Oracle Forensics Part 4:
Live Response

David Litchfield [davidl@ngssoftware.com]
20th April 2007

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

An organization should have a clear understanding of what actions should be taken in the event of an incident occurring. For those that don't have a plan often the knee-jerk response is to pull the plug or disconnect the system from the network. This prevents further incursions and theft of data so it is an understandable reaction to have. In taking this action however, useful evidence such as volatile, in-memory data may be lost. Or even worse – consider the case where a logic bomb has been planted. Assuming an attacker compromises an Oracle database with DBA privileges they could create the following trigger:

```
CREATE OR REPLACE TRIGGER VANISH BEFORE SHUTDOWN ON DATABASE
BEGIN
DELETE FROM SYS.AUD$;
END;
/
```

When the database is shutdown cleanly this would wipe the audit trail making the task of the forensic examiner that little bit harder. Of course, the attacker could do more than just wiped the audit trail in such a trigger. Due to issues like this and the loss of volatile information, some organizations prefer to perform an analysis on the system whilst it's still powered on and connected to the network. This is called a Live Response. Live Response is all about recovering and safely storing volatile data for later analysis, in other words, all the information that will disappear when the machine is disconnected from the network and switched off. Further, Live Response gives the forensic examiner the chance to collect non-volatile evidence in a "human-readable" format that's easier to peruse than its stored binary version – for example event logs. This paper will start with an overview of general Live Response steps but will mainly focus on those aspects that are Oracle specific. There are many great books available that cover the more general steps of Live Response - may I suggest Real Digital Forensics (Addison-Wesley) by Richard Bejtlich, Keith Jones and Curtis Rose?

Where possible, the incident responder should perform Live Response actions in the presence of a person who is thoroughly intimate with the system being investigated and its configuration. This serves two purposes. Firstly, this person can act as a witness, which, when and if it comes to a court case, will help to strengthen the veracity. Secondly, this person will be able to help the incident responder spot anything suspicious, for example, a new user account or an account with DBA privileges that shouldn't have them, and they can also help direct the incident responder on questions relating to the system's set up.

Before getting into the specifics of Oracle Live Response I first want to talk about trust and assurance. Because a Live Response necessarily performs an analysis on an already compromised system can we trust the information we get from it? Can we be assured that the information is accurate? Let me further explain the issue with an example. When performing a Live Response on say a Windows server, the incident responder will have their tools all stored on a read-only CD. When they insert the CD and run one of the tools, due to the way Windows launches new processes, the tool will have key system dynamic link libraries in its address space, i.e. the memory the tool uses. Getting slightly more technical about it, when a program is executed, the shell from where the program is launched from will call CreateProcess() which in turn calls NTCreateProcess(). The kernel then creates a process object, allocates some

memory then maps ntdll.dll into that memory. The code of the program is then loaded into memory and any DLLs it uses are also loaded. The program is then started. At the stage where ntdll.dll is loaded into the process object's memory the trust and assurance may have already been compromised: if an attacker has full control of the system then they can patch, in memory, ntdll.dll so that it performs actions that the attacker fully controls. The same goes before *nix based platforms as well. If an attacker has control over the system then they can fake the responses tools generate.

Due to this alone one may question the benefit of Live Response but there is another problem. Due to the nature of volatile information, once it's gone, it's gone. This means Live Response is not reproducible and as such could be challengeable in court. To help mitigate this, the incident responder needs to fully document everything they do to the system during Live Response and have a witness to corroborate.

So what is the benefit of Live Response then if the results can't be trusted and the results are challengeable? Well, firstly, not every attacker will fully compromise a system meaning that the results generally can't be faked; secondly, most won't perform anti-forensics or hide their attack even if they do "own" the system; thirdly, the information obtained from Live Response can be used immediately to get the analysis going; fourthly, if a Live Response pulls no suspicious information but other networked devices such as firewalls or NSMs indicate there has been a compromise then this becomes evidence in and of itself – an offline analysis should hopefully reveal the presence of rootkit technology.

So with the value of Live Response and its implications as far as trust and assurance are concerned in mind, let's look at Oracle Live Response.

General Steps of Live Response

During Live Response, it is impossible for an incident responder not to leave a footprint on the system which is the target of the investigation but they need to make this footprint as small as possible – they must be *minimally invasive*. For example, the very act of logging onto the server's console will possibly cause information to be written to the system's audit trail; a shell will be started which will affect memory and will cause modifications to the paging file. Changes like these are unavoidable but they are considered as acceptable. What is not acceptable is for an incident responder to create new files on the system such as by redirecting the output of their Live Response tools to a file. This could potentially overwrite blocks on the disk that contains deleted data which would have otherwise been recoverable during an offline analysis. All output from the Live Response tools should be written to a collection server across the network. There are three ways of doing this – firstly by mapping a drive if the system is running on Windows or has Samba and then using file redirection:

```
D:\>listdlls.exe > z:\case-0001-listdlls.txt
```

Using file redirection can be prone to error – for example the incident responder could type C instead of Z – which would be disastrous. The second method is to pipe output over the network using netcat or cryptcat. Cryptcat would be better than netcat as it encrypts data over the network. Bear in mind though that any output sent to "stderr" will not be piped over the network – only output going to "stdout" will:

```
D:\>type output.c

#include <stdio.h>

int main()
{
      fprintf(stdout,"This will be piped/redirected...\n");
      fprintf(stderr,"This will not be piped/redirected...\n");
      return 0;
}
D:\>output.exe | nc 192.168.1.100 7777
This will not be piped/redirected...
D:\>
```

The same is also true of redirection. If you're using pipes or redirection it's important to accurately record any errors that are output to the console via standard error. The alternative to redirection and piping is to use a set of tools that have networking built directly into them such a WebJob by KoreLogic. The collection server can be a laptop but the incident responder should ensure that it has enough free space to hold all of the data which could run into several gigabytes.

The Oracle specific stages of Live Response should be performed last and the more general steps should be taken first. These general steps include getting information on the following:

System time and date
The incident responder should first record the system time and date of system that they're investigating.

Logged on users
The list of users that are currently logged on to the system and from where and for how long is extremely useful.

List all users and groups
Obtain a list of all users, gathering details on when they last logged in, and groups on the server and group membership.

List open ports and connections
Before getting into this let's consider the following question: Do you disconnect the compromised system from the network or not. If you do, with some versions of Windows active connections will be dropped wiping away the information. If you don't disconnect from the network and the attacker is still "on" then more damage could be done. Then again, it provides the incident responder with the chance to gather evidence of the attack *as* it happens. If the value of the data being stolen would be "too expensive" if it got "out" then my advice would be to disconnect; disconnecting a live system handling hundreds of business requests a minute can also be expensive though. It's all a matter of weighing up pros and cons. The answer to the question as to whether to disconnect from the network or not must be weighed up and preferably a decision taken before an incident ever occurs.
All open and connected TCP ports should be collected as well as listening UDP ports. This information can often show how an attacker has managed to gain or maintain

access to the system. Bear in mind one or more of these connections will be incident responder's connection to the collection server. Pay close attention to connections to the Oracle server. Depending upon whether this server is running in shared mode or not will determine whether the client connections are all bound to port 1521 (assuming this is the Oracle port of course) or random high ports. Another interesting telltale sign to look out for is many ports that show a SYN sent state. These are indicative of a TCP scan being performed from the host being investigated.

As well as getting port information the incident responder should ascertain what process is listening on what ports.

List running processes
A list of all running processes should be obtained. Close attention should be paid to suspicious looking entries and also any shells such as cmd.exe or /bin/sh – indeed keep an eye out for //bin/sh (note two slashes) as this may indicate an overflow or format string exploit has been launched. The forensic examiner should also get a list of each process's parent process. Sometimes, if the parent process has exited, then this may be made more difficult but an examination of open handles might help reveal the parent's parent.

List of DLLs or shared objects
A list of the DLLs or shared objects that are loaded by each process should be obtained. Keep an eye out for odd looking names; on Windows look out for DLLs that are loaded via a UNC path across the network.

List of open handles
As well as what file handles a process has open a list of other handles should be obtained as well. Whilst this can reveal what an attacker may have been doing it can also help identify "parentless" processes. We'll look at an example of this shortly.

Perform memory dumps
Memory dumps of all running process should be gathered even in what appear to be "normal" looking processes. The reason for this is to catch cloaking attacks – an attacker may launch a benign process like "notepad" and using CreateRemoteThread() load code into its address space.

Perform system memory dump
A dump of all system memory should be performed. This will cover those bit of memory not dumped when dumping each process.

Get file names and MACTimes
The incident responder should perform a full recursive directory list of every disk and get file and directory names as well as their creation, access and modification times. They should also gather information about each file's owner and any special attributes such as whether the read only, system or hidden attributes are set.

Dump registry information
On Windows all registry information should be dumped.

Locate and take copies of log files and message logs

All of the servers log files and event and message logs should be copied to the collection server for analysis. These logs will vary from system to system depending upon what services are running.

Collecting the Oracle files of interest

The Oracle specific log, trace and control files can be located in various places so let's take a quick look at how to find them. Firstly we need to know where each instance of Oracle is installed – this can be extracted from the ORACLE_HOME environment variable if set. On Windows the HKEY_LOCAL_MACHINE\Software\Oracle Registry key stores information about each Oracle home. For each Oracle home the incident responder should locate the server's start up parameter file. This will be found in the "database" directory on a Windows system or the "dbs" directory on a *nix system. Generally the filename is "spfile*sid*.ora" where "*sid*" is the database service identifier. This file contains information about where log and trace files etc are written to.

```
audit_file_dest
background_dump_dest
core_dump_dest
db_recovery_file_dest
user_dump_dest
utl_file_dir
control_files
db_create_file_dest
db_create_online_log_dest_n
log_archive_dest
log_archive_dest_n
```

The incident responder should also be aware that what is listed in the start up file may not actually be what settings the Oracle server is actually using – for example if an attacker has used the "ALTER SYSTEM" or "ALTER DATABASE" command. We'll talk about getting information about this later but for now we'll discuss these locations.

Audit_file_dest

If auditing is enabled and configured to log to the operating system's file system then the audit files will be located in this directory.

Background_dump_dest

This directory contains the alert.log file and the trace files for the background processes.

Core_dump_dest

Oracle server core dumps are written to this location. Core files may indicate a buffer overflow exploitation attempt.

Db_recovery_file_dest

This is the location of the flash recovery area and contains archived redo logs.

User_dump_dest

Trace information for user processes are written to this directory.

Utl_file_dir
This is the location used for PL/SQL file I/O – for example when using UTL_FILE.

Control_files
This parameter is a list of all the control files used by the server. These control files contain information about data files.

Db_create_file_dest
This is the location for Oracle managed data files. It may or may not be set on the server in question. You can also find Oracle managed redo logs and control files here if the db_create_online_log_dest_n parameters have not been set.

Db_create_online_log_dest_n
This is where redo logs can be found. The n should be replaced with a number starting with 1. Again this may or may not be set on the server in question.

Log_archive_dest, log_archive_dest_n and log_archive_duplex_dest
These locations can be used for archived redo logs.

Oracle Data Files
The control files contain the location of the Oracle data files but here there is a potential hurdle. A database server may have many terabytes, perhaps even a petabyte or two of data and it is not generally possible to copy these to the collection server. However, it should be possible to copy the key data files. These include those that form the SYSTEM, SYSAUX, TEMP and UNDO tablespaces. All should be copied to the collection server.

External Files
Oracle can write files to the file system using Java and the UTL_FILE PL/SQL package. The UTL_FILE_DIR parameter dictates where Oracle can read and write files from and to. Files in this directory should be gathered for analysis. If the entry is an asterisk then Oracle can write to anywhere on the file system. We'll also discuss external tables and Oracle "directory" objects as well shortly.

Listener Logs Files
Collect the listener log files – these can be found in the ORACLE_HOME/network/log directory by default. If the log file doesn't exist here the location and name can be obtained from the ORACLE_HOME/network/admin/listener.ora file. As a caveat, the location specified in this file might not be the "live" setting. For example, in Oracle 9 and earlier, if no password has been set on the Listener then an attacker will be able modify the location of the log file – indeed they could even turn it off. Running the "lsnrctl status" command will reveal the actual location but there is an explicit warning with this, too – executing this command will cause a "status" entry to be written to the log file if logging is enabled:

```
C:\oracle\product\10.2.0\db_1\BIN>lsnrctl status

LSNRCTL for 32-bit Windows: Version 10.2.0.2.0 - Production on 06-
APR-2007 01:28:18
```

```
Copyright (c) 1991, 2005, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for 32-bit Windows: Version
10.2.0.2.0 - Production
Start Date                06-APR-2007 01:27:36
Uptime                    0 days 0 hr. 0 min. 41 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File
C:\oracle\product\10.2.0\db_1\network\admin\listener.ora
Listener Log File c:\temp\listener.log
Listening Endpoints Summary...

(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\EXTPROC1ipc)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=APOLLO)(PORT=1521)))
Services Summary...
Service "PLSExtProc" has 1 instance(s).
  Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this
service...
The command completed successfully

C:\oracle\product\10.2.0\db_1\BIN>
```

Here we can see the location of the log file is C:\temp\listener.log. The “status” entry will look similar to the following:

```
06-APR-2007 01:30:04 *
(CONNECT_DATA=(CID=(PROGRAM=)(HOST=)(USER=david))(COMMAND=status)(ARG
UMENTS=64)(SERVICE=LISTENER)(VERSION=169869824)) * status * 0
```

We can see that the time is recorded so before running the “lsnrctl status” command take and record the time.

Querying the server using SQLPlus

So as not to contaminate the available evidence the very last part of a live response an incident responder should do is connect to the database server. All other tasks should be performed first. The incident responder should connect to the database server with SYS privileges if possible. This enables them to gain access to the information required for the analysis. Once connected the incident responder should not take any action that changes the state of the database directly. For example, they should not perform and DML operations such as INSERT, UPDATE or DELETE; they should not CREATE objects – even global temporary tables – or DROP objects; they should not GRANT or REVOKE privileges; and they should not use ALTER – especially not to dump any files such as log files or data files. Be aware that, even despite these warnings, just the very act of connecting to the server may change it in some way: an “establish” entry will be created in the Listener’s log file; rows may be inserted into the audit table (AUD$) or audit file – if not connecting as SYS; log entries will be written to the operating system’s logging system. When SQL is executed this will indirectly change rows in the many of the fixed views such as V$SQL. This is why connecting to the RDBMS itself should be the last action the forensic examiner takes.

We'll assume that the incident responder will use sqlplus, the Oracle command line client, to connect to the server. Before connecting the incident responder should review the following files, if they exist, to ensure that they do not contain any SQL that will performs any of the operations mentioned above:

$ORACLE_HOME/bin/LOGIN.SQL
$ORACLE_HOME/dbs/LOGIN.SQL
$ORACLE_HOME/SQLPlus/admin/glogin.sql

One more warning – remember problems with trust? We can't necessarily trust the system we're performing the Live Response on and this is even more relevant when it comes to the database server itself. Whilst an attacker may not get "root" on the box itself it's fairly trivial to gain DBA privileges as far as the database goes. As part of hiding their access an attacker will often modify the text of views so all queries should be performed against the underlying tables and not views. For example, a hacker could modify the DBA_ROLE_PRIVS view to hide the fact that they have DBA privileges so they won't turn up in a query like:

```
SQL> SELECT GRANTEE FROM DBA_ROLE_PRIVS WHERE GRANTED_ROLE = 'DBA';

GRANTEE
------------------------------
SYS
SYSMAN
SYSTEM
```

However, they will be seen in a query to the underlying tables:

```
SQL> SELECT U.NAME FROM SYS.USER$ U, SYS.SYSAUTH$ A WHERE U.USER# =
A.GRANTEE# AND PRIVILEGE# = (SELECT USER# FROM SYS.USER$ WHERE NAME =
'DBA');

NAME
------------------------------
SYS
HACK101
SYSTEM
SYSMAN
```

Once sqlplus is running and before connecting to the server all queries and all results should be spooled to a file. This can be achieved using the SPOOL command followed by the name of the file. The file name should be "meaningful" – for example, it should contain the case number of the investigation and the date and time:

```
C:\oracle\product\10.2.0\db_1\BIN>TIME
The current time is:  6:12:22.93
Enter the new time:

C:\oracle\product\10.2.0\db_1\BIN>DATE
The current date is: 27/03/2007
Enter the new date: (dd-mm-yy)

C:\oracle\product\10.2.0\db_1\BIN>SQLPLUS /NOLOG
```

```
SQL*Plus: Release 10.2.0.2.0 - Production on Tue Mar 27 06:12:29 2007

Copyright (c) 1982, 2005, Oracle.  All Rights Reserved.

SQL> SPOOL C:\IR-CASES\N0017\SQL-CASE-N0017-27-03-2007-06-12-22.TXT
```

The connection should be made from their laptop over the network, specifying the IP address of the server being investigated, followed by the port number and the database service identifier (SID).

```
SQL> CONNECT SYS/PASSWORD@192.168.1.100:1521/ORCL AS SYSDBA
Connected.
SQL>
```

Once connected they should alter the session and set the date format to include hours, minutes and seconds otherwise data from columns of type DATE will appear as "DD-MON-YY" – for example "16-MAR-07"

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

Getting previously executed SQL

Once connected the very first query the forensic examiner should execute (assuming that they've not had to alter their session) should be the one that gets a copy of the most recently executed SQL. This can be retrieved from the V$SQL fixed view. On Oracle 10g the query should be:

```
SQL> SELECT LAST_ACTIVE_TIME, PARSING_USER_ID, SQL_TEXT FROM V$SQL
ORDER BY LAST_ACTIVE_TIME ASC;
```

This will list the SQL that was executed by who and when from the V$SQL fixed view. There are only a limited number of entries, around 2500 or so, in this view and it is circular – i.e. older entries are overwritten with new entries. If the incident responder can get this information as quickly as possible after the incident is noticed then there is a chance that there will still be evidence present. On a busy server the chances are, of course, lowered, but the query should still be executed. If there is content of value to the investigation then the user id and the time at which the SQL was execute are both available. It should be noted that this information should also be available in the memory dumps performed earlier.

In Oracle 9i there is not LAST_ACTIVE_TIME column so the query should be

```
SQL> SELECT PARSING_USER_ID, SQL_TEXT FROM V$SQL ORDER BY
PARSING_USER_ID ASC;
```

In 10g the Automatic Workload Repository History WRH$_SQLTEXT and WRH$_SQLSTAT tables may also contain evidence and should be queried:

```
SQL> SET LONG 2000000000
SQL> SELECT ST.PARSING_SCHEMA_ID, TX.SQL_TEXT FROM WRH$_SQLSTAT ST,
WRH$_SQLTEXT TX WHERE TX.SNAP_ID = ST.SNAP_ID;
```

The line that reads “SET LONG 2000000000” tells sqlplus to show up to 2000000000 characters for the SQL_TEXT as it will be truncated otherwise being of type CLOB. This table contains a very large number of previous SQL queries and there’s a good likelihood of there being useful evidence available. This contains SELECT, UPDATE, INSERT, DELETE and certain ALTER queries that took a “long time” to execute. Because packages like UTL_INADDR, UTL_HTTP, UTL_SMTP and UTL_TCP require network interaction and therefore “take a long time”, attacks that use them as a data exfiltration method will be found in this table. This is one of the few ways a forensic examiner has of being able to find some SELECT activity in the absence of auditing.

Once everything has been extracted from these tables it should be “safe” to move onto to other queries. “Safe” is enquoted because, as these new queries are executed, they’ll push out entries in the SQL_TEXT from the other previous queries. This is why we queried them first.

Next in line should be the audit log. Everything should be selected from this table for later consumption and analysis.

```
SQL> SELECT * FROM AUD$;
```

If audit information is logged to the server’s file system or to a syslog daemon then this should be gathered before connecting to and querying the database server.

Getting information about logons – current and old

Whilst the AUD$ table will contain information about logons, providing auditing is enabled of course and is logged to the database, further to this it is possible to find evidence of logons elsewhere. 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 the 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
```

The incident responder should get a list of who is currently logged on to the server:

```
SQL> SELECT SID, USER#, USERNAME, TERMINAL, OSUSER, PROGRAM,
LOGON_TIME FROM V$SESSION;
```

Getting a list of users and roles

The incident responder should get a complete listing of all users on the system.

```
SQL> SELECT USER#, NAME, ASTATUS, PASSWORD, CTIME, PTIME, LTIME FROM
SYS.USER$ WHERE TYPE#=1;
```

The password hash should be selected because an offline password audit should be performed to determine how easy it may be for an attacker to correctly guess the

password. The ASTATUS will let the examiner know if the account is locked or not and the LTIME will show when the account was locked. The CTIME shows when the user was created and PTIME shows when the password was last changed.

A list of roles should be selected.

```
SQL> SELECT USER#, NAME, PASSWORD, CTIME, PTIME FROM SYS.USER$ WHERE
TYPE#=0;
```

Getting a list of role memberships
Just in case there are any users that have membership of a role that they should not have the incident responder should get a list:

```
SQL> SELECT U.NAME AS "GRANTEE", U2.NAME AS "ROLE" FROM SYS.USER$ U,
SYS.USER$ U2, SYS.SYSAUTH$ A WHERE U.USER# = A.GRANTEE# AND
PRIVILEGE# = U2.USER#;
```

Look out for all users or roles that have been assigned membership of the DBA role.

Getting a list of object privileges
It will be necessary to get a list of all object privileges that have been granted and to look for anything suspicious.

```
SQL> SELECT U.NAME AS "GRANTEE", P.NAME AS "PRIVILEGE", U2.NAME AS
"OWNER", O.NAME AS "OBJECT" FROM SYS.USER$ U, SYS.USER$ U2,
SYS.TABLE_PRIVILEGE_MAP P, SYS.OBJ$ O, SYS.OBJAUTH$ A WHERE U.USER# =
A.GRANTEE# AND A.OBJ# = O.OBJ# AND P.PRIVILEGE = A.PRIVILEGE# AND
O.OWNER#=U2.USER#;
```

Getting a list of all system privileges
The same goes for system privileges.

```
SQL> SELECT U.NAME AS "GRANTEE", S.NAME AS "PRIVILEGE" FROM SYS.USER$
U, SYS.SYSAUTH$ A, SYS.SYSTEM_PRIVILEGE_MAP S WHERE U.USER# =
A.GRANTEE# AND PRIVILEGE# = S.PRIVILEGE ORDER BY U.NAME;
```

Getting a list of all objects
The incident responder should get a complete list of every object, covering its ID, its owner, its name and type as well as the creation time, modification time and specification time.

```
SQL> SELECT OBJ#, OWNER#, NAME, TYPE#, CTIME, MTIME, STIME FROM
SYS.OBJ$ ORDER BY CTIME ASC;
```

This query will order the results according to when they were last modified.

Getting a list of dropped tables
In 10g, if a user has dropped any tables and they have not been purged from the recyclebin then a list of dropped tables should be present. This may indicate evidence of an attack:

```
SQL> SELECT U.NAME, R.ORIGINAL_NAME, R.OBJ#, R.DROPTIME, R.DROPSCN
FROM SYS.RECYCLEBIN$ R, SYS.USER$ U WHERE R.OWNER#=U.USER#;
```

Getting a list of block changes

Each data file is split into blocks of data and each block contains a time stamp when it was last modified for example after a COMMIT. The time stamp is recorded in an SCN – or System Change Number. When a row is modified the SCN is updated in the block meaning all other rows in the block reflect the same SCN. In terms of analysis this is an important distinction. As an example consider the following query and snippet of results:

```
SQL> SELECT O.ORA_ROWSCN, O.CTIME, O.MTIME, O.STIME, U.NAME, O.NAME
FROM SYS.OBJ$ O, SYS.USER$ U WHERE U.USER#=O.OWNER# ORDER BY 1;

…
…
2282280 05-APR-07 06-APR-07 05-APR-07 SYS WRH$_SERVICE_WAIT_CLASS
2282280 05-APR-07 05-APR-07 05-APR-07 SYS WRH$_SERVICE_WAIT_CLASS_PK
2282280 05-APR-07 05-APR-07 05-APR-07 SYS P_TEST
2282280 05-APR-07 05-APR-07 05-APR-07 SYS VP_TEST
2282280 05-APR-07 05-APR-07 05-APR-07 SYS P_TEST
2282280 05-APR-07 05-APR-07 05-APR-07 SYS VANISH
2282280 05-APR-07 09-APR-07 09-APR-07 SYS X
2282280 05-APR-07 06-APR-07 05-APR-07 SYS WRH$_SQLSTAT
2282280 05-APR-07 06-APR-07 05-APR-07 SYS WRH$_SYSTEM_EVENT
…
…
```

If we convert the SCN to a time stamp we can see the change occurred on the 9th April 2007:

```
SQL> SELECT SYS.SCN_TO_TIMESTAMP(2282280) FROM DUAL;

SYS.SCN_TO_TIMESTAMP(2282280)
---------------------------------------------------------
09-APR-07 14.39.56.000000000
```

Looking through the results from the query before we see that the object called X has a MTIME and STIME set to this date and it is the only object that matches. Thus we can see that the SCN for all the other objects (read rows) in the same block share the same SCN even though they have not been modified. Here is another example:

```
SQL> SELECT U.ORA_ROWSCN, U.NAME FROM SYS.USER$ U WHERE TYPE#=1 ORDER
BY 1;

ORA_ROWSCN NAME
---------- ------------------------------
    537106 EXFSYS
    537106 DMSYS
    537106 TSMSYS
    537106 DBSNMP
    537106 ANONYMOUS
    537106 XDB
    537106 CTXSYS
    537106 WMSYS
   1465169 OUTLN
   1465169 DIP
   1465169 SYS
   1465169 SYSTEM
```

```
   2277427 MARK_POINT2
   2277427 PWDTEST
   2277427 MARK_POINT
   2277427 FINDME_TOO
   2277427 FINDME
   2277427 SCOTT
   2277427 MGMT_VIEW
   2277427 MDDATA
   2277427 SYSMAN
   2277427 MDSYS
   2277427 SI_INFORMTN_SCHEMA
   2277427 ORDPLUGINS
   2277427 TESTUSER
   2277427 OLAPSYS
   2277427 ORDSYS

27 rows selected.
```

Users MARK_POINT2, PWDTEST, MARK_POINT, FINDME_TOO, FINDME, SCOTT, MGMT_VIEW, MDDATA, SYSMAN, MDSYS, SI_INFORMTN_SCHEMA, ORDPLUGINS, TESTUSER, OLAPSYS and ORDSYS all exist in the same block and therefore share the same SCN. If we then alter SCOTT's password the SCN should change for all these users:

```
SQL> ALTER USER SCOTT IDENTIFIED BY PASSWORD;

User altered.

SQL> SELECT U.ORA_ROWSCN, U.NAME  FROM SYS.USER$ U WHERE TYPE#=1
ORDER BY 1;

ORA_ROWSCN NAME
---------- ------------------------------
    537106 EXFSYS
    537106 DMSYS
    537106 TSMSYS
    537106 DBSNMP
    537106 ANONYMOUS
    537106 XDB
    537106 CTXSYS
    537106 WMSYS
   1465169 OUTLN
   1465169 DIP
   1465169 SYS
   1465169 SYSTEM
   2283062 MARK_POINT2
   2283062 PWDTEST
   2283062 MARK_POINT
   2283062 FINDME_TOO
   2283062 FINDME
   2283062 SCOTT
   2283062 MGMT_VIEW
   2283062 MDDATA
   2283062 SYSMAN
   2283062 MDSYS
   2283062 SI_INFORMTN_SCHEMA
   2283062 ORDPLUGINS
   2283062 TESTUSER
   2283062 OLAPSYS
   2283062 ORDSYS
```

```
27 rows selected.
```

As we can see, the SCN has changed for all users in the same block so whilst getting the SCN doesn't direct you to the proverbial needle from the proverbial haystack, it does reduce it somewhat.

Getting the server's version and patch level
This query will return the major and minor version of the database server.

```
SQL> SELECT BANNER FROM V$VERSION;
```

However, it will not indicate the server's patch level. Instead, this can be taking a checksum for each PL/SQL package in the database and comparing them with a list of known checksums. This will also to discover any packages that may have been modified.

```
SQL> SET NUMF 99999999999999999999999999999
SQL> DECLARE
      TYPE C_TYPE IS REF CURSOR;
      CV C_TYPE;
      V_ONAME VARCHAR2(30);
      V_OWNER VARCHAR2(30);
      V_OBJID NUMBER;
      V_HASH NUMBER;
BEGIN
      DBMS_OUTPUT.ENABLE(100000);
      OPEN CV FOR 'SELECT U.NAME,O.NAME,O.OBJ# FROM SYS.OBJ$ O,
      SYS.USER$ U WHERE O.OWNER#=U.USER# AND O.TYPE# = 11
      ORDER BY U.NAME';
      LOOP
            FETCH CV INTO V_OWNER,V_ONAME,V_OBJID;
            EXECUTE IMMEDIATE 'SELECT
            SUM(SYS.DBMS_UTILITY.GET_HASH_VALUE
            (SOURCE,1,1073741824)) FROM SYS.SOURCE$ WHERE OBJ# = :1'
            INTO V_HASH USING V_OBJID;

            DBMS_OUTPUT.PUT_LINE(V_OWNER||'.'||V_ONAME||':'||V_HASH);

            EXIT WHEN CV%NOTFOUND;
      END LOOP;
      CLOSE CV;
END;
/
```

This will produce output similar to the following:

```
…
…
SYS.DBMS_PICKLER:235688071
SYS.DBMS_JAVA_TEST:821262485
SYS.UTL_FILE:992849395
SYS.UTL_RAW:407384329
SYS.UTL_TCP:308711346
…
…
```

Here, the number after the owner and package name is the hash. This hash has been created using the sum of every line of source once it has been hashed using the DBMS_UTILITY.GET_HASH_VALUE function. These hashes can then be compared against a list of known hashes. Be aware that, in a compromised system, an attacker could have modified the DBMS_UTILITY package or any of its underlying objects including in-memory C functions.

Server Parameters
It is not enough simply to obtain a copy of the server's start up parameter file. The reason for this is simple – the start up parameter file contains the settings for when the server first starts. At any point after this the "live" settings may have been changed with an "ALTER SYSTEM" statement.

```
SQL> SELECT NAME,VALUE FROM V$PARAMETER
```

Any differences between what the start up file and the "live" parameters should be noted an investigated.

There are a number of hidden server parameters. These can be collected with the following query:

```
SQL> select n.ksppinm as "NAME", v.ksppstvl as "VALUE" from
sys.x$ksppi n, sys.x$ksppcv v where n.inst_id=userenv('Instance') and
v.inst_id=n.inst_id and n.indx=v.indx and substr(n.ksppinm,1,1)='_';
```

Getting all the server parameters, both hidden and "visible", in the same query can be achieved by removing the "`and substr(n.ksppinm,1,1)='_'`" from the query above.

Gathering External Files
The incident responder may also "discover" new locations for dump destinations from the query above and any files present in those locations should be copied to the collection server.

List all data files
As already discussed the names of the data files that make up the SYSTEM, SYSAUX, TEMP and UNDO tablespaces can be obtained from the control files. The query below can confirm if any have been missed:

```
SQL> SELECT T.NAME AS "TABLESPACE", D.NAME AS "FILNAME" FROM
V$DATAFILE D, TS$ T WHERE T.TS#=D.TS#;
```

Any file listed as making up part of the tablespaces mentioned should be copied to the collection server.

List all directories
Directories are used by Oracle for external file accesses. The query below will reveal the file path for such directories.

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "DIRECTORY", D.OS_PATH AS
"PATH" FROM SYS.OBJ$ O, SYS.USER$ U, SYS.DIR$ D WHERE
U.USER#=O.OWNER# AND O.OBJ#=D.OBJ#;
```

Any files in these directories should be copied to the collection server.

List all external tables
External tables also use Oracle directories. The contents from these should also be copied and examined.

```
SQL> SELECT O.NAME, D.DEFAULT_DIR FROM SYS.OBJ$ O, SYS.EXTERNAL_TAB$
D WHERE D.OBJ# = O.OBJ#;
```

The System Monitor (SMON) MON_MOD$ Table
The SMON MON_MOD$ table contains information about what DML activities have occurred on what table.

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "OBJECT", M.OBJ#, M.INSERTS,
M.UPDATES, M.DELETES, M.TIMESTAMP FROM SYS.MON_MODS$ M, SYS.USER$ U,
SYS.OBJ$ O WHERE O.OBJ#=M.OBJ# AND U.USER#=O.OWNER#;
```

This can often help provide pointers to suspicious activity.

Getting information about triggers
Triggers can be used by attackers as a backdoor mechanism or a logic bomb so they should be checked carefully, especially those that occur on database startup or shutdown or when someone logs on or off. The query below will list those triggers that are enabled.

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "ENABLED_TRIGGER_NAME",
DECODE(T.TYPE#, 0, 'BEFORE',2, 'AFTER','NOTSET') AS "WHEN" FROM
SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE O.OBJ#=T.OBJ# AND
O.OWNER# = U.USER# AND ENABLED=1;
```

Changing the "`AND ENABLED=1`" to "`AND ENABLED=0`" will list disabled triggers.

Finding enabled triggers that fire after startup:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "ENABLED_TRIGGER_NAME" FROM
SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE O.OBJ#=T.OBJ# AND
O.OWNER# = U.USER# AND ENABLED=1 AND BITAND(T.SYS_EVTS,1) = 1;
```

Finding enabled triggers that fire before shutdown:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "ENABLED_TRIGGER_NAME" FROM
SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE O.OBJ#=T.OBJ# AND
O.OWNER# = U.USER# AND ENABLED=1 AND BITAND(T.SYS_EVTS,2) = 2;
```

Finding triggers that fire after logon

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "ENABLED_TRIGGER_NAME" FROM
SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE O.OBJ#=T.OBJ# AND
O.OWNER# = U.USER# AND ENABLED=1 AND BITAND(T.SYS_EVTS,8) = 8;
```

Finding triggers that fire before logoff

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "ENABLED_TRIGGER_NAME" FROM
SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE O.OBJ#=T.OBJ# AND
O.OWNER# = U.USER# AND ENABLED=1 AND BITAND(T.SYS_EVTS,16) = 16;
```

Get the source of all triggers

```
SQL> SET LONG 10000000
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "TRIGGER_NAME", T.ACTION# AS
"TEXT" FROM SYS.OBJ$ O, SYS.TRIGGER$ T, SYS.USER$ U WHERE
O.OBJ#=T.OBJ# AND O.OWNER# = U.USER#;
```

You can also get the source with the following query from the SOURCE$ table:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROCEDURE", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=12;
```

Checksumming the trigger text

This anonymous block of PL/SQL will hash the text of every trigger so the hashes can be compared against a list of known good hashes. This way, triggers that have been modified can be quickly identified.

```
DECLARE
      TYPE C_TYPE IS REF CURSOR;
      CV C_TYPE;
      V_ONAME VARCHAR2(30);
      V_OWNER VARCHAR2(30);
      V_OBJID NUMBER:=52296;
      V_HASH NUMBER:=0;
      V_BUFFER LONG(32767);
      CUR NUMBER;
      RES NUMBER;
      POS NUMBER;
      LEN NUMBER;
BEGIN
      DBMS_OUTPUT.ENABLE(1000000);
      OPEN CV FOR 'SELECT U.NAME,O.NAME,O.OBJ# FROM SYS.OBJ$ O,
SYS.USER$ U WHERE U.USER# = O.OWNER# AND O.TYPE# = 12 ORDER BY
U.NAME';
      LOOP
            FETCH CV INTO V_OWNER,V_ONAME,V_OBJID;
            EXIT WHEN CV%NOTFOUND;
            CUR:=DBMS_SQL.OPEN_CURSOR;
            DBMS_SQL.PARSE(CUR,'SELECT T.ACTION# FROM SYS.TRIGGER$ T
            WHERE T.OBJ# = :1',DBMS_SQL.NATIVE);
            DBMS_SQL.BIND_VARIABLE(CUR, ':1', V_OBJID);
            DBMS_SQL.DEFINE_COLUMN_LONG (CUR, 1);
            RES := DBMS_SQL.EXECUTE_AND_FETCH (CUR);
            IF RES > 0 THEN
            POS:=0;
            V_HASH:=0;
            LOOP
            DBMS_SQL.COLUMN_VALUE_LONG(
            CUR,1,32767,POS,V_BUFFER,LEN);
            EXIT WHEN LEN = 0;
            V_HASH:= V_HASH + SYS.DBMS_UTILITY.GET_HASH_VALUE
            (V_BUFFER,1,1073741824);
            POS := POS + LEN;
            END LOOP;
            DBMS_SQL.CLOSE_CURSOR (CUR);
            END IF;
            DBMS_OUTPUT.PUT_LINE(V_OWNER||'.'||V_ONAME||':'||V_HASH);
            V_BUFFER:=NULL;
      END LOOP;
      CLOSE CV;
```

```
END;
/
```

This produces output similar to the following:

```
…
…
SYS.OLAPISTARTUPTRIGGER:443575894
SYS.OLAPISHUTDOWNTRIGGER:473426999
SYS.PREVENT_DDL:283296376
SYS.VANISH:797468158
SYSMAN.MGMT_METRIC_COLL_DEL:731772430
SYSMAN.MGMT_SQL_METRIC_TR:91156933
…
…
```

Again, not that this PL/SQL calls the DBMS_UTILITY package and, this time, the DBMS_SQL package. This may have repercussions as far as trust is concerned.

Getting Information about Views

Views can be used by attackers to hide information. The text of all views should be retrieved. Special attention should be paid to any view beginning with DBA_ particularly DBA_VIEWS, DBA_USERS, DBA_ROLE_PRIVS, DBA_TAB_PRIVS and DBA_JOBS.

```
SQL> SET LONG 100000000
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "VIEW", V.TEXT FROM
SYS.VIEW$ V, SYS.OBJ$ O, SYS.USER$ U WHERE O.OBJ#=V.OBJ# AND
O.OWNER#=U.USER# ORDER BY U.NAME;
```

Checksumming the view text

This anonymous block of PL/SQL will hash the text of every view so the hashes can be compared against a list of known good hashes. This way, views that have been modified can be quickly identified.

```
DECLARE
      TYPE C_TYPE IS REF CURSOR;
      CV C_TYPE;
      V_ONAME VARCHAR2(30);
      V_OWNER VARCHAR2(30);
      V_OBJID NUMBER:=52296;
      V_HASH NUMBER:=0;
      V_BUFFER LONG(32767);
      CUR NUMBER;
      RES NUMBER;
      POS NUMBER;
      LEN NUMBER;
BEGIN
      DBMS_OUTPUT.ENABLE(1000000);
      OPEN CV FOR 'SELECT U.NAME,O.NAME,O.OBJ# FROM SYS.OBJ$ O,
SYS.USER$ U WHERE U.USER# = O.OWNER# AND O.TYPE# = 4 ORDER BY U.NAME,
O.NAME';
      LOOP
            FETCH CV INTO V_OWNER,V_ONAME,V_OBJID;
            EXIT WHEN CV%NOTFOUND;
            CUR:=DBMS_SQL.OPEN_CURSOR;
```

```
            DBMS_SQL.PARSE(CUR,'SELECT V.TEXT FROM SYS.VIEW$ V WHERE
V.OBJ# = :1',DBMS_SQL.NATIVE);
            DBMS_SQL.BIND_VARIABLE(CUR, ':1', V_OBJID);
            DBMS_SQL.DEFINE_COLUMN_LONG (CUR, 1);
            RES := DBMS_SQL.EXECUTE_AND_FETCH (CUR);
            IF RES > 0 THEN
            POS:=0;
            V_HASH:=0;
            LOOP
            DBMS_SQL.COLUMN_VALUE_LONG(
            CUR,1,32767,POS,V_BUFFER,LEN);
            EXIT WHEN LEN = 0;
            V_HASH:= V_HASH + SYS.DBMS_UTILITY.GET_HASH_VALUE
            (V_BUFFER,1,1073741824);
            POS := POS + LEN;
            END LOOP;
            DBMS_SQL.CLOSE_CURSOR (CUR);
            END IF;
            DBMS_OUTPUT.PUT_LINE(V_OWNER||'.'||V_ONAME||':'||V_HASH);
            V_BUFFER:=NULL;
      END LOOP;
      CLOSE CV;
END;
/
```

This produces output similar to

```
…
…
SYS.DBA_UNUSED_COL_TABS:215069642
SYS.DBA_UPDATABLE_COLUMNS:377298913
SYS.DBA_USERS:958803668
SYS.DBA_USTATS:203495787
SYS.DBA_VARRAYS:670521746
SYS.DBA_VIEWS:49730132
…
…
```

Getting information about all libraries

Libraries can be used as a mechanism for running arbitrary code and so they should be examined.

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "LIBRARY", L.FILESPEC AS
"PATH" FROM SYS.LIBRARY$ L, SYS.USER$ U, SYS.OBJ$ O WHERE
O.OBJ#=L.OBJ# AND O.OWNER#=U.USER#;
```

Any DLL which is not default should be copied to the collection server and examined.

Getting information about all database links

Database links are often created by attackers to connect to other Oracle database servers. The CTIME column indicates when the link was created.

```
SQL> SELECT U.NAME AS "OWNER", L.NAME AS "LINK", L.HOST, L.USERID,
L.PASSWORDX, L.CTIME FROM SYS.LINK$ L, SYS.USER$ U WHERE
L.OWNER#=U.USER#;
```

If the owner is listed as “PUBLIC” then the link is a PUBLIC link and can be accessed by anyone. As to who created a link which is PUBLIC, this is more difficult to ascertain and sources such as the redo logs should be used.

Getting information about all synonyms
Synonyms can be used by attackers to influence PL/SQL execution and hide information. A full list of synonyms and their owners should be listed:

```
SQL> SELECT U.NAME AS "OWNER", S.OWNER AS "REAL-OWNER", S.NAME AS
"OBJECT", O.NAME AS "SYNONYM" FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SYN$
S WHERE S.OBJ#=O.OBJ# AND O.OWNER#=U.USER#;
```

Getting information about all database jobs
Database jobs can be used by attackers to take actions at a set time and date – even when they are not logged into the system. Indeed, one the published database rootkits written by Cesar Cerrudo uses a database job.

```
SQL> SELECT JOB, LOWNER, POWNER, COWNER, LAST_DATE, NEXT_DATE, WHAT
FROM SYS.JOB$;
```

Oracle 10g introduced the Job Scheduler and the database should be queried for jobs using this functionality – the job information is stored in a different table. To get a list of jobs execute the following:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "JOBNAME", J.PROGRAM_ACTION
FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SCHEDULER$_JOB J WHERE
J.OBJ#=O.OBJ# AND O.OWNER#=U.USER#;
```

Even if a job has been deleted there may be information about its past in the scheduler event log so this should be queried, too:

```
SQL> SELECT LOG_ID, LOG_DATE, NAME, OWNER, STATUS FROM
SYS.SCHEDULER$_EVENT_LOG ORDER BY LOG_ID;
```

For a list of programs (executable, PL/SQL block or stored procedure) associated with the job scheduler execute the following queries

Executable:
```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROGRAM-EXECUTABLE",
P.ACTION FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SCHEDULER$_PROGRAM P WHERE
O.OBJ#=P.OBJ# AND U.USER#=O.OWNER# AND BITAND(P.FLAGS,32)=32;
```

Stored Procedure:
```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROGRAM-STORED-PROCEDURE",
P.ACTION FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SCHEDULER$_PROGRAM P WHERE
O.OBJ#=P.OBJ# AND U.USER#=O.OWNER# AND BITAND(P.FLAGS,4)=4;
```

PL/SQL Block
```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROGRAM-PLSQL-BLOCK",
P.ACTION FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SCHEDULER$_PROGRAM P WHERE
O.OBJ#=P.OBJ# AND U.USER#=O.OWNER# AND BITAND(P.FLAGS,2)=2;
```

Getting information about PL/SQL objects

The source of PL/SQL objects should be retrieved and analyzed. Much of the source is encrypted or "wrapped" to use the Oracle term. The incident responder should obtain an "unwrapper" to examine the clear text as an attacker can modify a PL/SQL object and re-encrypt it to hide their attack. A commercial "unwrapper" can be acquired from NGSSoftware Ltd.

Procedures:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROCEDURE", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=7;
```

Functions:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PROCEDURE", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=8;
```

Packages:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PACKAGE", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=9;
```

Package bodies:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "PACKAGE-BODY", S.SOURCE
FROM SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=11;
```

Triggers:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "TRIGGER", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=12;
```

Types:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "TYPE", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=13;
```

Type bodies:

```
SQL> SELECT U.NAME AS "OWNER", O.NAME AS "TYPE-BODY", S.SOURCE FROM
SYS.USER$ U, SYS.OBJ$ O, SYS.SOURCE$ S WHERE O.OBJ#=S.OBJ# AND
O.OWNER#=U.USER# AND O.TYPE#=14;
```

Getting information about Java objects

Java "code" can be loaded into the database as a class file (bytecode), or as Java source which is compiled when the source is loaded. The source is loaded into the X$JOXFS table and the bytecode is loaded into the IDL_UB1$ table.

Getting Java source

```
SQL> SELECT U.NAME, O.NAME, S.JOXFTSRC FROM SYS.USER$ U, SYS.OBJ$ O,
X$JOXFS S WHERE O.OBJ# = S.JOXFTOBN AND O.OWNER# = U.USER#;
```

Checksumming the Java bytecode

An attacker can make a modification to the bytecode. This should be checksummed and matched a list of known good checksums.

```
DECLARE
      TYPE C_TYPE IS REF CURSOR;
      CV C_TYPE;
      V_ONAME VARCHAR2(30);
      V_OWNER VARCHAR2(30);
      V_OBJID NUMBER:=52296;
      V_HASH NUMBER:=0;
      V_BUFFER RAW(32767);
      CUR NUMBER;
      RES NUMBER;
      POS NUMBER;
      LEN NUMBER;
BEGIN
      DBMS_OUTPUT.ENABLE(1000000);
      OPEN CV FOR 'SELECT U.NAME,O.NAME,O.OBJ# FROM SYS.OBJ$ O,
SYS.USER$ U WHERE U.USER# = O.OWNER# AND O.TYPE# = 29 ORDER BY
U.NAME';
      LOOP
            FETCH CV INTO V_OWNER,V_ONAME,V_OBJID;
            EXIT WHEN CV%NOTFOUND;
            CUR:=DBMS_SQL.OPEN_CURSOR;
            DBMS_SQL.PARSE(CUR,'SELECT S.PIECE FROM SYS.IDL_UB1$ S
WHERE S.OBJ# = :1',DBMS_SQL.NATIVE);
            DBMS_SQL.BIND_VARIABLE(CUR, ':1', V_OBJID);
            DBMS_SQL.DEFINE_COLUMN_RAW (CUR, 1, V_BUFFER, 32767);
            RES := DBMS_SQL.EXECUTE_AND_FETCH (CUR);
            IF RES > 0 THEN
            POS:=0;
            V_HASH:=0;
            DBMS_SQL.COLUMN_VALUE_RAW(CUR,1,V_BUFFER);
            V_HASH:= V_HASH + SYS.DBMS_UTILITY.GET_HASH_VALUE
            (V_BUFFER,1,1073741824);
            DBMS_SQL.CLOSE_CURSOR (CUR);
            END IF;
            DBMS_OUTPUT.PUT_LINE(V_OWNER||'.'||V_ONAME||':'||V_HASH);
            V_BUFFER:=NULL;
      END LOOP;
      CLOSE CV;
END;
/
```

This produces output similar to the following

```
…
…
SYS./bf8c3870_ConnectionStructHelp:52013415
SYS./f7546018_DatabaseInterface:766489862
SYS./ad32ba4d_DatabaseInterfaceHel:974677841
SYS./f0cd62e8_DatabaseInterfaceStu:202096582
SYS./d100f507_PropertySequenceHelp:222649136
SYS./809cbfa9_PropertyStruct:904769799
SYS./91352016_RemoteAuthentication:905079855
…
…
```

We can also use this query to hash the code of other PL/SQL objects that also can be found in the IDL_UB1$ table.

Finishing Up
Once all queries have been executed the spool file should be closed and sqlplus can be closed.

```
SQL> SPOOL OFF
SQL> QUIT
Disconnected from Oracle Database 10g Enterprise Edition Release
10.2.0.2.0 – Production With the Partitioning, OLAP and Data Mining
options

C:\oracle\product\10.2.0\db_1\BIN>
```

Once disconnected from the server an md5 checksum should be made of the spool file and recorded with a witness present.

Wrapping Up
With all the Live Response data collected and stored safely on the collection server one of several actions could be taken. Firstly the system being investigated can be shutdown and taken off the network then processed into evidence – assuming of course the organization intends to take further (legal) action against the individual(s) involved. This may not be viable however as a replacement system may not be available and to keep the business online and ticking over the system may need to be left in place. Regardless, as quickly as possible, the incident responder needs to work out as quickly as possible how the attacker managed to gain entry in the first instance and provide recommendations on how to prevent further incursions.