Understanding Oracle Auditing

GIAC Security Essentials
Certification (GSEC)
Practical Assignment
Version 1.4b

Option 1 - Research on Topics
in Information Security

Submitted by: Wayne Reeser
Sep 21, 2004
Location: SANS Baltimore, 2004

Table of Contents

Abstract

Understanding Oracle Auditing is critical for comprehensive application security, but it is perceived as difficult and complex. While the typical DBA can probably configure and enable auditing, especially given “recommended” auditing options by the latest DB security scanner, it is unlikely that the auditing design will be as efficient and effective as it could be. This paper will address basic Oracle auditing and will explain some of the common “features” of audit which can confuse or mystify even experienced DBAs. A strong grasp of the basics will provide a good foundation for later forays into advanced auditing and understanding of the results generated by enabling the auditing options required by Oracle security guides.

Introduction

The first auditing question asked of DBAs is usually “How much will auditing hurt performance?” This is an incomplete question that can’t be answered effectively without a lot more information. Appropriate information includes the requirements for auditing and consideration of the role auditing will play in the overall security architecture. An effective audit strategy aims to collect the minimum amount that is necessary to meet requirements, and may be dynamic in that certain incidents trigger increased levels of auditing. Maximum value auditing is thus achieved with minimum impact. This can best be accomplished when auditing is approached as a system design problem and consideration is given to the best audit methods.

Standard Oracle auditing is complex, but with the right foundation, it is possible to approach it with confidence. Once standard audit is understood, the best usage of the other auditing mechanisms will fall into place. The key is to learn enough about the various audit tools to be able to pick the right tool for the job, and to minimize the data collected. In the space available, this paper will attempt to explain standard audit, provide some useful scripts for an audit toolkit, and provide a roadmap to exploring the other auditing mechanisms.

All examples have been worked in Oracle version 9.2. While most of the examples will work in earlier as well as later versions, results may vary.

Requirement for Auditing

Auditing serves as a deterrent to misuse, a tool for detection and damage assessment after an incident, and an option for accepting risk if a cost effective safeguard is not available. Kewley and Lowry (2001) documented a DARPA study to determine if additional layers of security always resulted in greater overall security. They found that depending upon the objective of the attacker, additional layers sometimes added no additional security, and often made it easier to complete the attack. The more complex the system, the more likely that vulnerabilities will exist. Despite Defense in Depth, most internet connected systems will experience security incidents, and effective auditing is an important tool to assist in detecting and in evaluating those incidents.

The five goals of an audit system, as paraphrased from NCSC (1987, para. 5.1) are:

1. Allow review of patterns of access to objects by individuals and the effectiveness of their protection mechanisms
2. Allow discovery of repeated attempts to bypass the protection mechanisms
3. Allow discovery of use of elevated privileges, even when legitimate.
4. Act as a deterrent – a perpetrator should be aware of the audit existence and usage.
5. Provide assurance that attempts to bypass protection are recorded and discovered, even if the attempt is successful.

These goals provide a useful framework for evaluating an auditing design. It is also important to have the legal department review auditing plans. Auditing is a form of monitoring. Monitoring must be done appropriately and in compliance with corporate policy both in order to avoid legal liability and to enable the use of the audit data in legal proceedings. A complete plan will derive from corporate policy and will include direction for allowable auditing, access to audit data, storage and handling of audit data, and archiving and destruction.

Where to start?

It is critical to obtain the Oracle documentation as a reference when approaching auditing. Oracle provides documentation for supported versions on Oracle Technet. The Oracle Database Security Guide describes how to enable standard auditing, how to determine the current auditing in effect, and how to examine the audit trail. The Oracle Database SQL Reference describes the AUDIT and NOAUDIT syntax and options.

Although necessary as a reference, the Oracle documentation can be overwhelming as an introduction. Finnigan's "Introduction to Simple Oracle Auditing" introduces several of the key auditing features and provides worked examples that show how to use auditing to detect certain abuses. This is a good overview, but it fails to address some common auditing quirks that can be quite confusing.

Another difficulty to approaching auditing is that there are many ways to collect data. According to Burleson, there are five ways to audit within Oracle:

- SQL audit command (for DML)
- Auditing with object triggers (i.e., DML auditing)
- Auditing with system-level triggers (i.e., DML and DDL)
- Auditing with LogMiner (i.e., DML and DDL)
- Fine-grained auditing (i.e., select auditing)

In addition, Finnigan points out a sixth, Oracle system logs, which may be required to fully analyze an incident. Beyond that, Flashback Query is an excellent tool to let an investigator go back and see what data might have been exposed by the audited event, exactly as it appeared at the time of the event. Each of these methods has its own interface, audit trail location(s), and effective usage, and it is necessary to know a bit about all of them to be most effective.

This paper will concentrate on standard auditing in order to explain some of the quirks which make it appear more complex and confusing to the uninitiated. The audit interface can be tricky at best and is easily the least intuitive of all of the auditing methods.

Where is the audit trail?

Standard audit sends output to two locations, DB and OS. Standard audit records are written only if two conditions are true: First, the initialization parameter AUDIT_TRAIL must be set to something besides “OFF”. Second, audit records will only be produced for audit events currently enabled via the SQL command AUDIT. Regardless of the AUDIT_TRAIL setting, no actions when connected “AS SYSDBA” are audited by the AUDIT command.

AUDIT_TRAIL determines the destination of the audit trail data, and can be set to “DB”, “OS”, or “OFF”. AUDIT_TRAIL is a static parameter, requiring a DB restart to change its value. “OFF” disables standard audit, but does not change the audit settings established by the AUDIT command.

DB audit trail

The DB audit trail table is normally found in SYS.AUD$. DB audit records are sparse, in that many of the fields are not populated depending upon the audit type. To make things a bit easier, Oracle provides numerous views based on AUD$ which limit the type of records and columns displayed for particular interests.

DBA_AUDIT_TRAIL (Figure A- 1) is the most comprehensive view and it includes all audit records plus provides code lookups to make the AUD$ data presentable. Other views are more specialized, but an experienced DBA will develop customized queries against DBA_AUDIT_TRAIL instead.

OS audit trail

The OS audit trail location varies by platform. On Solaris, individual files containing one or more audit records are, by default, put in

$ORACLE_HOME/rdbms/audit. The optional initialization parameter AUDIT_FILE_DEST can override the default location on some platforms. On Windows, the OS trail goes into the event log which can be accessed by event viewer or dumped to a flat file with dumpel.exe, a free resource kit component from Microsoft.

How to choose between audit trail locations

The decision on which audit trail to use is driven by security and ease of access. The DB audit trail is the most common choice since it contains more information and is easier to access and analyze. The OS audit trail is more difficult to access and modify from inside the database, which makes it easier to protect from malicious database users or DBAs. However, this increased protection is offset by a difficult to parse file format and decreased audit information. In either case, certain audit records always go to the OS audit trail and these should be reviewed regularly.

Mandatory OS auditing

By design, database startup, shutdown, and connection attempts by SYSDBA are always audited to the OS audit trail regardless of the AUDIT_TRAIL setting. This auditing cannot be disabled.

A first audit

The following exercises should be performed on a non-production database, preferably with no other users active. Obtain permission to modify any database auditing which is present. When experimenting with the audit trail, it is much easier to figure out why a certain audit record was generated in response to an action if there are no other users potentially producing audit records.

- Set the database initialization parameter AUDIT_TRAIL=DB and restart the database to enable auditing.
- Place the scripts from Appendix B in the SQL*PLUS working directory.
- The script in Figure B- 1 creates a role and a user with the privileges necessary to perform the remainder of the exercises. Execute the script AS SYSDBA.
- Connect as the newly created user "AUD". Unless otherwise stated, do all exercises from the AUD user. Remember, "AS SYSDBA" connections do not produce standard audit records.

First, determine if auditing options are already present. Figure B- 2 is a script called audopts.sql which will show all standard auditing. While this information can be determined via multiple data dictionary views, this one-stop script is easier. This is the expected result if no auditing is active:

```
SQL> @audopts.sql
dba_stmt_audit_opts union dba_priv_audit_opts

no rows selected

dba_obj_audit_opts

no rows selected
```

If auditing options are present, drop the auditing by using the script in Figure B- 3. Execute the script noaudits.sql and it will produce a file called noaudit.sql and automatically execute it. Now try audopts.sql again. The script has some limitations noted in the script documentation, review it if any auditing remains on.

Now clear the audit trail by deleting from AUD$. If this is not allowable, the script in Figure B- 4 is useful. Edit the script audtr.sql to add the current system time to the embedded query. Replace this string before starting an audit test and this will help to return only the relevent audit trail records.

One of the most important goals of auditing is to find out who is accessing the system. AUDIT CREATE SESSION is an efficient way to record this and it should be the first option considered when enabling auditing. From the AUD account, issue the following statements:

```
SQL> audit create session;

Audit succeeded.

SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

USER_NAME  PROXY AUDIT_OPTION                   AUD
---------- ----- ------------------------------ ----
                 CREATE SESSION                 A/A
```

Notice the output of the audopts script shown above. The “A/A” is a two position code. The first letter is for auditing “whenever successful” and the second position is for auditing “whenever unsuccessful”. The codes are A= “by access”, S= ”by session”, and “-“ = “not set”. These options will be discussed below. The proxy column is used for auditing “proxy authentication.” Now reconnect and check the audit trail:

```
SQL> connect aud/aud@ORCA;
Connected.
SQL> select * from dba_audit_trail;

-- see Figure A- 2
```

Examine the audit trail (see Figure A- 2). First notice that many fields are null. The type of audit record determines which fields are populated. Use DBA_AUDIT_TRAIL rather than selecting limited columns whenever experimenting to make sure that all information is available. The three ID fields are critical. The SESSIONID ties together all audit records for a session. ENTRYID is a unique key for each record. STATEMENTID indicates the SQL statement which caused the audit action, and multiple records can be generated for a single statement. Note that ACTION_NAME=LOGON, RETURN_CODE=0, all LOGOFF fields are NULL, COMMENT_TEXT field has authentication information, and PRIVILEGE_USED = CREATE SESSION. This type of record will be produced for each non-sysdba login under "audit create session." Now reconnect again and take another look:

```
SQL> connect aud/aud@ORCA;
Connected.
SQL> select * from dba_audit_trail;

-- see Figure A- 3
```

Most audit trails would have three records: the original LOGON, a LOGOFF, and the new LOGON. In fact, Figure A- 3 shows two records, one with ACTION_NAME=LOGOFF and one with LOGON. What happened to the original LOGON record? That is Oracle audit quirk #1. We'll cover the audit command a bit more and then get back to this and other mysteries below.

The Basics (SQL AUDIT command)

ORACLE supports three different kinds of audits enabled via various syntax of the SQL command AUDIT: statement, privilege, and object. Statement and privilege audits share syntax and can be limited by user. Object audits apply to a single object but cannot be restricted by user. All three share certain common options.

Audit options relevant to all auditing

All Oracle auditing produced by the AUDIT statement goes to the same audit trail. However, the following options are applicable to all forms of the AUDIT statement and modify the details of when auditing is recorded.

AUDIT option BY SESSION

Auditing BY SESSION produces a single audit trail record per audit option regardless of the number of successful or unsuccessful attempts within that session. There is no field which provides an occurrence count or an error code for failed attempts (RETURNCODE is always 0). In most cases, auditing BY ACCESS is used instead of BY SESSION for the increased information.

Only certain options may be enabled BY SESSION. If the option is not a BY SESSION option, the audit will be enabled BY ACCESS and no error will be returned. Whenever the audit command is used, it is wise to follow up with a check to see what resulted. A check of the audit options enabled (use the audopts.sql script) will show which AUDIT method is actually being performed.

AUDIT option BY ACCESS

Auditing BY ACCESS generates an audit trail record for every user attempt. An ACCESS record with a nonzero DBA_AUDIT_TRAIL.RETURNCODE indicates a failed attempt. The RETURNCODE is simply the Oracle Error code returned due to the failure. The benefit of BY ACCESS is that the audit trail shows the number of times the audited action was attempted, the sequence of audited actions, and the result (either success or the failure code) of each action.

AUDIT WHENEVER SUCCESSFUL or NOT SUCCESSFUL

All AUDIT types can be restricted to audit only when a user action succeeds, only when it fails, or both. Note that the syntax is "WHENEVER [NOT] SUCCESSFUL". If the clause is removed entirely, the audit statement will enable audit independent upon the outcome. This can be a useful tool in limiting the "noise" in the audit trail. If users are allowed to perform certain actions, and there is no security relevance to those actions, there is no need to audit. Instead, audit whenever NOT SUCCESSFUL. Not only will this catch attempts to escalate privileges and "fishing trips" by malicious users, it will also catch failed SQL indicating application coding errors or users in need of training.

Statement and Privilege Auditing

Statement and privilege auditing are separated in the Oracle documentation and data dictionary views. However, they use identical syntax, and considering them as identical will simplify things greatly. By definition, a statement audit fires when a user issues the matching SQL statement. A privilege audit fires when the SQL statement requires that privilege in order to succeed. As an example, consider if SCOTT issues the following two statements:

```
DROP TABLE SCOTT.MYTAB;
DROP TABLE HR_APP.PAYROLL;
```

Both of these are drop table statements, but the second one would need the privilege “DROP ANY TABLE” to succeed. If the requirement is to detect drop actions by users against objects which they do not own, auditing the *statement* would produce many audit records of no consequence. The privilege audit “AUDIT DROP ANY TABLE BY ACCESS”, however, fires only on the privilege use, which will detect successful drops.

Continuing with the statement vs. privilege discussion: Issue the DROP ANY TABLE audit and directly check the views which indicate whether a privilege or statement option is set:

```
SQL> audit drop any table;

SQL> select * from DBA_PRIV_AUDIT_OPTS;

USER_NAME  PROXY PRIVILEGE              SUCCESS    FAILURE
---------- ----- ---------------------- ---------- ----------
                 CREATE SESSION         BY ACCESS  BY ACCESS
                 DROP ANY TABLE         BY ACCESS  BY ACCESS

SQL> select * from DBA_STMT_AUDIT_OPTS;

USER_NAME  PROXY AUDIT_OPTION           SUCCESS    FAILURE
---------- ----- ---------------------- ---------- ----------
                 CREATE SESSION         BY ACCESS  BY ACCESS
                 DROP ANY TABLE         BY ACCESS  BY ACCESS
```

First note that the AUDIT syntax does not make a distinction between statement and privilege options. Is "CREATE SESSION" a statement or a privilege option? What about "DROP ANY TABLE?" According to the Oracle views, each appears to be both, but only one audit trail record results from each audited action. To get rid of the audit from both views, simply issue a single "NOAUDIT DROP ANY TABLE". The best approach is to use the audopts.sql script to view the auditing options, and not worry about it.

```
SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

USER_NAME  PROXY AUDIT_OPTION                        AUD
---------- ----- ----------------------------------- ----
                 CREATE SESSION                      A/A
                 DROP ANY TABLE                      A/A
```

To use the AUDIT statement to set statement and privilege options, you must have the AUDIT SYSTEM privilege.

Object Auditing

Object auditing allows the access or usage of specific objects to be audited. Unlike statement/privilege auditing which can be limited to audit only specific users, object auditing is active for all users, but it is limited to one object. The AUDIT ANY privilege is required to be able to set an object audit in general. However, the object owner can enable or disable auditing on owned objects, as well as see which audit options are currently enabled for the object.

Here is an object auditing example:

```
SQL> create table mytab (x number);
SQL> create table mytab2(x number);

SQL> audit select,insert,update on mytab;
SQL> audit select on mytab2 by access;

SQL> @audopts

dba_obj_audit_opts

object
name       ALT AUD COM DEL GRA IND INS LOC REN SEL UPD EXE REA
---------- --- --- --- --- --- --- --- --- --- --- --- --- ---
MYTAB      -/- -/- -/- -/- -/- -/- S/S -/- -/- S/S S/S -/- -/-
MYTAB2     -/- -/- -/- -/- -/- -/- -/- -/- -/- A/A -/- -/- -/-
```

Note that the default auditing option for statement/privilege audits is BY ACCESS, but the default for object auditing is BY SESSION. The best approach is to always specify the desired option and not keep track of the defaults. In addition, always check the options which are enabled after attempting to set them. In this case, audopts.sql shows that the two audit statements produced different results. The comments in the audopts.sql script (see Figure B- 2) specify what each of the character codes mean, but in this example the "S/S" under SEL means auditing of SELECT is active BY SESSION for both success and failure for table MYTAB (since BY ACCESS was not specified and BY

SESSION is the default). For MYTAB2 the “A/A” means BY ACCESS. Now execute some statements which will be audited.

```
SQL> select x from mytab;
SQL> select y_fail from mytab;  -- this fails, bad column name
SQL> select x from mytab2;
SQL> insert into mytab values (1);
SQL> select x from mytab2;
SQL> select x from mytab;


SQL>
select obj_name,action_name,returncode,ses_actions
from dba_audit_trail
where obj_name like 'MYTAB%'
order by timestamp;

OBJ_NAME      ACTION_NAME     RETURNCODE SES_ACTIONS
------------- --------------- ---------- ---------------
MYTAB         SESSION REC              0 ------S--B------
MYTAB2        SELECT                   0
MYTAB2        SELECT                   0
```

The difference in the results of the two audits can be seen in the audit trail. In the case of MYTAB2, the two actions resulted in two BY ACCESS audit trail records. However, in the case of MYTAB, *a single BY SESSION record was created*. The single record is UPDATED whenever an auditable event new to that session occurs. The SES_ACTIONS field is positionally coded as explained in Figure A- 1. In this case, position 7 is “insert” and position 10 is “select.” ‘S’ means success, ‘F’ means failure, and ‘B’ means that both occurred during the session.

Note that if the OS audit trail is in use, BY SESSION records are not updateable on most platforms, so multiple records may be written.

Now continue the previous example:

```
Connect scott/tiger@orca
SQL> insert into aud.mytab values (1);
ORA-00942: table or view does not exist
SQL> select * from aud.mytab2;
ORA-00942: table or view does not exist

connect aud/aud@orca
select * from dba_audit_trail where username='SCOTT';

USERNA OWNER  OBJ_NAME ACTION_NAME RETURNCODE SES_ACTIONS
------ ------ -------- ----------- ---------- --------------
SCOTT  AUD    MYTAB    SESSION REC          0 ------F-------
SCOTT  AUD    MYTAB2   SELECT            2004
```

This example illustrates two points. First, note that the returncode that SCOTT saw in SQL*PLUS (ORA 942) is different from the audit trail returncode in both cases. The session record has returncode 0, even though the ses_action shows a FAIL event. This is expected for session records; they do not record the failure code or the number of occurrences. The ACCESS record shows returncode 2004, which stands for “security violation.” Scott did not have the privilege to select from AUD.MYTAB2, so Oracle indicates to SCOTT that the object does not exist while the audit trail records the real reason. This is a feature designed to limit the information available to a malicious user.

Default Auditing

Oracle object auditing supports a default option in the syntax. It is possible to issue "AUDIT INSERT ON DEFAULT BY ACCESS." Note that this *does not enable any auditing*. Default auditing has no effect on existing objects, rather it creates an audit whenever a new object is subsequently created. For example:

```
SQL> @noaudits
SQL> audit alter on default by access;
SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

no rows selected

dba_obj_audit_opts

           object     object
OWNER      name       type    ALT AUD COM DEL
---------- ---------- ------  --- --- --- ---
DEFAULT    DEFAULT    DEFAULT A/A -/- -/- -/-

SQL> create table testtab (x number);

SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

no rows selected

dba_obj_audit_opts

           object     object
OWNER      name       type    ALT AUD COM DEL
---------- ---------- ------  --- --- --- ---
DEFAULT    DEFAULT    DEFAULT A/A -/- -/- -/-
AUD        TESTTAB    TABLE   A/A -/- -/- -/-
```

In this example, default auditing was enabled, then a table was created. As shown, the default audit option was automatically applied to the table. Default auditing is not particularly useful, but it is important to be aware that it represents auditing on future objects and does not enable those default settings on existing objects.

Tips and Lessons Learned

Common mistakes when testing

Here are some common mistakes made when testing auditing.

- SYSDBA sessions are *never audited by standard audit*. The 9i AUDIT_SYS_OPERATIONS initialization parameter explained below in "Auditing SYSDBA" does not enable standard auditing on SYSDBA sessions, instead it copies all SYSDBA SQL to the OS audit trail. Login as a normal user, not as sysdba, when testing standard audit.

- Auditing options take effect upon next login. Active sessions will not reflect changes in auditing options. If running two sessions, issuing AUDIT commands from one and testing SQL from another, the second session will not reflect the AUDIT changes. Reconnect the second session in order to pick up the current auditing.
- Be meticulous. Certain audit options interfere and will generate different audit trail results. If experimenting, only enable the desired options. Keep scripts and results to test repeatability. Always test auditing in the "production" configuration to ensure that the selected security relevant activities produce the expected entries in the audit trail.
- Be sure to enable the audit option desired. "AUDIT TABLE" is quite different than "AUDIT CREATE TABLE".
- Always check the auditing that was actually enabled by the AUDIT command. Certain AUDIT commands do not produce the expected results and will complete successfully.

Auditing Create Session

AUDIT CREATE SESSION is one of the primary sources of confusion for new auditors. This audits connections and connection attempts to the database. Records produced by this audit option are unusual in that they are *updated after the initial write to the audit trail*. The initial record is written with an action of LOGON, and when the user disconnects the record is updated and the action is changed. This can cause confusion for new Oracle auditors, since they will only find a scattering of “LOGON” records, but many “LOGOFF” records in the audit trail. If AUDIT CREATE SESSION is enabled, session statistics are accumulated for each successful connection and written into the LOGOFF_% fields when the session ends. The ACTION_NAME is changed from “LOGON” to “LOGOFF” for a clean exit, or to “LOGOFF BY CLEANUP” if the session terminated without logging off. Records with ACTION_NAME=LOGON are either unsuccessful login attempts (check the RETURNCODE to see why) or current sessions (RETURNCODE is null). The database does not clean up LOGON records during instance recovery, so LOGON records indicating successful logon with no matching session are an indication of that.

In the following example, the first record has a nonzero returncode, in this case a logon attempt with the wrong password. The second record resulted from a connect and logoff. The “LOGOFF BY CLEANUP” record was produced by connecting as AUD and then killing the SQL*PLUS client process instead of logging out. The last “LOGON” record is the current session and was used to execute the query

```
select username, to_char(timestamp,'MMDD HH24:MI:SS') ts
, action_name, returncode
, to_char(logoff_time,'MMDD HH24:MI:SS') Logoff
from dba_audit_trail order by timestamp;
```

```
USERNA TS         ACTION_NAME          RETURNCODE LOGOFF
------ ---------- -------------------- ---------- ----------
AUD    0829 22:29 LOGON                      1017
AUD    0829 22:36 LOGOFF                        0 0829 22:39
AUD    0829 22:39 LOGOFF BY CLEANUP             0 0829 22:40
AUD    0829 22:41 LOGON                         0
```

Note that the time elapsed for the session is available by subtracting the logoff time from the login time. Calculate the elapsed time in days and then adjust to the desired units.

```
select (LOGOFF_TIME-TIMESTAMP)*(60*24) "session time in minutes"
from dba_audit_trail
where action_name like 'LOGOFF%'
and logoff_time is not null;
```

The DBA_AUDIT_TRAIL.COMMENT_TEXT field is critical in analyzing CREATE SESSION audits since it contains authentication information about the connection method. Although IP addresses and client machine names can be spoofed, the information is still useful in most cases.

Failed SYSDBA connections always go to the OS audit trail. However, the format is not obvious. Test various invalid types of login attempts and learn the audit signature on each platform in use. Try to connect with the following and see what happens:

- Nonexistent user "AS SYSDBA"
- Valid user without SYSDBA, using valid password "AS SYSDBA"
- Valid user without SYSDBA, bad password "AS SYSDBA"
- Valid SYSDBA user, bad password "AS SYSDBA"
- Valid SYSDBA user, valid password "AS SYSDBA"

The following is a Windows event log produced from a failed “connect aud@DBSID as sysdba,” followed by a successful connect. Note that the record does not directly indicate that this was an attempt at sysdba (it is inferred since this is the OS audit log and the PRIVILEGE is NONE) and the status 1031 is “insufficient privileges”. AUD is a valid account but does not have the SYSDBA privilege. Another status code, 1017, is “bad username/password.” This one appears if the user account does not exist at all.

```
Audit trail: ACTION : 'CONNECT' DATABASE USER: 'aud' PRIVILEGE :
NONE CLIENT USER: xxxx\wreeser CLIENT TERMINAL: xxxx STATUS: 1031.

Audit trail: ACTION : 'CONNECT' DATABASE USER: 'sys' PRIVILEGE :
SYSDBA CLIENT USER: SYSTEM CLIENT TERMINAL: xxxx STATUS: 0 .
```

If testing from an OS account with the OS DBA role enabled, be aware that the password is not checked during SYSDBA logins. In order to test completely, use a non-DBA OS account.

Auditing SYSDBA

Full SYSDBA auditing was not possible prior to version 9, when the AUDIT_SYS_OPERATIONS initialization parameter was added. This parameter does not make SYSDBA subject to standard auditing. Instead, the SQL text of every statement goes in the OS audit trail ACTION field.

Even if the DBA is trusted, it is critical to review failed connections to the SYSDBA account, as this is the only way to detect a password cracking attempt against SYSDBA. SYSDBA connection attempts always go to the OS audit log regardless of whether SYSDBA auditing is enabled, but these records are often ignored or deleted to conserve space. SYSDBA auditing provides a valuable way to track authorized or unauthorized SYSDBA actions, but it is only useful if the OS audit trail is reviewed.

The OS audit trail files consist of name/value pairs and can be parsed. Consider forwarding audit files to a separate system as they are created and process them there to reduce the danger of modification or deletion. Be aware, that certain audit files are kept open for extended periods. Consult a platform system administrator for the best way to securely transfer and parse the files so as not to lose information. Although full automated analysis or reduction of the OS audit trail is difficult, full analysis of connection attempts is not, and provides the biggest benefit.

When analyzing the SYSDBA audit trail, note that the SQL text of each statement is written into the audit file, complete with carriage returns in some cases. Also, there is a limited length per audit record on most platforms. If the audit statement exceeds this length, multiple audit events are generated, each with a chunk of the SQL text until the SQL text is exhausted. There is no "chunk id" in the audit trail, so reassembling these chunks can be difficult.

Protecting the audit trail

It is important to protect the audit trail from modification. To audit changes made to the database audit trail (the AUD$ table), use the following statement:

```
AUDIT AUDIT, INSERT, UPDATE, DELETE ON AUD$ BY ACCESS;
```

Audit records in the AUD$ table can only be deleted by a SYSDBA or an account with delete on AUD$. In general, the SYSDBA account should be restricted to a highly trusted DBA and all other DBAs should be operating under least privilege. Write a package to automate the process of purging the audit trail so that the direct privilege is not needed by the auditors (see “Managing the audit trail for performance” below).

If connected as SYSDBA, there is no need for a malicious user to modify the DB audit trail to cover his actions since any audits would go to the OS audit trail. Because the OS audit trail writes to a small number of separate files per session, it is trivial for a DBA with OS privileges to find and delete the relevant files. Make sure that all DBAs have personal OS accounts and that they use the OS group to obtain OS privileges on Oracle. This allows file permissions to be set so that most DBAs cannot delete the OS audit files. If this is done, consider how to deal with the situation of a filled audit OS directory. This could be accidental, or a deliberate DOS attack and is as simple as launching an infinite loop. Ensure that procedures exist to allow audit trail maintenance by all DBAs in such a situation, along with oversight to ensure that this condition is not generated to hide malicious actions.

Even if the audit trail integrity is suspect, some actions can be detected after the fact by good configuration management. One way to detect object changes is to query the DBA_OBJECTS table where last_ddl_time is recent. A more accurate approach would be to run a periodic comparison of the schema to the expected results, similar to the way in which Tripwire is used to check the integrity of the operating system files.

It is as important to monitor the auditing options which are in effect as it is to guard against changes to the audit trail. Auditing the AUDIT statement itself (AUDIT AUDIT SYSTEM and AUDIT AUDIT ANY) will record any changes to auditing options in the DB audit trail. In addition, “AUDIT AUDIT on app_object BY ACCESS” should be issued for each object in an application schema. Here is a quick example:

```
SQL> rem  CLEAN UP AUDIT TRAIL FIRST
SQL> @NOAUDITS.SQL
SQL> @DELETE FROM AUD$

-- reconnect to clear any options from the session
SQL  CONNECT AUD/AUD@sid

SQL> AUDIT AUDIT, INSERT, UPDATE, DELETE ON AUD$ BY ACCESS;
SQL> AUDIT AUDIT SYSTEM;
SQL> AUDIT AUDIT ANY;
SQL> AUDIT AUDIT ON MYTAB BY ACCESS;
SQL> AUDIT AUDIT ON MYTAB2;
SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

USER_NAME  PROXY AUDIT_OPTION                     AUD
---------- ----- -------------------------------- ----
                 AUDIT ANY                        A/A
                 AUDIT SYSTEM                     A/A
                 SYSTEM AUDIT                     A/A

dba_obj_audit_opts

object
name    ALT AUD COM DEL GRA IND INS LOC REN SEL UPD EXE REA
------- --- --- --- --- --- --- --- --- --- --- --- --- ---
AUD$    -/- A/A -/- A/A -/- -/- A/A -/- -/- -/- A/A -/- -/-
MYTAB   -/- A/A -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
MYTAB2  -/- S/S -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
```

Note that the statement audit options in effect show “audit system” and “system audit.” This is a side effect of the privilege/statement issue mentioned previously. It represents a single audit, and is removed by “noaudit audit system”.

Note that to include audit modifications on objects, you must explicitly use "BY ACCESS." As can be seen in the MYTAB2 example above, for objects BY SESSION is the default. Now remove the auditing and check the audit trail.

```
select obj_name,action_name,returncode,ses_actions
from dba_audit_trail
where obj_name like 'MYTAB%'
order by timestamp;


OBJ_NAME   ACTION_NAME     RETURNCODE SES_ACTIONS
---------- --------------- ---------- ----------------
MYTAB2     SESSION REC              0 -S--------------
MYTAB      AUDIT OBJECT             0

SQL> noaudit all on mytab;
SQL> noaudit all on mytab2;
SQL>
select obj_name,action_name,returncode,ses_actions
from dba_audit_trail
where obj_name like 'MYTAB%'
order by timestamp;

OBJ_NAME   ACTION_NAME     RETURNCODE SES_ACTIONS
---------- --------------- ---------- ----------------
MYTAB2     SESSION REC              0 -S--------------
MYTAB      AUDIT OBJECT             0
MYTAB      NOAUDIT OBJECT           0
```

In this case, the session record does not provide enough information to see that the auditing has been removed. However, BY the ACCESS records show the noaudit.

Just as with object configuration management, audit option configuration management is a good addition to defense. Once the desired auditing options are set, the audopts.sql script output can be baselined and an automatic check can be scheduled to detect changes against the baselined output. The combination of the audit trail and the configuration check should minimize errors and maximize the detection of audit changes.

Managing the audit trail for performance

The audit trail tables AUD$ and FGA_LOG$ are located in the SYSTEM tablespace and most DBA modifications to the table are not permissible under Oracle Support. In order to efficiently perform analysis on the audit trail, indexes are necessary. Since indexes cannot be added to AUD$ for support and performance reasons (they would slow down inserts into the audit trail), a solution is needed. Here is one solution that may be of use:

- Create an auditor schema which will remained locked once built.
- Duplicate the AUD$ and FGA$ tables there (create table as select). Index them as needed if you intend to do analysis. Partition the tables by timestamp if the audit volume is large.
- Create an audit utilities package under the auditor schema. This will contain all of the needed procedures for managing the audit data.

- Schedule a job to move the audit data out of the system tablespace and into the auditor schema tables. Run the job frequently enough that the number of records to be moved is small.

  The “move” job which transfers data from AUD$ to the audit schema AUD$ is critical to get right. It must only move data which does not belong to current sessions. Remember that LOGON and SESSION records are updated. If the records have been moved, the update information will be lost, which could be a big problem in some industries. Compare the audit trail sessionid to the V$SESSION information to determine if the record can be safely moved. Choose whether to copy (and later duplicate or overwrite) updateable records which belong to active sessions or simply wait until the session disconnects. The typical scenario is to perform an "insert as select" with some where clause to move the data into the audit repository, followed by a delete. Remember to include a max timestamp in both where clauses or use a single transaction so that the delete statement does not remove new records which were not copied.

  For connection pooled architectures, deciding whether to wait for a session to end will have significant consequences in terms of early detection of malicious activity, since sessions may stay active for a long time. If a connection pool is involved, consider cycling the connections (have the application server reconnect a pool connection after a certain number of uses) or restarting the connection pool on some schedule.
- If the tables are partitioned, deal with them as they fill. Move them to a separate auditing DB, or analyze and delete them. By maintaining them in partitions, it is easy to meet such requirements as “maintain three months of audit data online” while still being able to roll off data without a massive delete operation. Empty partitions take little space, allowing a full year "rolling window" of audit partitions to be kept online if you partition by week number.
- Assign auditors execute access on the audit utilities package to manage the data transfers. This will allow them to perform the duties without direct permissions on the audit trail tables, and their actions can in turn be audited.
- If system performance and security concerns permit, index the audit schema table and allow auditors to work from it. If not, transfer the data to an audit DB and perform analysis there.

The benefit of this approach is that the SYSTEM audit tables are kept small. To the extent possible, the audit work is taken offline with as little impact on the production system as possible while still maintaining the audit trail integrity.

Object Auditing tips

In general, the schema which owns an object has the right to enable or disable auditing on that object. For this reason, it is a bad idea to allow users or administrators to connect to the schema of application tables. In fact, the schemas should be locked if at all possible, and all privileges explicitly granted to application DBA accounts. If locking the account is not possible, it is still possible to protect the auditing on those objects. First, AUDIT AUDIT ON OBJECT_NAME BY ACCESS. This will alert that someone attempted to modify the auditing. Second, disable AUDIT and NOAUDIT by using database triggers on the DDL. For example, this trigger will prevent NOAUDIT:

```
create or replace trigger no_noaudit
before noaudit on database
begin
  raise_application_error(-20000,'Noaudit command is disabled');
end;
```

Once this trigger is enabled, no auditing can be disabled unless the user has the privileges to drop the trigger, which would normally be limited to SYSDBA.

What if the desired audit option does not exist?

Certain audit statements result in the audit of related statements. For example, "AUDIT DROP TABLE" does not exist. Instead, AUDIT TABLE will audit create, drop, and truncate. The AUDIT section of the Oracle Database SQL Reference contains tables which provide the available syntax and the resulting auditing. These tables are fairly accurate, but some exceptions exist. Always verify that the desired auditing is in place by attempting the action and verifying that an audit record is produced.

AUDIT NOT EXISTS

AUDIT NOT EXISTS catches failed attempts to access existing or nonexisting objects. This is a useful auditing option because it will detect actions that indicate either malicious activity or broken application code.

Issues when modifying existing auditing

When manipulating existing auditing, especially to change the success/not success or the session/access options, the results are not predictable. In general it is safer to NOAUDIT the entire option, then AUDIT the desired new options.

For example:

```
SQL> audit create table by access;
SQL> noaudit create table whenever not successful;
```

Based on this, the logical result should be that auditing whenever successful is left on by access. Instead, the odd result is that the statement is present in the auditing options, but it is disabled in both states.

```
SQL> @audopts
dba_stmt_audit_opts union dba_priv_audit_opts

USER_NAME  PROXY AUDIT_OPTION                       AUD
---------- ----- ------------------------------ ----
                 CREATE TABLE                       -/-
```

The correct way to get the expected auditing is to drop the audit and reapply the desired options:

```
SQL> noaudit create table;
SQL> audit create table by access whenever successful;
```

Always check the audit options to see if the audit statement actually performed the action expected. In some cases, certain auditing combinations are not supported but the AUDIT statement succeeds. A check of the audit options will show what was actually enabled.

Whenever testing new audit options, execute a test case and review the audit trail to see that the audit signature produced is the expected one. It is better to learn what produces certain odd looking audit trail entries by generating them rather than to find them while reviewing a production audit trail and have no explanation for what they mean or how they got there.

Audit return codes

The DBA_AUDIT_TRAIL table has a RETURNCODE column which indicates the results of the auditing action. The code is the Oracle error message (ORA-nnnn) that was audited.

While a non-zero returncode is the Oracle error code, it is not necessarily the error code returned to the user. In some cases, the user gets a generic error while a more specific one is written to the audit trail to avoid giving malicious users unauthorized information.

The easiest way to look up the error code is to use the UNIX oerr facility (“oerr ora nnnn”). Here are some common codes.

RETURNCODE	Oracle Error text
0	success
1	unique constraint violated
942	table does not exist
995	invalid synonym
1004	default username feature not supported
1017	bad username/pwd
1927	cannot REVOKE privileges you did not grant
1031	insufficient privileges
2004	security violation (this error code is only written to the audit trail, the user will see a different error code)
4043	object does not exist

Removing the "ANY CLIENT" audit option

Often DBAs experimenting with AUDIT issue the following statement to see if SYS can, in fact, be audited. The following audit option is the result:

```
SQL> Audit create table by SYS;
SQL> select * from dba_stmt_audit_opts;

USER_NAME  PROXY_NAME AUDIT_OPTION   SUCCESS    FAILURE
---------- ---------- -------------- ---------- ----------
ANY CLIENT            CREATE TABLE   BY ACCESS  BY ACCESS
```

Normally, to turn off an audit option, you simply issue "NOAUDIT audit_option BY user_name". In this case it fails:

```
SQL>  noaudit create table by any client;
                        *
ERROR at line 1:  ORA-00987: missing or invalid username(s)
```

The correct statement to remove the audit option is:

```
SQL> noaudit create table by sys;
```

The "ANY CLIENT" user is used for auditing of Oracle proxy authentication, but then the proxy_name is not null. For example, here is an example of the documented behavior:

```
SQL> audit create table by scott on behalf of any;
Audit succeeded.

SQL> select * from dba_stmt_audit_opts;

USER_NAME  PROXY_NAME AUDIT_OPTION   SUCCESS    FAILURE
---------- ---------- -------------- ---------- ----------
ANY CLIENT SCOTT      CREATE TABLE   BY ACCESS  BY ACCESS

SQL> noaudit create table by scott on behalf of any;
Noaudit succeeded.
```

Other Auditing Options

Auditing and Oracle Label Security

Oracle Label Security provides an out-of-the-box implementation of Mandatory Access Control (MAC). OLS auditing is administered via the SA_AUDIT stored procedure, but the resulting audit trail is written to the AUD$ table, just as standard auditing is. Be aware that when OLS is installed, the SYS schema AUD$ table is moved into the SYSTEM schema (although it is still in the SYSTEM tablespace. While synonyms and permissions are adjusted to make this transparent, it has the potential to confuse administrators, and the security policy must address this change to avoid leaving the audit trail vulnerable to access, modification, or loss.

When using OLS auditing, be aware that OLS audit actions are not decoded by DBA_AUDIT_TRAIL. It is possible to create a view using the OLS procedure SA_AUDIT_ADMIN.CREATE_VIEW. Compare the results of the OLS audit trail view with the DBA_AUDIT_TRAIL in order to learn the decode of the action codes from OLS.

Oracle Fine-Grained Auditing

Fine-Grained Audit (FGA) allows much greater information collection by the auditor. With FGA, the auditor can log the actual SQL of a query along with bind variables. Even better, additional selectivity can be placed upon the audit policy in order to focus the triggering of the audit event. When the policy is created, a table column can be identified along with a SQL predicate such as 'salary > 100000'. The audit only occurs if a record is retrieved by the user which matches the predicate. In addition, a handler procedure can be bound which will fire when the audit event occurs. This allows capability such as notifying an auditor via pager when certain suspicious activity occurs. In Oracle 9, FGA is limited to auditing successful select statements.

Fine Grained audit has several unexpected behaviors. The audit event is triggered when the first row which meets the audit criteria is processed into the result set, and then that audit policy is ignored for the remainder of the query. Thus one audit event per query is fired, not one per record. If the query is cancelled before the first matching record is processed, or if no matching records are present (even if the query would have retrieved them), the audit is not triggered. FGA does not capture unsuccessful attempts, so it must be combined with other mechanisms to form a complete solution. FGA only works properly under the Cost Based Optimizer. If the rule based optimizer is used, false positives can occur (but no events auditable under the policy conditions will ever be missed).

"Selective Audit" tool

For US Federal customers, Oracle Consulting's Advanced Programs Group has created an audit management product called "Selective Audit." The tool is based on Enterprise Manager and supports auditing options through a GUI. In addition the tool provides text and graphical reporting with drill down capabilities. One of the best features is the ability to select a statement in the audit trail report screen and "replay" it using flashback query to see the query results as they appeared to the user who submitted the query. In addition, the tool includes a graphical interface for Oracle Logminer. Selective Audit also provides "SQL Capture," the ability to capture SQL text and bind variables for Select, Insert, Update, and Delete in Oracle 9i (SQL Capture is distinct from Fine Grained Audit).

A link to a product datasheet is included in the references section. Selective Audit may be available for commercial customers as well, check with an Oracle Sales representative.

N-tier

In an N-tier system, a user authenticates to an application server, which then performs queries against other systems such as Oracle databases on behalf of the user. For Oracle security and auditing to be effective, it is necessary for the database to know the end user identity. Oracle provides proxy authentication as one way to provide both a pooling mechanism and pass identity information to the database. When using proxy authentication, the database must have an account for the user, but the application server can “proxy” this identity over a pool account. The database audits the connection in the comment_text field during the user logon event.

A second method used by custom applications is to set the client identifier. This is a session variable that can be set and reset by the application server whenever it is doing work on behalf of an end user. The contents of this variable are written automatically into both the standard and fine-grained audit trails whenever an audit event occurs. If Oracle Label Security (OLS) is being used, the application can also use the PROFILE_ACCESS privilege to enable the user’s OLS privileges. Oracle’s Secure Application roles can also be leveraged in this scheme to securely provide object privileges specific to the user.

The whitepaper "Oracle Database 10g Security and Identity Management" provides further detail on the security and auditing technologies available for custom development. As application server code becomes more complex, attempts are made to abstract or allow the application server to handle much of the work. When this happens, the first thing lost is usually the ability to identify the end user to the database. With the growth of J2EE, there is an effort to return to placing the security rules in the application layer. Depending upon the application server, it is still possible to preserve the end user identity into the database. Piermarini shows how Oracle auditing and Fine Grained Access features can be used in a J2EE environment such that the database is aware of the end user and can authorize and audit accordingly.

Auditing Enhancements in 10G

This paper is limited to the basics of auditing in 9i to prepare a groundwork for further investigation. Arup Nanda’s “Auditing Tells All” is a great introduction to auditing features added in 10G. 10G enhances standard auditing as well as Fine Grained Auditing.

What Should be Audited?

Deciding what to audit should be dictated by requirements. The more important the data, the more important it is to audit. Many documents exist which recommend auditing as part of securing Oracle. The Center for Internet Security provides a hardening guide as well as a scoring tool. This is an excellent place to start and it provides specific recommendations for hardening as well as specific auditing to enable. Auditing is only a single part of securing the system, and all parts must be addressed.

Conclusions

Effective auditing is not an easy task. For today's multi-tier systems, it must be addressed as a requirement during the system design which addresses identity management from the client, through the middle tiers, and to the back end servers. Oracle provides a number of tools which can be incorporated to allow effective N-tier auditing. Even when designed well, auditing requires careful planning and continued maintenance. Failure to administer the process will result in missed opportunities to detect problems as well as the possibility of a system crash due to full storage devices. Auditing does require resources, and the need to audit must be carefully balanced with the benefits to determine the appropriate level of auditing. An effective audit program will modify and tune the level of auditing in response to system activity over time.

Oracle standard auditing is the most complex of all of the Oracle auditing mechanisms to use well. With a good background in standard audit, it should be possible to approach the other tools with confidence and to learn where they fit in a well rounded audit toolkit. This paper has presented a basic overview of Oracle auditing, some lessons learned, and features which commonly cause confusion. For a good understanding of Oracle audit, combine this paper with the Oracle documentation and a few practice sessions.

References

Burleson, D. (Jul, 2003) "Oracle design security from the ground up" URL: http://builder.com.com/5100-6388-5035131.html (22 Aug 2004).

Center for Internet Security, Benchmark and Scoring Tool for the Oracle Database, URL: http://www.cisecurity.org/bench_oracle.html (22 Aug 2004)

Finnigan, P. (Apr. 2003) "Introduction to Simple Oracle Auditing." URL: http://www.securityfocus.com/infocus/1689 (22 Aug 2004).

Kewley, D and Lowry, J. (June 2001), "Observations on the effects of defense in depth on adversary behavior in cyber warfare." Proceedings of the 2001 IEEE Workshop on Information Assurance and Security.
URL: http://www.itoc.usma.edu/Workshop/2001/Authors/Submitted_Abstracts/paperT3C2(18).pdf (22 Aug 2004).

National Computer Security Center, NCSC-TG-001 Version 2, A Guide to Understanding Audit in Trusted Systems, (1987), URL: http://www.radium.ncsc.mil/tpep/library/rainbow/ (22 Aug 2004).

Oracle Corporation (2003), Oracle Database Security Guide10g Release 1 (10.1), URL: http://download-west.oracle.com/docs/cd/B14117_01 /network.101/b10773/toc.htm (22 Aug 2004).

Oracle Corporation (2003), Oracle Database SQL Reference 10g Release 1 (10.1), URL: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/toc.htm (22 Aug 2004).

Oracle Corporation (Dec, 2003), "Oracle Database 10g Security and Identity Management" URL: http://www.oracle.com/technology/deploy/security/pdf/twp_security_db_securityoverview_10r1_1203.pdf (22 Aug 2004).

Oracle Corporation (2001), "Oracle Selective Audit" URL: http://www.oracle.com/industries/government/selective_audit.pdf (22 Aug 2004).

Oracle Corporation, Shortcut to full product Documentation, URL: http://tahiti.oracle.com/ (22 Aug 2004).

Oracle Technet, URL: http://www.oracle.com/technology/index.html (22 Aug 2004).

Nanda, Arup "Oracle Database 10*g*: The Top 20 Features for DBAs, Week 10, Auditing Tells All" URL: http://www.oracle.com/technology/pub/articles/10gdba/week10_10gdba.html (1 SEP 2004)

Piermarini, M. and Knox, D. (Oct. 2003), "Leveraging Oracle Database Security with J2EE Container Managed Persistence." URL: http://otn.oracle.com/tech/java/oc4j/pdf/j2ee-cmp-with-vpd.pdf (22 Aug 2004).

Appendix A: Miscellaneous Figures

Figure A- 1 DBA_AUDIT_TRAIL table definition

Column	Description
OS_USERNAME	Operating system login whose actions were audited
USERNAME	Name (not ID number) of the user audited
USERHOST	Numeric instance ID for the Oracle DB instance
TERMINAL	Identifier of the user's terminal
TIMESTAMP	Timestamp of audit or login time for LOGON action
OWNER	Schema of the object affected by the action
OBJ_NAME	Name of the object affected by the action
ACTION	Numeric type code corresponding to the action
ACTION_NAME	Text name corresponding to the ACTION. If null in DBA_AUDIT_TRAIL, the audit record was probably generated by an add-on such as Label Security and not the AUDIT command. Check those components for an audit trail view which will decode those actions.
NEW_OWNER	Schema of the NEW_NAME object
NEW_NAME	New object name after RENAME or the name of the underlying object
OBJ_PRIVILEGE	Object privileges of a GRANT or REVOKE statement
SYS_PRIVILEGE	System privileges of a GRANT or REVOKE statement
ADMIN_OPTION	The role or system priv. was granted with ADMIN option
GRANTEE	Name of grantee in a GRANT or REVOKE statement
AUDIT_OPTION	Auditing option set with the AUDIT statement
SES_ACTIONS	Session summary (a string of 16 characters, one for each action type in the order ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE, REFERENCES, and EXECUTE. Pos. 14- 16 reserved. <'-' none, S success, F failure, B both>
LOGOFF_TIME	Timestamp for user log off. LOGOFF_% fields are only populated for records with an action of LOGOFF or LOGOFF BY CLEANUP
LOGOFF_LREAD	Logical reads for the session
LOGOFF_PREAD	Physical reads for the session

LOGOFF_LWRITE	Logical writes for the session
LOGOFF_DLOCK	# of Deadlocks detected during the session
COMMENT_TEXT	For login, indicates how the user was authenticated. The method can be one of the following: DATABASE - authentication was done by password NETWORK - SQL or the Advanced Security authentication PROXY - the client was authenticated by another user. The name of the proxy user follows the method type
SESSIONID	Use this to get all audit records for a single session.
ENTRYID	Unique ID for each audit trail record
STATEMENTID	Connects all audit trail records generated by a single user SQL statement
RETURNCODE	Oracle Server message code (ORA - error).
PRIV_USED	System privilege used to execute the action

Figure A- 2 DBA_AUDIT TRAIL example of a LOGON record

```
SQL> connect aud/aud@orca
Connected.
SQL> select * from dba_audit_trail;

OS_USERNAME
----------------------------------------------------------------------------------------------------
USERNA USERHOST
------ ---------------------------------------------------------------------------------------------
TERMINAL
----------------------------------------------------------------------------------------------------
TIMESTAMP OWNER  OBJ_NAME                 ACTION ACTION_NAME          NEW_OWNER
--------- ------ -------------------- ---------- -------------------- ------------------------------
NEW_NAME
----------------------------------------------------------------------------------------------------
SES_ACTIONS      LOGOFF_TI LOGOFF_LREAD LOGOFF_PREAD LOGOFF_LWRITE LOGOFF_DLOCK
---------------- --------- ------------ ------------ ------------- ---------------------------------
COMMENT_TEXT
----------------------------------------------------------------------------------------------------
SESSIONID ENTRYID STATEMENTID RETURNCODE PRIV_USED            CLIENT_ID
--------- ------- ----------- ---------- -------------------- --------------------------------------
xxxxxx\wreeser
AUD
xxxxxx
26-AUG-04                                   100 LOGON


Authenticated by: DATABASE; Client address: (ADDRESS=(PROTOCOL=tcp)(HOST=xxx.xxx.xxx.xxx)(PORT=4380))
      665       1           1          0 CREATE SESSION


1 row selected.
```

Figure A- 3 DBA_AUDIT TRAIL example, part 2

```
OS_USERNAME
----------------------------------------------------------------------------------------------------
USERNA USERHOST
------ ---------------------------------------------------------------------------------------------
TERMINAL
----------------------------------------------------------------------------------------------------
TIMESTAMP OWNER  OBJ_NAME                 ACTION ACTION_NAME         NEW_OWNER
--------- ------ -------------------- ---------- -------------------- ------------------------------
NEW_NAME
----------------------------------------------------------------------------------------------------
SES_ACTIONS       LOGOFF_TI LOGOFF_LREAD LOGOFF_PREAD LOGOFF_LWRITE LOGOFF_DLOCK
---------------- --------- ------------ ------------ ------------- ---------------------------------
COMMENT_TEXT
----------------------------------------------------------------------------------------------------
SESSIONID ENTRYID STATEMENTID RETURNCODE PRIV_USED            CLIENT_ID
--------- ------- ----------- ---------- -------------------- --------------------------------------
xxxxxx\wreeser
AUD
xxxxxx
26-AUG-04                                   101 LOGOFF

                 26-AUG-04           95            0            11 0
Authenticated by: DATABASE; Client address: (ADDRESS=(PROTOCOL=tcp)(HOST=xxx.xxx.xxx.xxx)(PORT=4380))
      665       1           1          0 CREATE SESSION
<<<<second record begins here>>>>
xxxxxx\wreeser
AUD
xxxxxx
26-AUG-04                                   100 LOGON


Authenticated by: DATABASE; Client address: (ADDRESS=(PROTOCOL=tcp)(HOST=xxx.xxx.xxx.xxx)(PORT=4421))
      666       1           1          0 CREATE SESSION


2 rows selected.
```

Appendix B: Useful Auditing Scripts

Figure B- 1 Privileges needed to run audit scripts

```
REM
REM
REM  Privileges needed to run audit scripts
REM
REM If Oracle Label Security is installed,
REM AUD$ has been moved to SYSTEM.AUD$
REM
REM  verify that there is no existing role named audmgr before executing
REM

drop role audmgr;
create role audmgr;

grant select,delete on sys.aud$ to audmgr;
grant select on sys.dba_audit_trail to audmgr;
grant select on sys.dba_stmt_audit_opts to audmgr;
grant select on sys.dba_priv_audit_opts to audmgr;
grant select on sys.dba_obj_audit_opts to audmgr;
grant select on sys.all_def_audit_opts to audmgr;
grant select on sys.obj$ to audmgr;
grant select on sys.tab$ to audmgr;
grant select on sys.stmt_audit_option_map to audmgr;
grant select on sys.system_privilege_map to audmgr;

grant select on sys.dba_users to audmgr;
grant select on sys.dba_objects to audmgr;
grant select on sys.dba_tables to audmgr;
grant select on sys.obj$ to audmgr;
grant select on sys.tab$ to audmgr;

grant audit any to audmgr;
grant audit system to audmgr;


Create user aud identified by aud
default tablespace users
temporary tablespace temp;
grant create session, resource, audmgr to aud;

create synonym aud.aud$ for sys.aud$;
```

Figure B- 2 audopts.sql: a script to show enabled standard auditing

```
REM
REM  audopts.sql
REM
REM  Created by Wayne Reeser
REM
REM Script to report all enabled standard auditing
REM

set pages 9999
SET ECHO off
col user_name format a10
col proxy_name format a5
col audit_option format a30
col timest format a13
col userid format a8 trunc
col obn format a10 trunc
col name format a13 trunc
col sessionid format 99999
col entryid format 999
col owner format a10 wrap
col object_name hea object|name format a10 wrap
col object_type hea object|type format a6 wrap
col priv_used format a15 wrap
col privilege format a30 wrap
col aud format a4

col acname format a12 heading "Action name"

prompt dba_stmt_audit_opts union dba_priv_audit_opts
select user_name,proxy_name,audit_option,
decode(success,'BY ACCESS','A','BY SESSION','S','-')||
'/'||decode(failure,'BY ACCESS','A','BY SESSION','S','-') aud
from sys.dba_stmt_audit_opts
union
select user_name,proxy_name,privilege audit_option,
decode(success,'BY ACCESS','A','BY SESSION','S','-')||
'/'||decode(failure,'BY ACCESS','A','BY SESSION','S','-') aud
from sys.dba_priv_audit_opts
order by audit_option
/

REM the following query will produce output like this for each
REM object
REM A= by access, S= by session, - = off:

REM  ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE CRE REA WRI
REM  --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
REM  -/- -/- -/- A/A -/- -/- A/A -/- -/- -/- A/A -/- -/- -/- -/- -/-

REM   ALT   ALTER       LOC   LOCK
REM   AUD   AUDIT       REN   RENAME
REM   COM   COMMENT     SEL   SELECT
REM   DEL   DELETE      UPD   UPDATE
REM   GRA   GRANT       REF   REFERENCES (not used)
REM   IND   INDEX       EXE   EXECUTE
REM   INS   INSERT      REA   READ on Directories
REM   CRE, WRI  (CREATE and WRITE on Directories -- not supported)

REM the first part gets the default auditing options.
REM the all_def_audit_opts  doesn't have the REA col.

REM Note the final line in the where clause:
```

```
REM    and substr(alt,1,1) in ('-','A','S')
REM is intended to filter out IOT overflow segments, which
REM are classified as object type ‘table’ and have their audit
REM field filled with the NUL character if the characterset is
REM ASCII and something else, otherwise.

col alt for a3
col aud for a3
col com for a3
col del for a3
col gra for a3
col ind for a3
col ins for a3
col loc for a3
col ren for a3
col sel for a3
col upd for a3
col exe for a3
col rea for a3

prompt dba_obj_audit_opts
select 'DEFAULT' owner, 'DEFAULT' object_name, 'DEFAULT' object_type,
substr(t.audit$, 1, 1) || '/' || substr(t.audit$, 2, 1) ALT,
substr(t.audit$, 3, 1) || '/' || substr(t.audit$, 4, 1) AUD,
substr(t.audit$, 5, 1) || '/' || substr(t.audit$, 6, 1) COM,
substr(t.audit$, 7, 1) || '/' || substr(t.audit$, 8, 1) DEL,
substr(t.audit$, 9, 1) || '/' || substr(t.audit$, 10, 1) GRA,
substr(t.audit$, 11, 1) || '/' || substr(t.audit$, 12, 1) IND,
substr(t.audit$, 13, 1) || '/' || substr(t.audit$, 14, 1) INS,
substr(t.audit$, 15, 1) || '/' || substr(t.audit$, 16, 1) LOC,
substr(t.audit$, 17, 1) || '/' || substr(t.audit$, 18, 1) REN,
substr(t.audit$, 19, 1) || '/' || substr(t.audit$, 20, 1) SEL,
substr(t.audit$, 21, 1) || '/' || substr(t.audit$, 22, 1) UPD,
substr(t.audit$, 25, 1) || '/' || substr(t.audit$, 26, 1) EXE,
substr(t.audit$, 29, 1) || '/' || substr(t.audit$, 30, 1) REA
from sys.obj$ o, sys.tab$ t
where o.obj# = t.obj#
and o.owner# = 0
and o.name = '_default_auditing_options_'
and ( instr(t.audit$,'S')>0
     or instr(t.audit$,'A')>0
    )
union
select owner,object_name,object_type,
alt,aud,com,del,gra,ind,ins,loc,ren,sel,upd,exe,rea
 from sys.dba_obj_audit_opts
 where (
      alt !='-/-' or aud !='-/-' or com !='-/-'
   or del !='-/-' or gra !='-/-' or ind !='-/-'
   or ins !='-/-' or loc !='-/-' or ren !='-/-'
   or sel !='-/-' or upd !='-/-' or exe !='-/-' or rea !='-/-'
   )
   and substr(alt,1,1) in ('-','A','S')
/
```

Figure B- 3: noaudits.sql: A script to remove most audits

```
REM
REM  noaudits.sql
REM
REM  Created by Wayne Reeser
REM
REM  produces and executes a script called noaudit.sql to remove
REM  most statement/privilege audits and turn off default auditing.
REM
REM  a case where script does not work:
REM  Note: If the user_name is 'ANY CLIENT' and the proxy_name column is
REM  null in either dba_stmt_audit_opts or dba_priv_audit_opts,
REM  then you can only delete the audit
REM  by issuing "noaudit xxxx by SYS" (where xxxx is the audit option)
REM  if user_name is 'ANY CLIENT' and proxy_name is not null,
REM  for example:  if proxy_name is 'scott', then
REM    "noaudit xxxx by scott on behalf of any"
REM
REM  Case 2:  “AUDIT ALL PRIVILEGES BY SCOTT”
REM    This will produce audits which do not meet the individual
REM    NOAUDIT syntax.  The way to get rid of these is to
REM    issue “NOAUDIT ALL PRIVILEGES BY SCOTT”
REM    -- automating it is left as an exercise for the reader…
REM

set termout off
set pages 9999
set lines 71
col a format a70
set head off
set echo off
set feedback off
spool noaudit.sql

select 'noaudit '||audit_option
    ||decode(user_name,'','',' by '||user_name)||';' a
from sys.dba_stmt_audit_opts
union
select 'noaudit '||privilege
    ||decode(user_name,'','',' by '||user_name)||';' a
from sys.dba_priv_audit_opts
union
select 'noaudit all on default;' a
from dual
union
select 'noaudit all on '||owner||'.'||object_name||';' a
from sys.dba_obj_audit_opts
where (
     alt !='-/-' or aud !='-/-' or com !='-/-'
  or del !='-/-' or gra !='-/-' or ind !='-/-'
  or ins !='-/-' or loc !='-/-' or ren !='-/-'
  or sel !='-/-' or upd !='-/-' or exe !='-/-' or rea !='-/-'
  )
  and substr(alt,1,1) in ('-','A','S');

spool off;
set termout on
set head on
set feedback on
set echo on
@noaudit.sql
```

Figure B- 4: audtr.sql: Audit trail quick look

```
REM
REM  audtr.sql
REM
REM  Created by Wayne Reeser
REM
REM  produces a quick dump of the audit trail, useful when
REM  experimenting with audit.

REM  select to_char(sysdate,'YYYYMMDD HH24MISS') from dual;
REM  and replace the string in the where clause with the result
REM  to limit the records returned (if you are unable to
REM  truncate/delete the audit trail)
REM
REM  spools output to results.txt
REM

set lines 200
set trimspool on
set pages 9999
col username for a6 trunc
col owner for a6 trunc
col obj_name for a20
col action_name for a20 trunc
col priv_used for a20 trunc
col audit_option for a20 trunc
-- col returncode
col ses_actions for a16 trunc
col obj_privilege for a16 trunc
col sys_privilege for a20 trunc
col audit_option for a20 trunc
col grantee for a6 trunc
col ts for a10 trunc
spool result.txt
select username, owner, obj_name, action_name, priv_used,
   audit_option, returncode, ses_actions,
   obj_privilege,sys_privilege, admin_option,grantee,
   to_char(timestamp,'MMDDHH24MISS') ts
from dba_audit_trail
where to_char(timestamp,'YYYYMMDD HH24MISS')>'20040901 161840'
order by to_char(timestamp,'YYYYMMDD HH24MISS');
spool off
```