ELSEVIER
DIGITAL
PRESS

Ron Ben Natan

Implementing Database Security and Auditing

Includes
Examples For:
**ORACLE
SQL SERVER
DB2 UDB
SYBASE**

12

Auditing Categories

In the previous chapter you learned about several common regulations that affect database auditing projects and how to use these requirements in the context of defining an auditing project. It's time to see what auditing categories you may need to implement in your environment in order to comply with these requirements. Because the database is so rich in functionality, you can produce many types of audit trails for a database environment. This does not mean that every category mentioned in this chapter is right for you, but knowing what categories exist and how you can implement them will help you address compliance requirements.

As mentioned in the previous chapter, the key to a good auditing implementation is to understand what the requirements are and to use reverse mapping to see what requirements you can check off using the auditing categories listed in this chapter. This chapter can therefore be used as a catalog from which you can pick audit trails to implement, and possibly in what order.

12.1 Audit logon/logoff into the database

When you walk into a meeting in a corporate office, the first thing you're asked to do is sign in at the front desk. Among other things, this ensures that the company has a full log of anyone who came into the building, which may be useful to track down and investigate "who done it" when something goes wrong. This log usually records who you are, when you came in, and when you left. The same process is true for any database, and the first category of auditing that is required in most environments is a full audit trail of anyone who has signed onto the database.

You will need to record two events for this audit category: an event for the sign-on and an event for the sign-off. For each such event, you need to save at least the login name used for signing on and a timestamp for the

event, but you should consider recording additional information. This includes the TCP/IP address of the client initiating the connection and the program used to initiate the connection. For example, in an Oracle environment, you will want to know if the connection was initiated from SQL Plus, TOAD, and such tools as opposed to a data source in Excel or a J2EE server.

In addition to these two events, you should also record all failed login attempts. In fact, failed login events are probably even more important than successful logins from a security point of view. Failed login attempts are not only recorded for auditing and compliance purposes; they are often used as the basis for alerts and even for account lockout.
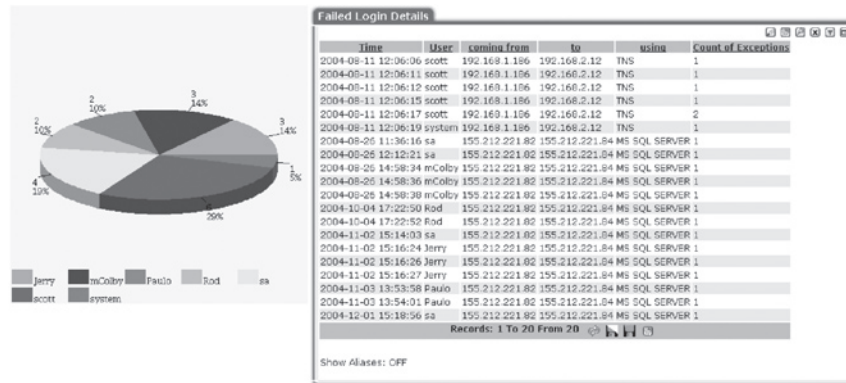
Although you may keep these three event types in the same file or table, you will probably report on them differently. Successful logon/logoff reports are not something most people look at unless they are doing some kind of investigation, because these logs reflect normal operations. Apart from investigations, an exception could be comparing files from different periods to see if patterns are changing. However, excessive failed logins are certainly an interesting security report, and many people periodically look at the breakdown of these failed login attempts based on the following dimensions:

- The username
- The client IP from which connections are failing
- Source program
- Time of day

For example, Figure 12.1 shows two views, including a breakout of failed logins based on the login name (left) and a report showing a detailed view of failed logins, what login name was used, which IP address the connection requests came from, to which database server, and what the communication type was (right).

Logon and logoff activity can be audited using database features or by using an external database security solution. All database vendors support this basic auditing function, and because the number of these events is rather small (at least when compared with the number of events you may get when auditing actual SQL calls), there is little performance penalty in having the database perform this level of auditing.

Figure 12.1
*Failed login
reports.*

In Section 9.6 you saw how to implement this type of audit trail in DB2 using event monitors and how to implement this type of audit trail in SQL Server using traces. While the context in that section was actually one of a hacker trying to plant a Trojan that collects this information to be used in launching an attack, the methods shown are precisely what you would use to create a login/logout audit trail in DB2 or SQL Server. Oracle has more than one way to produce this audit trail, but perhaps the easiest one is using system-level triggers that have been around since Oracle 8i.

Just as an Oracle trigger fires when you insert or update a row, a system-level trigger fires at specific system events such as logon, logoff, and DDL execution. Let's see how to implement this type of audit trail.

First, create a table where you will keep the information:

```
create table user_login_audit
(
   user_id          varchar2(30),
   session_id       number(8),
   host             varchar2(30),
   login_day        date,
   login_time       varchar2(10),
   logout_day       date,
   logout_time      varchar2(10)
);
```

Next, create the trigger to be fired upon a new login:

```
create or replace trigger
  user_login_audit_trigger
AFTER LOGON ON DATABASE
```

```
BEGIN
insert into user_login_audit values(
   user,
   sys_context('USERENV','SESSIONID'),
   sys_context('USERENV','HOST'),
   sysdate,
   to_char(sysdate, 'hh24:mi:ss'),
   null,
   null
);
COMMIT;
END;
```

Most of the data is populated upon login, but the logout date and time are populated using the trigger that is fired when the user logs out:
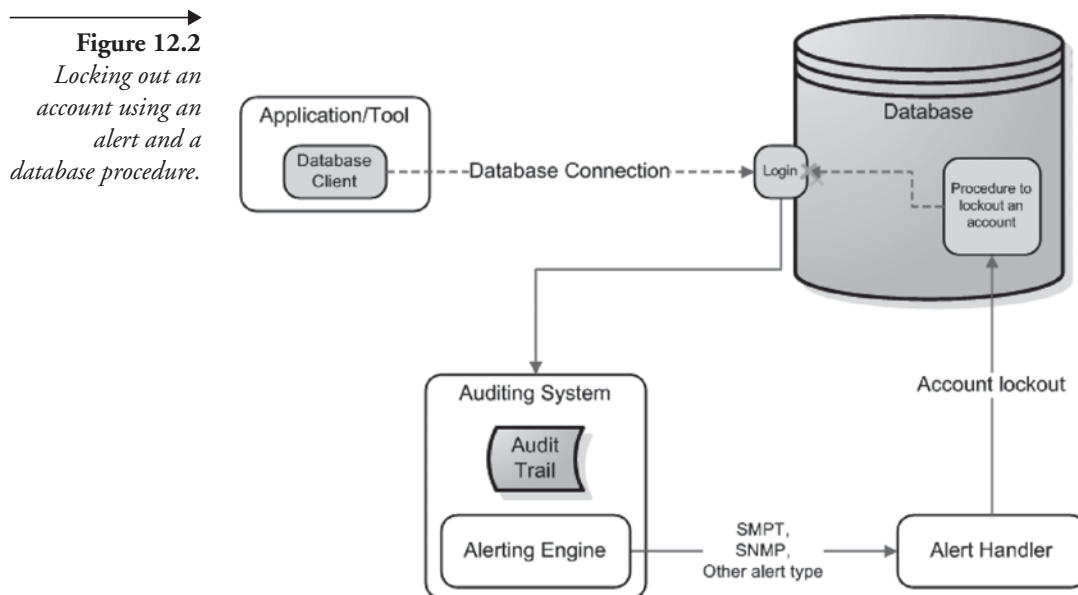
```
create or replace trigger
   user_logout_audit_trigger
BEFORE LOGOFF ON DATABASE
BEGIN
-- logout day
update
   user_login_audit
set
   logout_day = sysdate
where
   sys_context('USERENV','SESSIONID') = session_id;
-- logout time
update
   user_login_audit
set
   logout_time = to_char(sysdate, 'hh24:mi:ss')
where
   sys_context('USERENV','SESSIONID') = session_id;
COMMIT;
END;
```

That's all you need to do in an Oracle environment. If you run a Sybase environment, it is even easier, because you can audit all access to all databases using the following commands:

```
sp_configure "auditing", 1
go
sp_audit "dbaccess", "all", "all", "on"
go
```

Implementing alerting or account lockout based on failed logins requires support from either your database vendor or your database security solution. If you use the database to generate the audit trail for login/logout and your database vendor implements account lockout capabilities, then you can set that up within your database environment. For example, in Section 4.4 you saw how to set up an Oracle password policy. In another environment (e.g., SQL Server 2000), you cannot do this using native database features and need to either write code that inspects the Windows event log looking for collections of failed logins or use an external security system.

When using an external security system, you can use a SQL firewall that will block any connection using the login name after a certain number of failed login attempts. In this case, the database will not even get the connection attempts. because they would be rejected at the firewall level. Another option (which does not require you to put a security system in front of the database) is to use database procedures, as shown in Figure 12.2. In this case the auditing system generates an alert when the number of failed logins exceeds a certain threshold. The alert is sent to a system that is responsible to connect to the database and call a procedure that locks out the account. This system would typically also notify the DBA that this action has been taken so that an investigation would be initiated and the account released if needed.

Figure 12.2
Locking out an account using an alert and a database procedure.

In addition to creating an audit trail, login information can be used to create a baseline that may help you in identifying anomalies. A baseline for user login activity is a mapping of "normal" login behavior. Such a baseline is built by looking at all login activity over the course of a certain period of time (e.g., a week, a month). The baseline is built by listing all possible combinations and classifying them. For example, you can classify logins by network location, username, source programs, and time of day, in which case a baseline may look similar to the following:

```
user1   192.168.1.168    JDBC     24Hrs.
user2   192.168.X.X      Excel    Normal Business Hours (9-5)
user3   10.10.10.x       isql     Weekends
```

This baseline says that based on all of the login activity seen in the relevant recording period, user1 always comes in from 192.168.1.168 (e.g., it is the application server) and works around the clock. User2 is used to connect to the database from Excel, is used from multiple nodes on the network all within the 192.168 subnet, and is not used outside of normal business hours. Finally, user3 is used when access is initiated from isql, works over the weekend, and can come from any node on the 10.10.10 subnet.

Once you have this baseline, you can report on or alert on divergence from normal operations. If, for example, you see a successful login using user1 but from an IP address that is different from 192.168.1.168 and using a tool such as SQL Navigator, then either your environment has changed or possibly someone managed to take the username and password from the application server and is using it to extract information from your database (see Section 5.1). As another example, a login using user2 at 2 a.m. can be suspicious. It may just be that someone is working late, but depending on your environment, sensitivity, and how locked down your environment needs to be, it may be something you need to look into.

12.2 Audit sources of database usage

Related to the auditing of login activity is the auditing of client source information. This includes auditing which network node is connected to the database (e.g., using an IP address or a host name) and auditing which application is being used to access the database.

Although this information is normally one of the values you should capture when you audit database connections, it is often important to capture

Figure 12.3 *Viewing database information (IP and application type) in raw form and in business terms.*

Server Type	Server IP	Source Program	Count of Sessions	Server Type	Server IP	Source Program	Count of Sessions
MS SQL SERVER	155.212.221.84	SAP R/3	989	Financial	HR DB	SAP R/3	989
MS SQL SERVER	155.212.221.84	Aqua_Data_Studio	6	Financial	HR DB	Developer tool	6
MS SQL SERVER	155.212.221.84	Microsoft SQL Server	1	Financial	HR DB	Database link	1
MS SQL SERVER	155.212.221.84	SQL Query Analyzer	52	Financial	HR DB	SQL Query Analyzer	52
		Records: 1 To 4 From 4				Records: 1 To 4 From 4	

this information at a SQL call level. In addition to knowing that a user connected using Excel rather than the SAP system, you may also need to know whether a certain update was performed from an Excel spreadsheet as opposed to the SAP system. Therefore, the source program is often data that you should collect per query and per database operation that you want to keep in the audit trail, especially if the IP address uniquely identifies a user. If your architecture is based on client/server, then the source IP address often identifies a unique user (a person). In this case, tracking and reporting on the IP address per SQL call is as good as reporting on which end user did what operation and looked at what data—a valuable audit trail. If, on the other hand, you use an application server architecture, then the IP address will not help you identify and report on the end user and you will have to resort to techniques learned in Chapter 6.

Another decision that you will normally have to make when auditing and presenting audit information has to do with whether you present raw data or whether you present it as data that is easier to consume. For example, the left side of Figure 12.3 shows which source programs are used to access the SQL Server running on 155.212.221.84. This information is useful to people who know the environment intimately. The report on the right side of Figure 12.3 is meaningful to more people, who don't care about the IP address but know what the HR database is, and people who don't know what Aqua Data Studio is but understand the risks associated with a developer tool logged into the production HR database.

The issue of data abstraction is not only related to auditing the client source of database usage. It is a general topic relevant to all audits that are discussed in this chapter. However, as Figure 12.4 shows, it is especially important in source identification, where IP addresses may not be meaningful but where hostnames or even labels attached to nodes are informative.

Figure 12.4
*Viewing client
source information
(client IP and
source application)
in raw form and in
business terms.*

Client IP	Source Program	SQL Verb	Depth	Object Name	Total access
155.212.221.82	SQL Query Analyzer	SELECT	0	products	9
155.212.221.82	SQL Query Analyzer	SELECT	0	customers	4
155.212.221.82	SQL Query Analyzer	SELECT	0	fred1	6
155.212.221.82	SQL Query Analyzer	CREATE TABLE	0	fred	3
155.212.221.82	SQL Query Analyzer	SELECT	0	fred	7
155.212.221.82	SQL Query Analyzer	INSERT	0	fred	6
155.212.221.82	SQL Query Analyzer	DELETE	0	fred	4
155.212.221.82	SQL Query Analyzer	DROP TABLE	0	fred	4
155.212.221.82	SQL Query Analyzer	EXECUTE	0	sp_addlogin	8
155.212.221.82	SQL Query Analyzer	GRANT	0	mcolby	4
155.212.221.82	SQL Query Analyzer	SELECT	0	@@trancount	14
155.212.221.82	SQL Query Analyzer	GRANT	0	dave	2
155.212.221.82	Aqua_Data_Studio	SELECT	0	master.dbo.sysobjects	1
155.212.221.82	Aqua_Data_Studio	SELECT	0	charindex	1
155.212.221.82	Aqua_Data_Studio	SELECT	0	type_name	1
155.212.221.82	Aqua_Data_Studio	SELECT	0	convert	1
155.212.221.82	Aqua_Data_Studio	USE	0	master	3
155.212.221.82	Aqua_Data_Studio	SELECT	0	char	1
155.212.221.82	Aqua_Data_Studio	SELECT	0	odbcscale	1
155.212.221.82	Aqua_Data_Studio	SELECT	0	master.dbo.syscolumns	1

Records: 21 To 40 From 1364

Client IP	Source Program	SQL Verb	Depth	Object Name	Total access
Dave's PC	SQL Query Analyzer	SELECT	0	products	9
Dave's PC	SQL Query Analyzer	SELECT	0	customers	4
Dave's PC	SQL Query Analyzer	SELECT	0	fred1	6
Dave's PC	SQL Query Analyzer	CREATE TABLE	0	fred	3
Dave's PC	SQL Query Analyzer	SELECT	0	fred	7
Dave's PC	SQL Query Analyzer	INSERT	0	fred	6
Dave's PC	SQL Query Analyzer	DELETE	0	fred	4
Dave's PC	SQL Query Analyzer	DROP TABLE	0	fred	4
Dave's PC	SQL Query Analyzer	EXECUTE	0	sp_addlogin	8
Dave's PC	SQL Query Analyzer	GRANT	0	mcolby	4
Dave's PC	SQL Query Analyzer	SELECT	0	@@trancount	14
Dave's PC	SQL Query Analyzer	GRANT	0	dave	2
Dave's PC	Developer tool	SELECT	0	master.dbo.sysobjects	1
Dave's PC	Developer tool	SELECT	0	charindex	1
Dave's PC	Developer tool	SELECT	0	type_name	1
Dave's PC	Developer tool	SELECT	0	convert	1
Dave's PC	Developer tool	USE	0	master	3
Dave's PC	Developer tool	SELECT	0	char	1
Dave's PC	Developer tool	SELECT	0	odbcscale	1
Dave's PC	Developer tool	SELECT	0	master.dbo.syscolumns	1

Records: 21 To 40 From 1364

12.3 Audit database usage outside normal operating hours

Another topic that is related to the audit of database login is an audit of activities being performed outside of normal business hours. This is an intuitive requirement and one that is often required from a business and a compliance standpoint.

The intuitive requirement of auditing database usage outside of normal operating hours is needed because activities performed during off-hours are often suspect and may be a result of an unauthorized user trying to access or modify data. Of course, a good hacker will usually try to breach the database during a "camouflaged" period. It is far better to try when there is a lot of "noise" that serves as a diversion. However, less sophisticated misuse does often occur at night or early in the morning, and many people do watch a lot of movies that have people sneaking around the office at night doing inappropriate things.

When you audit off-hours activity, it is usually not enough to track only logins and logouts that occur off-hours. You will generally also want to capture what activities are performed—usually at a SQL level. If such logins are suspect, then it is important to capture what they were used to do within the database. Having a full audit trail of all activities that were performed by any user outside of normal operating hours is therefore often a good category to implement and will satisfy many regulatory and internal compliance requirements.

Although intuitively an off-hours audit trail makes a lot of sense, at a technical level you must be clear on the definition, because most database environments work 24-by-7, and you don't want to start generating tons of false alarms whenever an ETL script performs massive data uploads outside normal operating hours. Therefore, the key to a good implementation of

this audit trail is not to include activities that are *always* scheduled to run in off-hours as part of this audit trail.

Another approach to filtering out the normal activities that occur outside normal hours is to use a baseline. If you baseline your database access, you may see activities such as the following:

```
user1  192.168.1.168   SQLLoader  2am-4am
user2  192.168.1.168   ETL        12am-6am
```

If you see this type of activity occurring every night, then your off-hours audit trail should exclude any activity performed by these applications, using these login names, and coming from these IP addresses (or, as is often the case, from the localhost). Auditing only what diverges from the baseline helps reduce the size of the audit trails you may need to inspect, because activities that will be recorded are only those activities that are occurring outside of the norm.

12.4 Audit DDL activity

Schema change audits, or, more specifically, DDL activity audits have always been important and have recently become one of the most implemented audit trails. This is perhaps because schema change audits are important from a security standpoint, from a compliance standpoint, and from a configuration management and process standpoint. From a security standpoint, DDL commands are potentially the most damaging commands that exist and can certainly be used by an attacker to compromise any system. Even stealing information may often involve DDL commands (e.g., through the creation of an additional table into which data can be copied before extraction). From a compliance standpoint, many regulations require you to audit any modification to data structures such as tables and views. Some HIPAA requirements, for example, can be directly interpreted as a need to audit schema changes.

Regulatory requirements to audit schema changes are not always needed because of security. Sometimes the need is to avoid errors and to discover problems quickly. It is therefore not surprising that compliance requirements for schema changes auditing are often similar to the requirements defined as part of configuration management and IP governance initiatives. The example with HIPAA and schema changes is a good one. Changes to the schema need to be audited and saved for future reference as a way to identify and quickly resolve errors that may compromise data portability or

that may cause data corruption. In other cases, auditing of DDL activity is done to eliminate errors that developers and DBAs may introduce and that can have catastrophic effects. For example, a client I once worked for had a downtime of almost two days because of a change that was done by a developer—a change that the developer thought was being done on the development server but was mistakenly done on the production server. Tight controls over the configuration management process are important and one of the primary drivers of DDL audits.

There are three main methods to audit schema changes:

1.   Use database audit features

2.   Use an external auditing system

3.   Compare schema snapshots

Most database environments will allow you to audit DDL activity using audit mechanisms, event monitors, traces, and so forth. As an example, Oracle allows you to use system triggers based on DDL:

```
create table ddl_audit_trail
(
   user_id           varchar2(30),
   ddl_date          date,
   event_type        varchar2(30),
   object_type       varchar2(30),
   owner             varchar2(30),
   object_name       varchar2(30)
);
create or replace trigger
  DDL_trigger
AFTER DDL ON DATABASE
BEGIN
    insert into ddl_audit_trail (
       user_id,
       ddl_date,
       event_type,
       object_type,
       owner,
       object_name
    )
    VALUES
    (
```

```
                ora_login_user,
                sysdate,
                ora_sysevent,
                ora_dict_obj_type,
                ora_dict_obj_owner,
                ora_dict_obj_name
        );
    END;
```

In DB2 you use audit traces, in SQL Server trace functions, and in Sybase native auditing. In all cases it is up to you to extract the information, produce reports, and create baselines if you want to do so. This is where the second category comes in: external auditing tools. These tools not only collect the information on your behalf, but they also provide the tools for reporting, alerting, and advanced functions such as baselining.

The third category—comparing schema snapshots—does not give you a detailed audit trail of DDL activity and is inferior to the other two categories but is relatively easy to implement and can be used as a temporary solution until you implement a true auditing infrastructure. It is based on periodically collecting a full definition of the schema (typically once a day) and comparing the schema with the schema from the night before. Even a simple tool like diff can be used, because all you are trying to do in this method is determine whether changes have occurred. Although this method is fairly easy to implement, it suffers from the fact that when a change is made, you cannot track down who did it, when, or why. Also, if someone maliciously made a change, used it, and then rolled it back to what it was before the change, you will not see it so long as the whole process took less than a day. Therefore, this alternative is sometimes sufficient in a configuration management initiative but is often not good enough in a project driven by security or compliance requirements.

12.5 Audit database errors

Auditing errors returned by the database is important and is one of the first audit logs you should implement. This is especially true from a security standpoint, and you have seen many instances where this would be important. For example, when we discussed SQL injection attacks in Chapter 5, one of the things you learned is that in many cases attackers will make many attempts until they get it right. The example used was a UNION-based attack in which attackers need to guess the right number of columns. Until they get the right number, the database will continuously return an error code saying that the columns selected by the two SELECT statements

do not correspond. If you are logging all errors, you can identify this situation and react. Failed logins are another good example of an error that needs to be logged and monitored, even if you are not auditing logins to the database. Finally, any failed attempt to elevate privileges is a strong indicator that an attack may be in progress.

Errors are also important from a quality perspective, and this also maps well to compliance. Production applications that are causing errors because of bugs and application issues should be identified and fixed. Logging SQL errors is often a simple way to identify these problems. Therefore, even when your primary concern is a security initiative, providing this information to the application owners can make you a hero, because no one likes running code that still has issues that can usually be easily resolved. If you're lucky, these errors might even point you in the direction of problems that affect response time and availability.

Detailed error auditing is supported by some of the database vendors, and you can refer to the reference guide of your environment to see how to do this. In Oracle you can again use system triggers:

```
create table error_audit
(
   user_id           varchar2(30),
   session_id        number(8),
   host              varchar2(30),
   error_date        date,
   error             varchar2(100)
);
```

Next, create the trigger to be fired upon a new login:

```
create or replace trigger
  audit_errors_trigger
AFTER SERVERERROR ON DATABASE
BEGIN
insert into error_audit values(
   user,
   sys_context('USERENV','SESSIONID'),
   sys_context('USERENV','HOST'),
   sysdate,
   dbms_standard.server_error(1)
);
COMMIT;
END;
```

In SQL Server you can use either auditing features or trace features. If you choose to use traces, you need to set up the appropriate events that are relevant to errors using `sp_trace_event`. These include the event IDs shown in Table 12.1:

Table 12.1 *Event IDs and description relevant to error audits*

Event ID	Event Class	Description
16	Attention	Collects all attention events, such as client-interrupt requests or when a client connection is broken.
21	ErrorLog	Error events have been logged in the error log.
22	EventLog	Events have been logged in the application log.
33	Exception	Exception has occurred in the server.
67	Execution Warnings	Any warnings that occurred during the execution of a server statement or stored procedure.
55	Hash Warning	Hashing operation may have encountered a problem.
79	Missing Column Statistics	Column statistics for the query optimizer are not available.
80	Missing Join Predicate	Executing query has no join predicate. This can result in a long-running query.
61	OLEDB Errors	OLE DB error has occurred.

Multiple DB2 event monitors are relevant to error audits, and you may have to use a number of these types. For each that you feel is needed, you will need to filter those records that are related to errors. For example, you should select CHECKING events for ACCESS DENIED records and look at AUTHENTICATE_PASSWORD and VALIDATE_USER events in the VALIDATE category.

Although error logging and auditing are possible in some environments, this is one of the areas in which an external auditing system really shines (especially one that is based on inspecting all requests and responses, as described in Section 13.3). If you monitor all incoming SQL calls and all responses, tracking and reporting all errors is simple and does not put any

additional burden on the database. Errors can be reported using any set of criteria, and the information is readily available for building a baseline.

Baselining is important if your application environment is less than perfect. Not every database and application environment is squeaky clean, and in most environments some applications generate database errors even in production. However, errors that are generated by the applications are repetitive: the same errors occur at approximately the same place because the errors usually result from bugs—and these don't change. If you baseline errors and suddenly see errors occurring from different places or you see completely different error codes, then you should investigate what is going on.

12.6   Audit changes to sources of stored procedures and triggers

In Chapter 9 you learned about database Trojans and the importance of monitoring code changes made to triggers and stored procedures. Because these database constructs use flexible and fully featured procedural programming languages, it is easy to hide malicious code that would otherwise be undetectable. Therefore, you should adopt this best practice and audit all changes made to these constructs.

As in previous sections, this category can also be audited in several ways. The most primitive way is based on configuration control and can be implemented by periodically (e.g., daily) retrieving the code from the databases and comparing it with the code retrieved from the previous time

Figure 12.5
Real-time source change tracking for procedure source code changes.

period. This method is relatively simple to implement using a set of tools and scripts such as diff.

The second option, which was presented in Chapter 9, is to use an external database security and auditing system. Such systems can alert you on any create or modify command in real time and can easily produce a set of reports detailing the changes—both for procedures (e.g., Figure 12.5) and triggers (e.g., Figure 12.6).

The third option is to use a built-in database feature. For example, in SQL Server you can use the Recompile event to track changes to stored procedure:

Event ID	Event Class	Description
37	SP:Recompile	Indicates that a stored procedure was recompiled.

In most database environments, this feature would be supported through DDL audits, although it is not always easy to extract the source code from the commands and keep it in a way that is presentable to an auditor.

Figure 12.6
Real-time source change tracking for trigger source code changes.

12.7   Audit changes to privileges, user/login definitions, and other security attributes

This category is a must-have for database auditing; you should maintain a complete audit trail of any changes made to the security and privilege model of your database. The database manages a sophisticated scheme of security and permissions and changes, but the number-one rule in security is that changes to the security model must be audited. You should consider auditing the following changes:

- Addition and deletion of users, logins, and roles
- Changes to the mappings between logins and users/roles
- Privilege changes—whether by user or role
- Password changes
- Changes to security attributes at a server, database, statement, or object level

Because the security model within the database is the gateway, any changes to permissions and privileges must be audited. Attackers will often try to raise their privilege levels, and mistakes are often made when grants are inappropriately provided. A full audit trail of all changes that may affect the database's security is therefore akin to placing a surveillance camera watching the front door of the building, the place where the entry code is changed, and the place where badges are issued

As in previous auditing categories, you have three methods for auditing security attributes. However, because security permission changes can be hazardous to the database (in case of an attack scenario), you shouldn't rely on a once-a-day type of comparison and should opt for real-time notification of changes that are not planned in a production environment. This means you should either use an external database security and auditing system or build real-time alerts based on audit trails produced using built-in database mechanisms.

If you are going to implement this system yourself, you will need to capture relevant events and then build the alerting framework. Generating these events within the various database environments is similar to what you have already seen in previous sections. As an example, Table 12.2 shows you the relevant trace events available for SQL Server. In DB2, SEC-

Table 12.2   *Security-related SQL Server trace events*

Event ID	Event Class	Description
102	Audit Statement GDR	Occurs every time a GRANT, DENY, REVOKE for a statement permission is issued by any user in SQL Server.
103	Audit Object GDR	Occurs every time a GRANT, DENY, REVOKE for an object permission is issued by any user in SQL Server.
104	Audit Add/Drop Login	Occurs when a SQL Server login is added or removed—`sp_addlogin` and `sp_droplogin`.
105	Audit Login GDR	Occurs when a Windows login right is added or removed—`sp_grantlogin`, `sp_revokelogin`, and `sp_denylogin`.
106	Audit Login Change Property	Occurs when a property of a login, except passwords, is modified—`sp_defaultdb` and `sp_defaultlanguage`.
107	Audit Login Change Password	Occurs when a SQL Server login password is changed.
108	Audit Add Login to Server Role	Occurs when a login is added or removed from a fixed server role—`sp_addsrvrolemember` and `sp_dropsrvrolemember`.
109	Audit Add DB User	Occurs when a login is added or removed as a database user (Windows or SQL Server) to a database—`sp_grantdbaccess`, `sp_revokedbaccess`, `sp_adduser`, and `sp_dropuser`.
110	Audit Add Member to DB	Occurs when a login is added or removed as a database user (fixed or user-defined) to a database—`sp_addrolemember`, `sp_droprolemember`, and `sp_changegroup`.
111	Audit Add/Drop Role	Occurs when a login is added or removed as a database user to a database—`sp_addrole` and `sp_droprole`.
112	App Role Pass Change	Occurs when a password of an application role is changed.
113	Audit Statement Permission	Occurs when a statement permission (such as CREATE TABLE) is used.
114	Audit Object Permission	Occurs when an object permission (such as SELECT) is used, both successfully or unsuccessfully.

MAINT is one of the six auditing categories and generates records when granting and revoking object or database privileges, or when granting and revoking DBADM authority. Records are also generated when the database manager security configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, or SYSMAINT_GROUP are modified. Table 12.3 lists the possible SECMAINT privileges or authorities.

If you are using an external system that supports both auditing and real-time alerts, then you can add rules to your alerting policy that will inform you when security procedures or commands are used. For example, in an Oracle environment, you need to audit all uses of GRANT, CREATE USER, ALTER USER, DROP USER, REVOKE, CREATE ROLE, ALTER PROFILE, CREATE PROFILE, ALTER ROLE, and so on. In this case you can set up a group of commands you want to track, as shown in

Table 12.3   *DB2 SECMAINT events*

Event	Description
Control Table	Control privilege granted or revoked on a table or view
ALTER TABLE	Privilege granted or revoked to alter a table
ALTER TABLE with GRANT	Privilege granted or revoked to alter a table with granting of privileges allowed
DELETE TABLE	Privilege granted or revoked to drop a table or view
DELETE TABLE with GRANT	Privilege granted or revoked to drop a table with granting of privileges allowed
Table Index	Privilege granted or revoked on an index
Table Index with GRANT	Privilege granted or revoked on an index with granting of privileges allowed
Table INSERT	Privilege granted or revoked on an insert on a table or view
Table INSERT with GRANT	Privilege granted or revoked on an insert on a table with granting of privileges allowed
Table SELECT	Privilege granted or revoked on a select on a table
Table SELECT with GRANT	Privilege granted or revoked on a select on a table with granting of privileges allowed
Table UPDATE	Privilege granted or revoked on an update on a table or view

Table 12.3   *DB2 SECMAINT events (continued)*

Event	Description
Table UPDATE with GRANT	Privilege granted or revoked on an update on a table or view with granting of privileges allowed
Table REFERENCE	Privilege granted or revoked on a reference on a table
Table REFERENCE with GRANT	Privilege granted or revoked on a reference on a table with granting of privileges allowed
CREATEIN Schema	CREATEIN privilege granted or revoked on a schema.
CREATEIN Schema with GRANT	CREATEIN privilege granted or revoked on a schema with granting of privileges allowed
DROPIN Schema	DROPIN privilege granted or revoked on a schema
DROPIN Schema with GRANT	DROPIN privilege granted or revoked on a schema with granting of privileges allowed
ALTERIN Schema	ALTERIN privilege granted or revoked on a schema
ALTERIN Schema with GRANT	ALTERIN privilege granted or revoked on a schema with granting of privileges allowed
DBADM Authority	DBADM authority granted or revoked
CREATETAB Authority	CREATETAB authority granted or revoked
BINDADD Authority	BINDADD authority granted or revoked
CONNECT Authority	CONNECT authority granted or revoked
Create not fenced Authority	Create not fenced authority granted or revoked
Implicit Schema Authority	Implicit schema authority granted or revoked
Server PASSTHRU	Privilege granted or revoked to use the pass-through facility with this server (federated database data source)
Table Space USE	Privilege granted or revoked to create a table in a table space
Table Space USE with GRANT	Privilege granted or revoked to create a table in a table space with granting of privileges allowed
Column UPDATE	Privilege granted or revoked on an update on one or more specific columns of a table

Table 12.3 *DB2 SECMAINT events (continued)*

Event	Description
Column UPDATE with GRANT	Privilege granted or revoked on an update on one or more specific columns of a table with granting of privileges allowed
Column REFERENCE	Privilege granted or revoked on a reference on one or more specific columns of a table
Column REFERENCE with GRANT	Privilege granted or revoked on a reference on one or more specific columns of a table with granting of privileges allowed
LOAD Authority	LOAD authority granted or revoked
Package BIND	BIND privilege granted or revoked on a package
Package BIND with GRANT	BIND privilege granted or revoked on a package with granting of privileges allowed
EXECUTE	EXECUTE privilege granted or revoked on a package or a routine
EXECUTE with GRANT	EXECUTE privilege granted or revoked on a package or a routine with granting of privileges allowed
EXECUTE IN SCHEMA	EXECUTE privilege granted or revoked for all routines in a schema
EXECUTE IN SCHEMA with GRANT	EXECUTE privilege granted or revoked for all routines in a schema with granting of privileges allowed
EXECUTE IN TYPE	EXECUTE privilege granted or revoked for all routines in a type
EXECUTE IN TYPE with GRANT	EXECUTE privilege granted or revoked for all routines in a type with granting of privileges allowed
CREATE EXTERNAL ROUTINE	CREATE EXTERNAL ROUTINE privilege granted or revoked
QUIESCE_CONNECT	QUIESCE_CONNECT privilege granted or revoked

Figure 12.7. Then, add a rule to a policy that alerts you when any such command is used (e.g., the rule in Figure 12.8). The rule within the policy ensures that you will get an alert on such a command, but even without it you will still have a full audit trail that includes all occurrences of any one of the commands in the group.

Figure 12.7
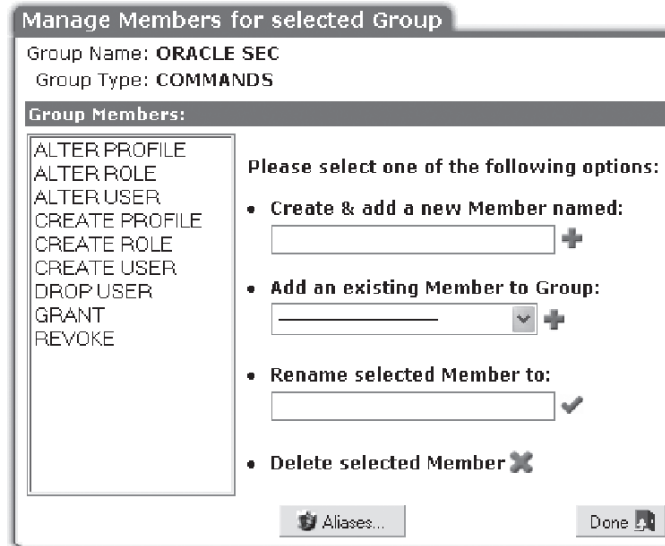*Group of commands used for tracking changes to privileges in Oracle.*

Figure 12.8
A real-time alert based on the group of commands shown in Figure 12.7.

12.8   Audit creations, changes, and usage of database links and of replication

Contrary to some of the previous categories, audits for links, synonyms, or nicknames and auditing of replication processes is an example where a periodic extraction and comparison is usually good enough. While you still have the three options—comparing snapshots, using the database's internal audit mechanisms, and using an external auditing and security system—a simple implementation using daily diffs is often good enough. In this case you only need a script that queries these definitions and places them into a file that you can use to compare with the next day.

If you prefer auditing using the internal database auditing mechanisms or using an external auditing system, then you will have to base these audit trails on objects and commands. In most database environments, there are no specific audit capabilities for replication and links. However, there are

many specific objects and commands that you can audit on. These were listed in Chapter 8. For example, Table 8.1 shows commands using an Oracle-centric replication scheme, Figures 8.6 and 8.7 show DB2 objects related to replication, and Figures 8.13 to 8.15 show SQL Server objects related to replication.

12.9   Audit changes to sensitive data

Auditing of DML activity is another common requirement, especially in scenarios such as a Sarbanes-Oxley project where accuracy of financial information is the main event. Data change audit trails are common in almost all major auditing initiatives.

A related auditing requirement that sometimes comes up (although it is not as common as the need to audit the DML activity) involves full recording of the old and the new value per DML activity. For example, you may need to create an audit trail for the column of an employee table in which yearly bonuses are stored. In this case you may have two different requirements. The first may be to fully record any update to these values and for each update record the user who performed the update, which client was used, which application was used, when it was done, and what the actual SQL statement was. A second requirement may be to record all of the above information but also record what the value was before the update and what the value was after the update. This is not always the same thing because I can give myself 50% more of a bonus by using a command such as the following:

```
UPDATE EMP SET BONUS = BONUS * 1.5
```

DML audit trails and recording old and new values are an important type of audit that you will probably need to include in your bag of tricks. However, you have to be careful with this category and realize that these audits should be done selectively. In some cases, people are overzealous about this type of audit trail and for the sake of simplicity think about activating it for every DML operation. While this is technically possible, the amount of data produced can be large, and you should make sure that your auditing infrastructure can manage this load, especially when you include the old and new values. As an example, suppose that you have 1 million DML transactions per day, and assume for simplicity that each transaction updates a single value, that you have 100 tables in the database each one with 10 values that may be updated, and that you start out with a database

that has 10,000 records in each table. Although this calculation is simplistic and imprecise, you should not be surprised that if you record old and new values, after one year your auditing database will be more than 35 times larger than the database itself.

Therefore, when you contemplate DML audit trails, you should selectively choose which objects and which commands to audit. For example, you can decide to create audit trails for a subset of the database tables, for a subset of logins or accounts, and so on. Even more selective is the choice of which tables and columns to maintain old and new values for.

DML audits are also supported through three main methods, but comparing daily (or periodic) snapshots is not an option in this case. The three methods are using database capabilities, using an external audit system, or using triggers.

All databases give you some way to implement audit trails for DML activities. In Oracle, for example, you can use the log miner tool that is based on the redo log. Because the redo log captures all DML activity (including the old and new values), log miner can extract this information and make it available to you. In SQL Server you can use a DOP trace event:

Event ID	Event Class	Description
28	DOP Event	Occurs before a SELECT, INSERT, or UPDATE statement is executed.

Moving on to the second category, external database audit systems support DML audits based on any filtering criteria, including database object, user, application, and so on. They also help in capturing and compressing this information and making it available to reporting frameworks even when the amount of data is overwhelming. As you'll see in the next chapter (Section 13.3), some of these tools are also based on mining the redo log (or transaction log).

Finally, the third option is simply to use your own custom triggers. This option is technically inferior to the other two options, but if you are not part of a widescale auditing project and just need to create a DML audit trail for a few objects, adding triggers that write the information to a special audit table may be the simplest and quickest thing to help you move on to your next project.

12.10 Audit SELECT statements for privacy sets

SELECT statements have not been the focus of audit trails in the past, but the recent focus on privacy has changed all that. If you need to ensure privacy for a California Senate Bill 1386 project, need to conform to GLBA-type privacy issues, or just need to assure your customers, partners, and employees that their confidential information does not leak from your databases, then you will have to start to audit SELECT statements. Specifically, you will need to be able to display where the SELECT statement came from (IP address, application), who selected the data (username), and what data was actually selected. As in the case of DML audit trails, auditing of SELECT statements is impractical for the entire database, and you need to focus on subsets that are meaningful and necessary.

The first step is a classification of what data is important in terms of a SELECT audit trail. I call this a *privacy set* because in real life collections of data values *together* are important from a privacy perspective. For example, my last name is not confidential, but my last name together with my driver's license number and my Social Security number is confidential. In the classification stage you should define where confidential information resides (which object names and which column names) and what combination is confidential. A privacy set is therefore a collection of 2-tuples, each tuple consisting of an object name and a column name.

Suppose, for example, that you have two tables for recording personal and driving information. The first table is called PERSON and the second is called LICENSE. Assume that these tables include the following fields:

```
PERSON
----------
ID—int 10
FirstName—varchar 25
MiddleInitial—char 1
LastName—varchar 25

LICENSE
-------------
LicNum—varchar 12
State—varchar 2
PersonID—int 10
```

In this case your privacy set may be:

```
{<PERSON, FirstName>, <PERSON, LastName>, <LICENSE, LicNum>}
```

In order to audit the privacy set, you need to ensure that the value for `<LICENSE, LicNum>` comes from the record with a `PersonID` matching the `ID` in the record from which `<PERSON, FirstName>` and `<PERSON, Last-Name>` are derived. Once you classify where your private information resides, you can turn to creating audit trails. This will ensure that you're not collecting too much information to process.

Creating SELECT audit trails is usually more difficult than for other audit categories. Obviously, snapshots are not an option here and neither are triggers, so you're left with using database traces or an external auditing system. There is also the option of building views with custom logging, but that tends to be too much work and requires too many changes. Even when using internal database features, your options are a bit more limited. For example, even if you have support for SELECT traces (e.g., using the DOP event in SQL Server as shown following), it is often not practical because you would be collecting too much information and would need to apply filters.

Event ID	Event Class	Description
28	DOP Event	Occurs before a SELECT, INSERT, or UPDATE statement is executed.

Therefore, when you need to do SELECT auditing, your best choice is often to use an external database auditing system. Note that not all approaches (see Section 13.3) support SELECT auditing; as an example, a solution that is based on the transaction log (the redo log) will not help with a SELECT audit trail.

12.11 Audit any changes made to the definition of what to audit

Audit changes made to the definition of the audit trail and any modification that may be made to the audit trail itself. If you have cameras looking at a building, you will want to monitor any maintenance made to the cameras and any changes made to the cameras in terms of where they are pointing. Otherwise, an intruder could first point the cameras at the wall (or attach a static picture to the camera as we've all seen in many movies) and then proceed to walk right through the door. In the same way, if you do not audit changes made to the audit trail, an attacker can either change the definitions of what is being audited or can come after the fact and change the

audit trail. Note that one part of this category involves an additional audit trail and one part involves the notion of segregation of duties, which was discussed in Chapter 11 and is discussed further in Section 13.2.

You can implement this audit trail using built-in database features or an external database security and auditing system. As an example, DB2 has an audit category called AUDIT that generates records when audit settings are changed or when the audit log is accessed, and SQL Server has the following trace event that you may use:

Event ID	Event Class	Description
117	Audit Change Audit	Occurs when audit modifications are made.

If you choose to use an external auditing system, make sure it implements full auditing capabilities within itself, as described in Section 13.6.

12.12 Summary

In this chapter you learned about the various categories of audit trails that you may need to implement in order to secure your database environment and/or in order to comply with regulatory or internal requirements. As mentioned in Chapter 11, while requirements may differ, they are usually easily mapped to a set of database auditing capabilities. These were the main focus of this chapter and were cataloged by this chapter.

In addition to selecting among the various auditing categories that are relevant to your needs, you need to select the methods and systems used to implement audit trails and may need to make architectural decisions. These are important because auditing is not a one-time effort, and anything you put in place must be sustainable over time. Therefore, you should understand what the different options are and what attributes to look for, which is the topic of the next chapter.

Implementing Database Security and Auditing

A Guide for DBAs, information security administrators and auditors

Ron Ben Natan

Today, databases house our "information crown jewels", but database security is one of the weakest areas of most information security programs. With this excellent book, Ben-Natan empowers you to close this database security gap and raise your database security bar!

—**Bruce W. Moulton,** CISO/VP, Fidelity Investments (1995 - 2001)

It's been said that everyone has their 15 minutes of fame. You certainly don't want to gain yours by allowing a security breach in your database environment or being the unfortunate victim of one. Information and Data are the currency of On Demand computing, and protecting their integrity and security has never been more important. Ron's book should be compulsory reading for managing and maintaining a secure database environment.

—**Bob Picciano,** VP Database Servers, IBM

Let's start with a simple truth about today's world: If you have a database and you make it available to customers, employees, or whomever over a network, that database will be attacked by hackers–probably sooner rather than later. If you are responsible for that database's security, then you need to read this book. No other single source covers all of the many disciplines and layers involved in protecting exposed databases, and it especially shines in synthesizing all of its concepts and strategies into very practical and specific checklists of things you need to do. I've been an Oracle DBA for 15 years, but I'm not embarrassed to admit that five minutes into Chapter One I was making notes on simple measures I had overlooked.

—**Charles McClain,** Senior Oracle DBA
North River Consulting, Inc.

This book is about database security and auditing. You will learn many methods and techniques that will be helpful in securing, monitoring and auditing database environments. The book covers diverse topics that include all aspects of database security and auditing - including network security for databases, authentication and authorization issues, links and replication, database Trojans, etc. You will also learn of vulnerabilities and attacks that exist within various database environments or that have been used to attack databases (and that have since been fixed). These will often be explained to an "internals" level. There are many sections which outline the "anatomy of an attack" before delving into the details of how to combat such an attack. Equally important, you will learn about the database auditing landscape–both from a business and regulatory requirements perspective as well as from a technical implementation perspective.

- Useful to the database administrator and/or security administrator–regardless of the precise database vendor (or vendors) that you are using within your organization

- Has a large number of examples–examples that pertain to Oracle, SQL Server, DB2, Sybase and even MySQL..

- Many of the techniques you will see in this book will never be described in a manual or a book that is devoted to a certain database product

- Addressing complex issues must take into account more than just the database and focusing on capabilities that are provided only by the database vendor is not always enough. This book offers a broader view of the database environment– which is not dependent on the database platform–a view that is important to ensure good database security

Ron Ben Natan is CTO at Guardium Inc., a leader in database security and auditing. Prior to Guardium Ron worked for companies such as Intel, AT&T Bell Laboratories, Merrill Lynch, J.P. Morgan and ViryaNet. He holds a Ph.D. in the field of distributed computing from the University of Jerusalem. Ron is an expert on the subject of distributed application environments, application security and database security and has authored nine technical books and numerous articles on these topics.

Audience: DBA's, System and Network
Administrators and Auditors

ELSEVIER
DIGITAL
PRESS

books.elsevier.com/digitalpress

ISBN:1-55558-334-2

90000

9 781555 583347