Oracle Forensics: Part 3
Isolating Evidence of Attacks
Against the Authentication Mechanism

David Litchfield [davidl@ngssoftware.com]
27th March 2007

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

One of the most important questions a forensic examiner should seek to answer during an incident response is, "Was there a successful breach?" One way of answering this question in part is to ascertain whether there was a successful logon or not. (I say "in part" because a breach doesn't necessarily require a logon as in the case of a pre-authentication overflow.) In this section we'll look at attacks against the authentication mechanism and evidence from the TNS Listener log file and audit trail, assuming CREATE SESSION is audited of course, and to check whether a logon attempt was successful or not. We'll also look at other attacks levelled at the authentication process including SID guessing, user enumeration and brute forcing of passwords over the network. We'll also look at the differences between a logon attempt via the FTP and Web services provided with the XML Database and directly with the RDBMS itself.

Spotting attempts to obtain the database Service Identifier

To be able to log into the RDBMS an attacker needs to know the Service Identifier or SID for the database. Before Oracle 10g this could be extracted from the TNS Listener with the SERVICES or STATUS command. The following entries are from the Listener log file after someone has performed such an information gathering exercise:

```
…
16-MAR-2007 13:02:31 *
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=)(USER=DAVID))(COMMAND=services)(A
RGUMENTS=64)(SERVICE=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.16
8.1.100)(PORT=1521))))(VERSION=169869824)) * services * 0
…
…
16-MAR-2007 13:02:32 *
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=)(USER=DAVID))(COMMAND=status)(ARG
UMENTS=64)(SERVICE=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.100)(PORT=1
521)))(VERSION=169869824)) * status * 0
…
```

Note that the IP address is not that of the requesting user but that of the server – so it's not an item of interest. The VERSION string is 169869824, which in hex reads as 0xA200200. This gives the version as 10.2.0.2.0 and indicates the version of the client software in use which may prove useful at a later stage of an investigation. Please note though that this value is entirely under the control of the client and may not actually be true. For example, an attacker can write their own Oracle client and spoof these values. We can also see the requesting username of "DAVID". This is the name of the OS account the user is logged on as. Again, this may prove useful, but bear in mind that this too is under the control of the attacker. We'll talk more about this in a moment.

Brute forcing the Service Identifiers

As stated previously, 10g no longer hands over the SID so easily and attackers need to find other means of getting the SID. One way is to attempt to guess it by a brute force attack. There are tools available on the internet to do this such as CQure.net's sidgusser.exe, Red Database Security's sidguess.exe and DatabaseSecurity.com's ora-brutesid.exe, part of the Oracle Assessment Kit. The log entries below have been created by the Listener after running cqure.net's sidguesser.exe program.

```
25-MAR-2007 20:03:54 *
(CONNECT_DATA=(SID=WINDOWS901)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
* (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=2145)) * establish
* WINDOWS901 * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 20:03:54 *
(CONNECT_DATA=(SID=WINDOWS902)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=)))
* (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=2146)) * establish
* WINDOWS902 * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 20:03:54 *
(CONNECT_DATA=(SID=WINDOWS9021)(CID=(PROGRAM=)(HOST=__jdbc__)(USER=))
) * (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=2147)) *
establish * WINDOWS9021 * 12505
```

```
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
```

Sidguesser.exe creates its own TNS packets and is written so that it appears as if the connecting client is using JDBC. We of course "know" that this was cqure.net's sidguesser.exe – because I ran it to get the signature – but in a real world scenario we can only say it matches the signature of the tool. For example, an attacker could have implemented their own SID guesser using JDBC – which would appear in the log file as the same as sidguesser. The next set of entries has been generated by Red-Database-Security's sidguess.exe program.

```
25-MAR-2007 19:52:30 *
(CONNECT_DATA=(SID=ORA8)(CID=(PROGRAM=C:\backup\sidguess.exe)(HOST=AP
OLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4333)) * establish *
ORA8 * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 19:52:30 *
(CONNECT_DATA=(SID=ORA805)(CID=(PROGRAM=C:\backup\sidguess.exe)(HOST=
APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4334)) * establish *
ORA805 * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 19:52:30 *
(CONNECT_DATA=(SID=ORA806)(CID=(PROGRAM=C:\backup\sidguess.exe)(HOST=
APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4335)) * establish *
ORA806 * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
```

This tool uses the Oracle client libraries and, as such, provides "extra" information in the log file – such as the hostname of the connecting client (APOLLO) and the OS username of the person running the tool – "DAVID". As already mentioned, whilst this looks like it may be useful evidence it's also worth bearing in mind that as these are both under the control of the client and these values may be "spoofed". There is nothing to stop someone writing their own SID guesser that sets the host to say, "GW.ORACLE.COM" and the user to "L.ELLISON". These entries are from ora-brutesid.exe (www.databasesecurity.com).

```
25-MAR-2007 20:38:43 * (CONNECT_DATA=(SID=2P)) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4051)) * establish *
2P * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 20:38:43 * (CONNECT_DATA=(SID=3P)) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4052)) * establish *
3P * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
25-MAR-2007 20:38:43 * (CONNECT_DATA=(SID=4P)) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=4053)) * establish *
4P * 12505
TNS-12505: TNS:listener does not currently know of SID given in
connect descriptor
```

This tool creates its own TNS packets and you'll note that there is very little "extra" information. As already indicated, this "extra" information is under the control of the client and they can choose not to send any. The lack of information becomes a "signature" of the tool, in and of itself.

Assuming that the attacker is successful in getting a SID via the "status" or "services" Listener command or manages to guess one then they can then attempt to authenticate.

Spotting user enumeration attacks

During Oracle authentication the client presents their username to the server in one packet. If the username exists in the database then the server issues a session key and the client sends over their encrypted password in a second packet. If the user does not exist the server sends back an error: `ORA-01017: invalid username/password; logon denied`. Due to this difference in behaviour, it's possible for an attacker to determine whether a given user account exists or not. [Please note, however, that Oracle has recently now fixed this in some versions.] Because the client only needs to send the server the first packet to determine if the account exists or not, full authentication is not attempted. How does this play out with the audit trail? Well, again, it depends on whether the account exists or not. If the account does not exist an entry is created in the audit trail with a 1017 return code:

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;

USERID              ACTION# RETURNCODE TIMESTAMP#
------------------- ------- ---------- -------------------
NOSUCHUSER0         100     1017       2007-03-25 23:42:38
NOSUCHUSER1         100     1017       2007-03-25 23:42:38
NOSUCHUSER2         100     1017       2007-03-25 23:42:38
NOSUCHUSER3         100     1017       2007-03-25 23:42:38
NOSUCHUSER4         100     1017       2007-03-25 23:42:39
```

If however the account does exist no entry in the audit trail is created. This is because authentication was never completed so no audit row is added. If the audit trail shows a large number of entries for nonexistent users, especially in a short space of time and from the same location, then this probably indicate a user enumeration attack. So what of the location – how do we know where the attempt has come from? The COMMENT$TEXT column contains the IP address of the system attempting to logon:

```
SQL> SELECT USERID, COMMENT$TEXT FROM SYS.AUD$;

USERID       COMMENT$TEXT
-------      ------------------------------------------------
NOSUCHUSER0  Authenticated by: DATABASE; Client address: (ADDRESS=
(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=2223))
```

What about the TNS Listener's log file? What does it tell us about an attack? This answer to this depends on the tool that was used to enumerate users. A little known fact is that there is no need to create a new TCP connection for each authentication attempt. Once connected to the server, an attacker can keep piping enumeration attempts down the same connected TCP circuit. They can do this for as long as they

want – and the same goes for *full* authentication attempts. To the author's knowledge there is currently only one tool that uses the same TCP connection – ora-brutesid.exe, part of the Oracle Assessment Kit. Most tools don't use the same connection though and will reconnect to the Listener first with every attempt filling the log with multiple "establish" entries.

```
..
25-MAR-2007 23:42:38 *
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl.databasesecurity.c
om)(CID= (HOST=APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1769)) * establish *
orcl.databasesecurity.com * 0
25-MAR-2007 23:42:38 *
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl.databasesecurity.c
om)(CID= (HOST=APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1772)) * establish *
orcl.databasesecurity.com * 0
25-MAR-2007 23:42:38 *
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl.databasesecurity.c
om)(CID (HOST=APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1774)) * establish *
orcl.databasesecurity.com * 0
25-MAR-2007 23:42:39 *
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl.databasesecurity.c
om)(CID= (HOST=APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1776)) * establish *
orcl.databasesecurity.com * 0
25-MAR-2007 23:42:39 *
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl.databasesecurity.c
om)(CID= (HOST=APOLLO)(USER=david))) *
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1778)) * establish *
orcl.databasesecurity.com * 0
..
```

Here we can see multiple "establish" entries in a short space of time from 192.168.1.64. If we stop and think about it for a moment though, this snippet from the log file does not prove a user enumeration attack took place – all that is shows is that there were multiple "establish" events. But when combined with evidence from the audit trail it looks more likely: the timestamps for both the log and the audit trail match and the accounts listed in the audit trail don't exist. As already mentioned, a more advanced tool won't reconnect to the listener for each account name guess and all that will appear in the listener log file is one "establish" entry so the evidence from the audit trail – multiple non-existent accounts – is all that can be relied upon from the database server itself.

Password Guessing Attacks

Once an attacker has a list of accounts they can then attempt to guess their passwords. A simple password brute forcer will make a new connection to the server for each new attempt and this entails connecting to the Listener again and this will be logged by the Listener as an "establish" entry. The log entries discussed in the previous section on account enumeration are what would appear as well for a multiple password guess attempt using a simple tool. They are indistinguishable. If you look back at these log entries, we note that there is a service request and the source port of the connecting client is incrementing by one. This is standard behaviour when a new

TCP connection is created. However, we can't definitely say that such an entry is the result of an attacker performing a brute force attack. For example it could simply be a mis-configured application server attempting to authenticate as it has the wrong username and password. The IP address in the log entry would indicate whether the connecting host is "known" so to speak and should help resolve the question.

A more advanced brute force tool, such as the Oracle Assessment Kit's ora-pwdbrute.exe (www.databasesecurity.com) will not reconnect however for each password attempt. As already mentioned, there is no need to tear down the connection and reconnect. Once the TNS Listener has handed the client off to the server the client can attempt to authenticate as many times as they like. Accordingly, only one entry is logged in the TNS Listener's log file. One of the odd things about ora-pwdbrute.exe is the way in which Oracle records the return code in the audit log of some of the attempts: 1005. This error code indicates the following:

```
ORA-1005: null password provided; logon denied
```

It seems that Oracle is slightly confused by multiple attempts coming down the same pipe and so fast – orapwdbrute.exe can perform 170 attempts per second over the network – this is just short of 15 million password guesses in a day – more than enough to cover all passwords up to five characters long. The snippet from the audit trail below shows the 1005 return code:

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;
..
SCOTT             100       1017 2007-03-27 02:50:17
SCOTT             100       1017 2007-03-27 02:50:17
SCOTT             100       1005 2007-03-27 02:50:17
SCOTT             100       1005 2007-03-27 02:50:17
SCOTT             100       1017 2007-03-27 02:50:17
SCOTT             100       1017 2007-03-27 02:50:17
..
```

A network capture indicates that the password is not null – despite the entry.

Another useful source of information that may indicate a brute force attempt, especially if auditing is not enabled, is the LCOUNT column of the USER$ table. If account lockout is enabled, which it is in 10g by default, for each failed log in attempt the value in this column for the user in question is incremented by 1.

```
SQL> SELECT NAME, LCOUNT FROM USER$ WHERE LCOUNT>0;
```

If account lockout is enabled and the account becomes locked out then the LCOUNT will not go above the specified lockout threshold. So, if the user's profile specifies 10 bad passwords before the account becomes locked then LCOUNT will only ever be a maximum of 10. If the account is unlocked, using the ALTER USER name ACCOUNT UNLOCK statement then the LCOUNT is reset to 0. The LCOUNT will also be set to 0 when the user successfully logs in. Lastly, with regards to the LCOUNT column, if the user account in question has a profile where FAILED_LOGIN_ATTEMPTS is set to UNLIMITED then the LCOUNT will not change – it will always be 0.

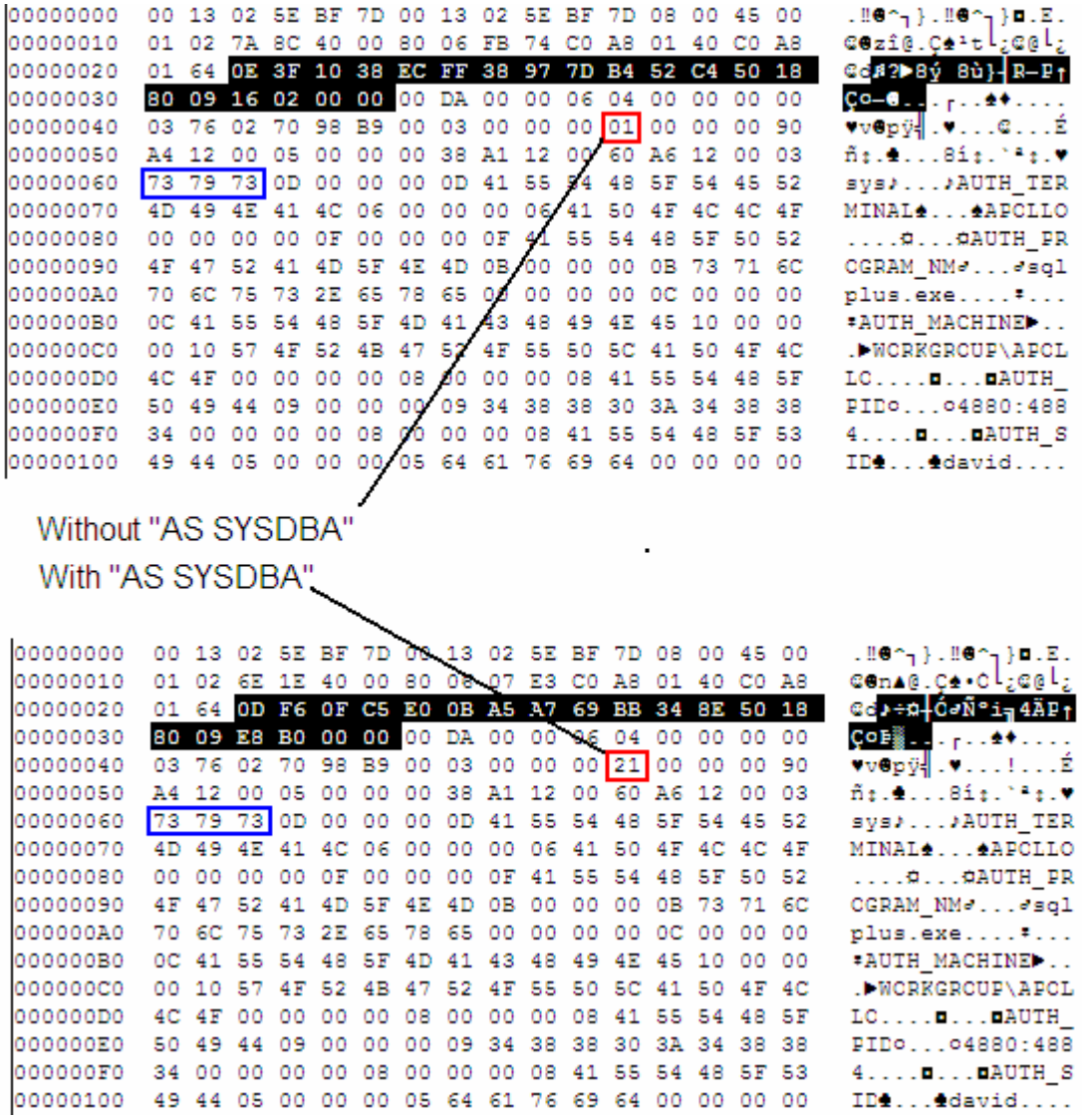
Assuming though that account lockout is enabled for the user then there is another bit of information that is useful when building a time line of events. There is a date column that records when the account was actually locked out to the nearest second.

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';

Session altered.

SQL> SELECT NAME, LTIME FROM USER$ WHERE ASTATUS = 4;

NAME                           LTIME
------------------------------ -------------------
SCOTT                          2007-03-25 08:27:29
```

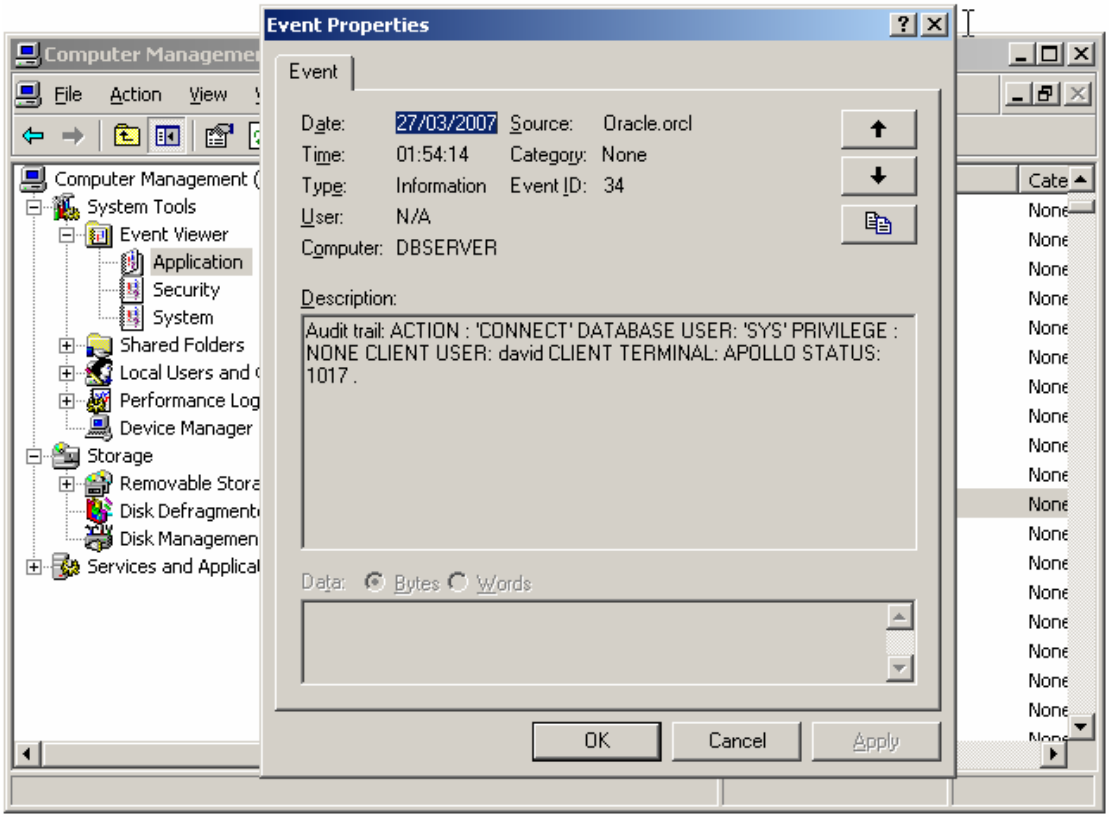
It's important to note though that if the account is unlocked at some point the LTIME will still show the time that the account was last locked out – it is not cleared like the LCOUNT column.

Brute forcing the SYS account

What's the difference between attempting to authenticate as the SYS user with "AS SYSDBA" and attempting to authenticate as the SYS user without "AS SYSDBA"? On the wire – not a lot – just the flipping of a bit flag:



This flag byte is used to indicate the requested privilege level of the connection. Bit 6 is set if connecting as SYSDBA and bit 7 is set if connecting as SYSOPER. The author has observed the 8th bit being set in certain situations but isn't sure yet what it means. Even if neither the 6th or 7th bits are set an attacker can still attempt to logon as SYS and work out if they have the correct password: if the password is wrong they'll get an `ORA-01017: invalid username/password; logon denied` error. If the password is correct they'll get an `ORA-28009: connection to sys should be as sysdba or sysoper`. This difference in response let's them know they've managed to get the password correct and then they can logon using with "as sysdba". Whilst there's only a single bit flipped as far as the network traffic is concerned there's a world of difference as far as the audit trail is concerned. If "AS SYSDBA" or "AS SYSOPER" is not specified during the logon attempt an entry is made in the AUD$ table. If the attempt was unsuccessful in guessing the password the return code is 1017; if the attacker manages to guess the password the return code will be 28009. If, however, the attacker sets the privilege level to SYSDBA or SYSOPER then nothing is logged in the audit trail. We need to look elsewhere: when it comes to SYSDBA and SYSOPER privileged connections Oracle writes an entry to the operating system's logging system. On Windows this is the Application Log of the Event Logging Service:



On *nix this messages are sent to the syslog daemon. If there are multiple entries in the operating system's log with a status code of 1017 then this indicates a brute force attack against the SYS account with the privilege bit set.

Attempts to exploit the AUTH_ALTER_SESSION flaw

Assuming a user either already has a username and password or manages to guess one they can attempt to exploit a well known flaw in the last stage of authentication. Once the client's username and password have been validated the client executes an ALTER SESSION statement. In November 2005 Imperva independently rediscovered a flaw that had already been found internally by Oracle during the execution of this statement: it executes with SYS privileges. As such an attacker could change the statement from an ALTER SESSION to a GRANT DBA which would then execute and succeed when the client logs in. This flaw was patched in the January 2006 Critical Patch Update. Provided auditing is enabled for CREATE SESSION then it is possible to spot attempts to exploit this flaw on patched systems. If the attacker attempts to execute any SQL that they didn't already have the permissions to execute then an error is generated: `ORA-00604: error occurred at recursive SQL level 1. ORA-01031: insufficient privileges`. This appears in the audit trail with a return code of 604.

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;
---------------------- ----    ---    -------------
..
SCOTT                  100     604    26-MAR-07
..
```

If the server has not been patched then the return code will be 0 which simply indicates a successful logon. To find evidence of an attacker exploiting this flaw on an unpatched system a forensic examiner will need to search elsewhere; the remaining chapters cover this.

Spotting attempts to log in via the XML Database (XDB)

The XML Database provides two network services – a FTP server listening on TCP port 2100 and a Web Server listening on TCP port 8080. Technically the TNS Listener holds open these ports and when a connection request comes in, it hands it off to the database server. Because the TNS Listener plays a role in this, albeit small, the connection is logged in the Listener's log file. If the connection is to the web service then it will read like so:

```
25-MAR-2007 21:07:25 * http * handoff * 0
```

If the connection is for the FTP service then there will be an entry that reads as follows:

```
27-MAR-2007 03:07:46 * FTP * handoff * 0
```

There's not a great deal of information there – just a time stamp and the fact that someone connected – we don't know who and where from though – no IP address is logged. If we look in the audit trail though we do see a successful logon entry with the same timestamp though:

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;
..
DBSNMP            100          0 2007-03-27 03:07:46
..
```

We can get the IP address information from the COMMENT$TEXT column but if we look at it there's nothing to differentiate it from a normal database logon:

```
SQL> SELECT COMMENT$TEXT FROM SYS.AUD$ WHERE USERID = 'DBSNMP';

COMMENT$TEXT
------------------------------------------------------------
Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.64)(PORT=1324))
```

Whilst it's not bullet proof – this can be confirmed as an XDB logon because when you select the TERMINAL and the SPARE1 (which stores the OS username) columns they're empty:

```
SQL> SELECT TERMINAL,SPARE1,TIMESTAMP# FROM SYS.AUD$ WHERE USERID=
'DBSNMP';

TERMINAL     SPARE1     TIMESTAMP#
-----------  ------     ----------
                        2007-03-27 03:07:46
```

I say that it's not bullet proof because, recall, that the client can choose whether to send over this information in a normal RDBMS connection so it too would appear like this.

Auditing not enabled?
If auditing is not enabled and you're running Oracle 10g then there still may be evidence showing which users were logged on when. The fixed view V$ACTIVE_SESSION_HISTORY uses a circular buffer in the SGA to store sampling information taken every second about active sessions. These sessions are flushed from the SGA to the WRH$_ACTIVE_SESSION_HISTORY table every so often, as part of 10g's Automatic Workload Repository. This historical data therefore contains information that is useful to a forensic examiner as it effectively records who was logged on when.

```
SQL> SELECT USER_ID, SESSION_ID, SAMPLE_TIME FROM
SYS.WRH$_ACTIVE_SESSION_HISTORY ORDER BY SAMPLE_TIME;

USER_ID SESSION_ID   SAMPLE_TIME
------- ----------   -----------------
0       149          27-MAR-07 06.58.26.127
24      142          27-MAR-07 07.02.16.140
..
..
1227 rows selected.
```

Here we can see that user 24 (dbnsmp) was logged on at two minutes past seven in the morning on the 27th of March 2007. In the absence of audit information this data can be extremely useful to help work out who was logged on when.

Lastly, a gotcha!
Here's something to be careful of with the audit trail. When a user successfully logs on a row is created in the audit trail. This has an ACTION# number of 100 (LOGON) and the TIMESTAMP# column reflects when the logon occurred:

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;

USERID                ACTION# RETURNCODE TIMESTAMP#
--------------------- ---------- ---------- -------------------
DBSNMP                100          0 2007-03-27 03:31:49
```

Then the user logs off. If we check the audit trail again we see that the ACTION# is changed from 100 (LOGON) to 101 (LOGOFF) but the timestamp remains the same!

```
SQL> SELECT USERID, ACTION#, RETURNCODE, TIMESTAMP# FROM SYS.AUD$;

USERID          ACTION# RETURNCODE TIMESTAMP#
--------------- ---------- ---------- -------------------
DBSNMP          101          0 2007-03-27 03:31:49
```

In building a timeline of events this is important. This effectively hides when the user actually logged on. However, if we describe the AUD$ table we can see a LOGOFF$TIME column. If we then query this column, too, we can reconcile the logon and logoff times:

```
SQL> select userid,action#,TIMESTAMP#,LOGOFF$TIME from aud$;

USERID        ACTION# TIMESTAMP#          LOGOFF$TIME
---------- ---------- ------------------- -------------------
SCOTT             101 2007-04-04 22:56:37 2007-04-04 23:00:00
```

Here we can see the action has been updated to 101 and the TIMESTAMP# column shows when the user logged on and the LOGOFF$TIME is when the log off occurred.

Wrapping Up

We can potentially find evidence of authentication attacks in both the TNS Listener's log file and the audit trail. We have also seen that different tools leave different footprints – and some leave very little when it comes to the Listener log file. Pay close attention to the return code in the audit trail. Lots of 1017 (and possibly 1005) entries with an action of 100 for a given user followed by a 0 indicate a successful brute force attack. Recall though that Oracle has many default accounts with default passwords and these will be the first choice for an attacker. It's all too common to see entries for DBSNMP with a return code of 0.