSG
RESPONSE
PROSVC
PROSRCH
SLC
adhoc
dba
dj
proprd
proprd1
omt
response
service_name
PROSVC
slc
```

Again, this shows some interesting results. The service names DJ, PROMEET, and PROCOMM have been use earlier. They showed up in the "not used" list, but note carefully – they showed up as "dj," "PROMEET," and "PROCOMM," all in lowercase, not uppercase. So there are two service names defined in the database — "PROCOMM" (uppercase) and "PROCOMM" (lowercase). The same is true for PROMEET and DJ. But the other service names — AFCP, SARATOGA and PVO — have never been used.

This is the only way to know that whether or not a service has been used. Of course, you can always check fro AWR; but if AWR has been disabled, for whatever reason, that source is gone also. In any case, the method I' just shown is the easiest way to find out.

Tracking Client Machines

While building a security perimeter around your database, the most important thing to find out is how to trac the hot machines from which the users are connect. How can you find out? Of course, one option is to turn on auditing and then track the client hosts. However, that may not be possible for a variety of reasons — auditing is an intrusive activity; the audit trails are written to system tablespace and may cause database failure if they make the tablespace fill up. To get an audit trail, the database must be restarted after setting the parameter audit_db to DB in the database initialization parameter file; however, that may not be feasible, and so on. We don't need to do all that; we can turn to our new friend the listener log for that information. To get this data, we can look into the value of the parameter HOST in the field CONNECT_STRING:

```
col host format a40
select parse_listener_log_line(connect_string,'HOST') host, count(1) cnt
from listener_log
group by parse_listener_log_line(connect_string,'HOST');
```

The output is:

```
HOST                                        CNT
---------------------------------------- ----------
BRAACTPAS02                                      1
BRAACTPAS03                                  26013
GYEET30                                         34
IBM-CGAGNE-T30                                   2
ST-ACHAN-T40                                     9
ST-LKWANG-T41                                   18
ST-NDRACEA-T40                                  30
STACHISHOLT42                                   20
STADHIMANT41                                     1
```

STAFOSTERT42	12
STANANDAT42	81
STATHOTANG	41
STATRANT41	18
STBJONEST41	1
STBQUINT41	13
STBWALSHT41	37
STCABUTTARO03	55
STCARINTOUL04	28
STCDCANNON02	15
STCDGILLIS01	6
STCGGU01	11
STCGLIGHT02	51
STCGMINYARD01	15
STCGWRINN02	1
STCHJALILI02	54
STCJHARDIN02	1
STCJSIWEK02	13
STCKBASEY01	2
STCKENLUB01	55
STCKESSEGIAN03	1
STCMSTDIS01	1
STCNGT40	5
STCNORTONT41	11
STCPARKT41	5
STCPKETELTAS03	2
STCROSSEDM01	6
STCSPRASAD02	9
STCWALKERT30	2
STCYHERWONO01	2
STCYKOSTOVA02	4
STDDELRIOT40	24
STDFANELLIT41	13
STEDEANT41	83
STHZEIGLERT41	23
STHZIEGLER40	1
STJARCHERT41	41
STJEILERST40	77
STJJENKINST41	25
STJNELSONT40	13
STJSABOTKAT40	5
STKFORNERT30	35
STMJAMEST41	12
STMLEET41	18
STNHEBBART41	3
STNRAZAKT41	20
STR-NROYCROF-T4	19
STR-ODADA-T40	6
STR-PJAIN-T41	4
STR-QCHAO-T40	3
STRKGOVINDAT30	3
STRMROYCROFTT3	3
STRSCOTIGERT3	3
STSPATELT40	16
STSZHANGT40	8
STTNGT40	11
STSCOTIGERAT42	1
STUMSTEARMAT41	25
STVCHANGT41	8
STWCHOIT41	17
SW0001022C042B	16
__jdbc__	337824
localhost.localdomain	13
odsddb01	2
prolin01	29793
prolin02	2
odsqdb02	2
sgpas02	35
stcbkgpas01.proligence.com	2
stcdocpdb01	4841
stcdwhdd	10
stcdwhpd	14831
stcdwhqd	10
stcfogpdb01.starwoodhotels.com	20
stciispas01	2329
stcshrpas06	4418

```
stcudtpdb03                                    6
                                           12632
```

This output should be a great help in understanding which client machines are connect to the database, and for how long. While going through this list, it's important to note well known appservers; most of the connections should be coming from them. If not, then you may have a rogue client banging away at the database without your knowledge. In any case, it's good to keep a history of these connections and compare them from time to time, because this will reveal when the new clients come on to connect to the database.

Note the following line:

```
__jdbc__                                  337824
```

It indicates that the host named __jdbc__ has tried to connect 337,824 times, but that doesn't sound like a host name; so what is it? It indicates a JDBC thin driver, and that these connections came from an appserver connecting via a JDBC thin driver. So, how do you track these clients?

The answer lies in the other column of our table — PROTOCOL_INFO — which also has a parameter called HOST. This parameter reflects the true IP address of the client. We can look for IP addresses from this colum instead of the CONNECT_STRING:

```
select
   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(protocol_info,'HOST')
order by
   count(1)
```

The output is:

```
HOST                          CNT
----------------------- ----------
10.20.199.60                   1
10.14.105.105                  2
10.23.35.217                   2
10.23.35.6                     2
10.14.32.67                    3
10.23.35.169                   3
10.23.35.37                    4
10.20.191.63                   5
10.14.32.97                    6
10.20.199.67                   6
10.14.105.48                   7
10.20.195.20                   8
10.14.104.203                  9
10.23.35.233                   9
10.14.32.76                    9
10.20.191.87                   9
10.20.191.86                  10
10.14.32.22                   16
10.14.32.8                    18
10.20.191.209                 20
10.14.104.105                 24
10.14.105.175                 36
10.14.104.122                 39
10.14.104.251                 43
```

```
10.14.32.34                      66
10.14.105.19                     80
10.14.104.99                     81
10.20.191.62                    115
10.14.104.58                    152
10.20.191.77                    192
10.20.191.117                   224
10.20.191.48                    265
10.20.170.35                    321
10.20.194.57                    409
10.20.191.89                    548
10.20.191.60                    562
10.20.218.194                   569
10.20.218.193                   575
10.20.191.90                    627
10.20.191.116                  1174
10.20.210.21                   1336
10.20.191.78                   1594
10.14.105.101                  1595
10.20.191.91                   1804
10.20.191.88                   1932
10.20.191.80                   2358
10.20.210.23                   2949
10.20.191.82                   3682
10.20.191.76                   4917
10.20.191.125                  7207
10.20.191.93                  48802
10.20.214.170                363442
```

Note these IP addresses and the number of connections they have made to the database. The last one — 10.20.214.170 — has made the largest number of connections. Is this your main application server? Going through the list should provide you with clues to identify a rogue appserver that is trying to establish connections.

Tracking Service Names

Earlier, we saw how service names, rather than SIDs in connect string — whether in the JDBC connect string or in TNSNAMES.ORA file — help you immensely. They allow the sessions to fail over in a RAC environment; they help the DBA track the performance based on service name; they allow the DBA to allocate resource limits, and so on. Now that you have created all the service names and instructed everyone to use them, how can you make sure that users are actually using them? Simple; just mine some more jewels from our trusted friend, the listener log.

When users connect to the database using SERVICE_NAME in the TNS connect string, the fact is recorded in the listener log, in the CONNECT_STRING field. The parameter is SERVICE_NAME. While searching for thi parameter, you can also search for the HOST parameter; so that you can track which hosts are using Service Names and which ones are not. We will use the following query:

```
col sn format a15
col host format a45
col cnt format 999,999
select
   parse_listener_log_line(connect_string,'SERVICE_NAME') SN,
   parse_listener_log_line(connect_string,'HOST') host,
   count(1) cnt
from listener_log
group by
   parse_listener_log_line(connect_string,'SERVICE_NAME'),
   parse_listener_log_line(connect_string,'HOST')
```

The output is:

SN	HOST	CNT
		13,006
	__jdbc__	61,793
	odsddb01	2
	prolin01	30,553
	odsqdb02	2
	STMLEET41	18
	STANANDAT42	53
	STCKENLUB01	55
	STNRAZAKT41	17
	stcdocpdb01	4,907
	ST-ACHAN-T40	9
	STCROSSEDM01	6
	STDDELRIOT40	18
	STKFORNERT30	35
	STCABUTTARO03	55
	STCGMINYARD01	1
	STCYHERWONO01	2
	STCYKOSTOVA02	4
	STJJENKINST41	25
	ST-NDRACEA-T40	22
	STCKESSEGIAN03	1
	STCPKETELTAS03	2
	STR-NROYCROF-T4	19
	stcfogpdb01.starwoodhotels.com	21
dj	stciispas01	2,365
dj	IBM-CGAGNE-T30	2
DBA	STATHOTANG	15
DBA	STANANDAT42	31
DBA	STCJSIWEK02	12
DBA	STNRAZAKT41	3
DBA	STSZHANGT40	8
DBA	stcudtpdb03	6
DBA	STRKGOVINDAT30	3
DBA	STSCOTIGERAT42	1
DBA	STUMSTEARMAT41	25
OMT	__jdbc__	156
OMT	stcshrpas06	3,448
SLC	__jdbc__	260,819
SLC	STVCHANGT41	9
SLC	STDDELRIOT40	6
dba	STWCHOIT41	17
omt	__jdbc__	1
slc	STCGWRINN02	1
PNAT	__jdbc__	1,273
PNAT	STHZIEGLER40	1
PNAT	ST-NDRACEA-T40	4
PCAT	sgpas02	35
PCAT	STTNGT40	11
PCAT	__jdbc__	897
PCAT	stcshrpas06	125
ADHOC	GYEET30	34
ADHOC	STCGGU01	11
ADHOC	STCNGT40	5
ADHOC	__jdbc__	328
ADHOC	prolin01	33
ADHOC	prolin02	2
ADHOC	STATRANT41	18
ADHOC	STBQUINT41	13
ADHOC	STCPARKT41	5
ADHOC	STEDEANT41	83
ADHOC	STCGLIGHT02	51
ADHOC	STCJSIWEK02	1
ADHOC	stcshrpas06	984
ADHOC	STAFOSTERT42	12
ADHOC	STCDCANNON02	15
ADHOC	STCDGILLIS01	6
ADHOC	STCHJALILI02	54
ADHOC	STCJHARDIN02	1
ADHOC	STCNORTONT41	11
ADHOC	STCSPRASAD02	9
ADHOC	STJARCHERT41	41
ADHOC	ST-LKWANG-T41	18
ADHOC	STBWEVODAUT42	1

```
ADHOC           STCARINTOUL04                                  28
ADHOC           STCGMINYARD01                                  14
ADHOC           STHZEIGLERT41                                  23
ADHOC           STR-PJAIN-T41                                   4
ADHOC           STRSCOTIGERT3                                   3
ADHOC           ST-NDRACEA-T40                                  3
ADHOC           localhost.localdomain                          13
DWETL           prolin01                                       21
DWETL           stcdwhdd                                       10
DWETL           stcdwhpd                                   15,332
DWETL           stcdwhqd                                       10
DWETL           STCMSTDIS01                                     1
PROLO           __jdbc__                                      331
adhoc           __jdbc__                                      329
adhoc           STBWALSHT41                                    37
adhoc           STRMROYCROFTT3                                  3
PROPRD          stcdocpdb01                                   217
REPORT          BRAACTPAS03                                27,071
PROMSG          __jdbc__                                    3,275
PROSVC          __jdbc__                                    2,910
PROSVC          BRAACTPAS02                                     1
PROSVC          BRAACTPAS03                                    29
PROSVC          STBJONEST41                                     1
PROSVC          STCKBASEY01                                     2
PROSVC          STCWALKERT30                                    2
PROSVC          STR-QCHAO-T40                                   3
PROSVC          SW0001022C042B                                 17
proprd          __jdbc__                                      128
proprd          STATHOTANG                                     26
proprd          stcdocpdb01                                     1
PROSVC          __jdbc__                                        6
BOOKING         __jdbc__                                    2,195
PROPEDIA         __jdbc__                                       8
PROCOMM         __jdbc__                                    2,288
PROCOMM         STMJAMEST41                                    12
PROCOMM         STJNELSONT40                                   13
PROCOMM         STDFANELLIT41                                  12
PROSRCH         __jdbc__                                    7,538
PROSRCH         STSPATELT40                                    16
PROSRCH         STJEILERST40                                   79
PROSRCH         STJSABOTKAT40                                   5
PROSRCH         STR-ODADA-T40                                   6
PROSRCH         stcbkgpas01.proligence.com                  2
proprd1         __jdbc__                                        9
PROMEET        __jdbc__                                       976
PROMEET        STADHIMANT41                                    1
PROMEET        STNHEBBART41                                    3
PROMEET        STACHISHOLT42                                  20
RESPONSE        __jdbc__                                      181
response        __jdbc__                                      475
PROPRD_ADHOC    __jdbc__                                        1
service_name    STDFANELLIT41                                   1
service_name    ST-NDRACEA-T40                                  1
```

The data previously shown reveals some very interesting information. Many sessions (shown towards the top of the output with the SN field as NULL) do not use Service Name. Now is the time to target them for a servic name transition.

Second, note the following lines:

```
proprd          __jdbc__                                      128
proprd          STATHOTANG                                     26
proprd          stcdocpdb01                                     1
```

Even though the service name column is not NULL, it shows the service name as proprd, the default service name; this means that the users connected as SID, or used the default service name, which should also be discouraged.

Third, note the following lines:

```
PROPRD_ADHOC    __jdbc__                                          1
service_name    STDFANELLIT41                                     1
service_name    ST-NDRACEA-T40                                    1
```

This shows service names used by clients as PROPRD_ADHOC and service_name. These are not valid servic names defined within the database. How could the user utilize them? Let's take a closer look:

```
select * from listener_log
where
   parse_listener_log_line(connect_string,'SERVICE_NAME') in
      ('PROPRD_ADHOC', 'service_name')
/
```

The output is:

```
LOG_DATE
---------
CONNECT_STRING
--------------------------------------------------------------------------------
PROTOCOL_INFO
--------------------------------------------------------------------------------
ACTION          SERVICE_NAME    RETURN_CODE
--------------- --------------- -----------
18-OCT-05
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=__jdbc__)(USER=))(SERVICE_NAME=PROPRD_ADHOC)
)
(ADDRESS=(PROTOCOL=tcp)(HOST=10.14.105.175)(PORT=3025))
establish       PROPRD_ADHOC          12514
24-OCT-05
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=service_name)(FAILOVER_MODE=(TYPE
=
SELECT)(METHOD=BASIC)(RETRIES=120)(DELAY=2))(CID=(PROGRAM=C:\Program
Files\Quest
 Software\Toad for Oracle\TOAD.exe)(HOST=STDFANELLIT41)(USER=dfanelli)))
(ADDRESS=(PROTOCOL=tcp)(HOST=10.14.105.101)(PORT=2721))
establish       service_name          12514
27-OCT-05
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=service_name)(FAILOVER_MODE=(TYPE
=
SELECT)(METHOD=BASIC)(RETRIES=120)(DELAY=2))(CID=(PROGRAM=C:\Program
Files\Embar
cadero\DBA700\DBArt700.exe)(HOST=ST-NDRACEA-T40)(USER=ndracea)))
(ADDRESS=(PROTOCOL=tcp)(HOST=10.14.104.113)(PORT=3948))
establish       service_name          12514
```

Note the RETURN_CODE column, which is not zero in all cases. This indicates the connection attempt was not successful. Noting the hostnames and users, we can ask the users if they are having issues connecting to the database. If they continue to have issues, we can tell them exactly why, and again, we look like heroes. (Yeah, right!)

Finally, note the following lines:

```
DBA             STATHOTANG                                       15
DBA             STANANDAT42                                      31
DBA             STCJSIWEK02                                      12
DBA             STNRAZAKT41                                       3
DBA             STSZHANGT40                                       8
```

```
DBA             stcudtpdb03                                      6
DBA             STRKGOVINDAT30                                   3
DBA             STSCOTIGERAT42                                   1
DBA             STUMSTEARMAT41                                  25
```

This output shows the hostnames connecting to the service name DBA. You may have allowed this special service name much higher resource utilization in the Resource Manager. So, if people start using this service name, you may have an issue. Going through the output, you could PROLOnize all the hostnames as the laptops of DBAs, except one — STSCOTIGERAT42, which belongs to a non-DBA. Why is that user connecting as DBA? Let's take a closer look:

```
select * from listener_log
where   parse_listener_log_line(connect_string,'SERVICE_NAME') = 'DBA'
and   parse_listener_log_line(connect_string,'HOST') = 'STSCOTIGERAT42'
```

The output is:

```
LOG_DATE
---------
CONNECT_STRING
--------------------------------------------------------------------------------
-
PROTOCOL_INFO
--------------------------------------------------------------------------------
-
ACTION          SERVICE_NAME    RETURN_CODE
--------------- --------------- -----------
02-NOV-05
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=DBA)(FAILOVER_MODE=(TYPE=SELECT)(
M
ETHOD=BASIC)(RETRIES=120)(DELAY=2))(CID=(PROGRAM=C:\Program Files\Quest
Software
\TOAD\TOAD.exe)(HOST=STSCOTIGERAT42)(USER=SCOTIGER)))
(ADDRESS=(PROTOCOL=tcp)(HOST=10.14.104.72)(PORT=1365))
establish       DBA                       0
```

The OS user that connected to the database was "SCOTIGERA," who connected through TOAD. She did connect successfully, as shown by the value of the RETURN_CODE – zero. Looks like she only connected onc on November 2. How else has she been connecting?

```
select * from listener_log
where parse_listener_log_line(connect_string,'HOST') = 'STSCOTIGERAT42';
```

The output is still the same record. So, she has only connected once to production, using the service name DBA. Now it's a time to pay her a visit and point out to her that she should change her service name to a different one.

DBA Service Names

Now that you have found that people have been using the service name DBA even though they are not DBAs, you are sure to be tempted to find out who else may be using it. Well, let's ask the tool:

```
col l_user format a15
col l_host format a30
```

```
col cnt format 999,999
select
   parse_listener_log_line(connect_string,'USER') l_user,
   decode (
      parse_listener_log_line(connect_string,'HOST'),
      '__jdbc__',
      parse_listener_log_line(protocol_info,'HOST'),
      parse_listener_log_line(connect_string,'HOST')
   ) l_host,
   count(1) cnt
from listener_log2
where
   parse_listener_log_line(connect_string,'SERVICE_NAME') = 'DBA'
group by
   parse_listener_log_line(connect_string,'USER'),
   decode (
      parse_listener_log_line(connect_string,'HOST'),
      '__jdbc__',
      parse_listener_log_line(protocol_info,'HOST'),
      parse_listener_log_line(connect_string,'HOST')
   )
order by 1,2,3
```

The previous DECODE statement was necessary since the HOST parameter in CONNECT_STRING does not contain the hostname in case of thin JDBC drivers; it contains the string "__jdbc__". In this case, we should look into the HOST parameter inside the PROTOCOL_INFO, which will show the IP address of the client.

The output is as follows:

```
L_USER          L_HOST                          CNT
--------------- ------------------------------ --------
achakrap        STSZHANGT40                       6
ananda          STANANDAT42                      30
athotang        STATHOTANG                       16
athotang        stcudtpdb03                       2
julisiw         STCJSIWEK02                       6
kgovinda        STRKGOVINDAT30                    4
mstearma        STUMSTEARMAT41                   20
oracle          stcudtpdb03                       1
praprol         prolin01                          2
SCOTIGER        STSCOTIGERAT42                    1
```

From the output, we see all the users and hostnames connecting as DBA service name. We can account for all except the user "SCOTIGER," who is not a known DBA. We have identified that in an earlier step, so it's not a big issue. This output just confirmed that she was the only one; there has not been a rampant abuse of the DB service name.

Conclusion

In part 2 of the article series, you learned how to use this tool to reveal some more interesting and practical information — how Service Name is utilized by the users and from which machines the database connections are coming. With this information, you can steer users to employ a service name, and you can use it as a tracking tool, to monitor progress. The other information — the list of client machines — is very helpful when you are building a perimeter defense around the database servers, using a firewall or using connection manager to limit connections. In the third and concluding article of the series, you will learn how to extract even more constructive information and also how to use it for proactive security establishment.

--

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.

DBA Service Names - Minor correction

DBA Service Names script shows
...
from listener_log2
...
should be
from listener_log