EPiC Series in Computing

Volume 58, 2019, Pages 117–126

Proceedings of 34th International Conference on Computers and Their Applications

Customized Intrusion Detection Based on a Database Audit Log

Thomas Le, William Mitchell, and Behnam Arad
California State University, Sacramento
TLe@dir.ca.gov, mitchell@ecs.csus.edu, arad@csus.edu

Abstract

The Internet enables world-wide communication for all areas of human activity. To deal with the massive data involved, companies deploy database products such as Oracle® Database, MySQL, Microsoft® SQL Server, and IBM® DB2. Databases are continuously under attack by intruders who probe for valuable customer and corporate information. Commercial databases have auditing support that facilitates after-the-fact review and analysis of data access. However, audit data collected has vendor-specific structure and content. Tools are needed to optimize response to security incidents and to proactively mine audit logs for vulnerabilities. This paper[1] demonstrates some database-independent techniques aimed toward automating the management of a site's audit information.

Keywords: Database audit, Oracle® database, security, intrusion detection, incident tracking, reporting.

1. Introduction

This paper presents a utility that integrates the built-in Oracle® audit package with an Oracle® task/job scheduler named *DBMS_scheduler* in order to improve the effectiveness of database auditing. The built-in Oracle® auditing process is complex and has some practical limitations. The auditing logs are read-only data dictionary views, and the built-in security restrictions prohibit the use of triggers on those views. The proposed utility automatically extracts and processes targeted log data without affecting the built-in audit at all.

The Oracle® auditing features are very rich and powerful. However, they can be complicated and hard to use. Database administrators (DBAs) must set polices and manually perform audit analysis. Data collected and recorded in Oracle® audit logs are extensive but cannot be used to create custom

[1] This paper is based on Thomas Le's Master of Science Project report. For details, please refer to [1]

security alerts. Audit records are stored in *SYS.AUD$*, *SYS.FGA_LOG$*, and system logs. Subsets of 18 views must be queried. Suspicious activities can be easily overlooked. Moreover, data logs from these views are read-only; therefore, event triggers cannot be used. Oracle® and other databases prohibit the use of triggers on crucial internal tables; in addition, the resources used must be considered. Triggers, custom reports, and stored procedures derived from *SYS.AUD$* and *SYS.FGA_LOG$* can present significant overhead.

```
SQL>
SQL> create or replace trigger test_trigger
  2  after insert
  3  on failed_login_report
  4  declare
  5  count1 integer;
  6  user_name failed_login_report.username%type;
  7  term failed_login_report.terminal%type;
  8  Ora_time   failed_login_report.timestamp%type;
  9  begin
 10          select count(*) into count1,username into user_name,
 11          terminal into term,timestamp into Ora_time
 12          from failed_login_report
 13          where returncode != 0
 14          group by username,terminal;
 15          if count1 >4 then
 16                  dbms_output.put_line('Failed login attempts:'||count1);
 17
 18          end if;
 19  end;
 20  /
create or replace trigger test_trigger
                          *
ERROR at line 1:
ORA-04089: cannot create triggers on objects owned by SYS
```

Figure 1: ORA-04089: cannot *CREATE* triggers on object owned by SYS [1]

An important objective of this work is to be able to efficiently collect and report the audit information from *SYS.AUD$* and *SYS.FGA_LOG$* on adaptive schedules. Another objective is to overcome Oracle®'s restriction on placing triggers on these audit records.

In this paper we use a hypothetical school database schema called *GRADE_DEPOT* to evaluate the proposed utility. It includes tables representing classes that are being taught at a school. Chem301 and Physics101 are two such classes with *Name*, *ID*, and *Grade* as columns.

It is the job of the school's DBA to keep the database and its contents secured and safe. The DBA relies on custom audit reports and security alerts to monitor the database for suspicious login attempts and unauthorized modifications of each database object.

In this work, we developed several custom report logs. The reports are covered in detail in [1]. We provide a summary here. There are two predefined log tables called *SYS.access_report* and *dba_access_report*, intended to store all database login attempts. All data manipulation language (DML) and data definition language (DDL) statements are reported to *SYS.Gen_report* and *dba_gen_report*. Four tables called *dba_grade_depot*, *dba_access_alert*, *dba_grade_log*, and *dba_grade_log2* track all changes made to the grade column of the audit object and the failed login attempts. With these custom reports, the DBA can create custom alert reports related to database security policy. If failed login attempts for a user exceed a threshold, the DBA is alerted. A printed message including the user name, the terminal used, and the number of failed login attempts is generated.

The audit policy includes the monitoring of all logins (successful or otherwise) to the database along with all DML and DDL actions (*CREATE, TABLE*, *DROP*, *INSERT*, *UPDATE*, DELETE, and *ALTER* table). If an *UPDATE* statement is executed on the object, the old value as well as the new value are reported. Furthermore, the exact SQL statements are captured.

The rest of the paper is organized as follows. Section 2 covers some background material on Oracle® database. The proposed utility is described in Section 3. Some test results and analysis are presented in Section 4. Conclusions appear in Section 5.

2. Background

We provide an overview of Oracle® database and its features used in this work. Oracle® Database, also known as Oracle® DBMS, is an object-relational database management system. It is developed and marketed by the Oracle ® Corporation. Please refer to [2] for detailed information.

Oracle® 11g Express Edition (Oracle® Database XE), release 11.2.0.1.0-64 bit, is used for the development of this utility. Oracle® Database 11G-XE is selected as it provides real application testing options, PL/SQL stored procedures, and DML triggers. It also includes the standard Oracle® audit subsystem that is the information base for this utility. Please see [3]-[4] for more details.

It is important to recognize that Oracle® Database XE is a reduced version of Oracle® Database Enterprise Edition. An application that works on Oracle® Database XE, will work on Oracle® Database Enterprise Edition, but not vice versa. The "Enterprise Edition contains all the components of Oracle® Database [4]." It provides "the performance, availability, scalability, and security required for mission-critical applications such as high-volume online transaction processing (OLTP) applications, query-intensive data warehouses, and demanding Internet applications [4]."

Safe and secure database practices such as security configuration of the database, enabling Oracle® data dictionary protection, and enforcing password management are not within the scope of this work. Oracle® Database 2 Day + Security Guide is an excellent reference for Oracle® Database Security [9]. Its Checklist White Paper "provides guidance on configuring the Oracle® Database based on security best practices for operational database deployments [5]."

Oracle® Database Express Edition Licensing Information [8] contains a comprehensive list of features that are available with Oracle® Database XE.

Oracle® 3GL (PL/SQL) is a database programming language that was developed by Oracle®. PL/SQL is an imperative 3GL that was "designed specifically for the seamless processing of SQL commands. It provides specific syntax for this purpose and supports exactly the same datatypes as does Oracle SQL. Server-side PL/SQL is stored and compiled in Oracle® Database and runs within the Oracle® executable. It automatically inherits the robustness, security, and portability of Oracle® Database [6]."

According to Oracle®, "auditing is the monitoring and recording of selected user database actions. [7]". It is used investigate suspicious activity and to monitor and gather data about specific database activities [7].

Oracle® supports three types of auditing: regular audit, fine-grained audit, and common audit (combination of regular and fine-grained audit). Once enabled, Oracle® audit can track actions on a specified schema, its objects, and users' login/logoff information. *Statement audit* audits DDL and DML SQL statements, such as *SELECT*, *CREATE*, *UPDATE*, DELETE, and *DROP*. *Privilege audit* audits the use of the target privilege. Audit records can be stored in either a data dictionary table, called the database audit trail, or an operating system audit trail. The database audit trails for fine-grained and regular audits are reported to *FGA_LOG$* and *SYS.AUD$* tables, respectively. These tables are in the *SYS* schema of each Oracle® database data dictionary.

3. The Proposed Utility

3.1 Design

The automation aspect of the proposed utility is achieved by utilizing Oracle® *DBMS_scheduler*. The utility consists of two schedulers that run independently. While one scheduler gathers data from Oracle®'s pre-defined audit views on a pre-set schedule, the other manipulates the gathered data and generates audit reports and security alerts.

The trigger mechanism is an important component. For example, when the grade column of our student table is audit-enabled, any change in that column will trigger a notification to the DBA. This is because we create triggers on audit data extracted from *AUD$* and *FGA_LOG$*. Our extracted data are updated only when *SYS.AUD$* and *SYS.FGA_LOG$* change.

Oracle® prohibits the creation of triggers on the tables (objects) that are owned by SYS. In addition, it is a good database practice that one performs DBA routines using a separate account with DBA privileges granted by SYS [5]. This utility uses a separate schema with full DBA privileges named *DBA_ADMIN* to monitor the database.

Our system design considers the resources used by the triggers and the copy and update of the audit records, especially for huge audit content. Performance considerations involve efficient trigger execution (with resistance to excessive trigger use), data extraction with well thought out SQL, stored procedures, and cursor processing between the Oracle® audit and our system. Figure 2 outlines the process structure of the proposed system.

3.2 Code

We comment on important aspects of the code developed for the utility. Please refer to [1] for the complete source code.

Audit features need to be enabled as Oracle® audit is disabled by default. Here is how to enable auditing and save audit records to the database audit trail:

```
SQL> ALTER SYSTEM SET audit_trail=db SCOPE=SPFILE;
```

The database needs to be re-started for the change to take effect as follows:

```
SQL> SHUTDOWN immediate
```

Here is how to restart the database instance:

SQL> STARTUP

Once audit is enabled, audit policies (audited actions) need to be specified. In this case, the audit policy calls for *AUDIT CREATE SESSION*, which monitors failed/successful login attempts, as well as DML and DDL statements on schema *GRADE_DEPOT*.

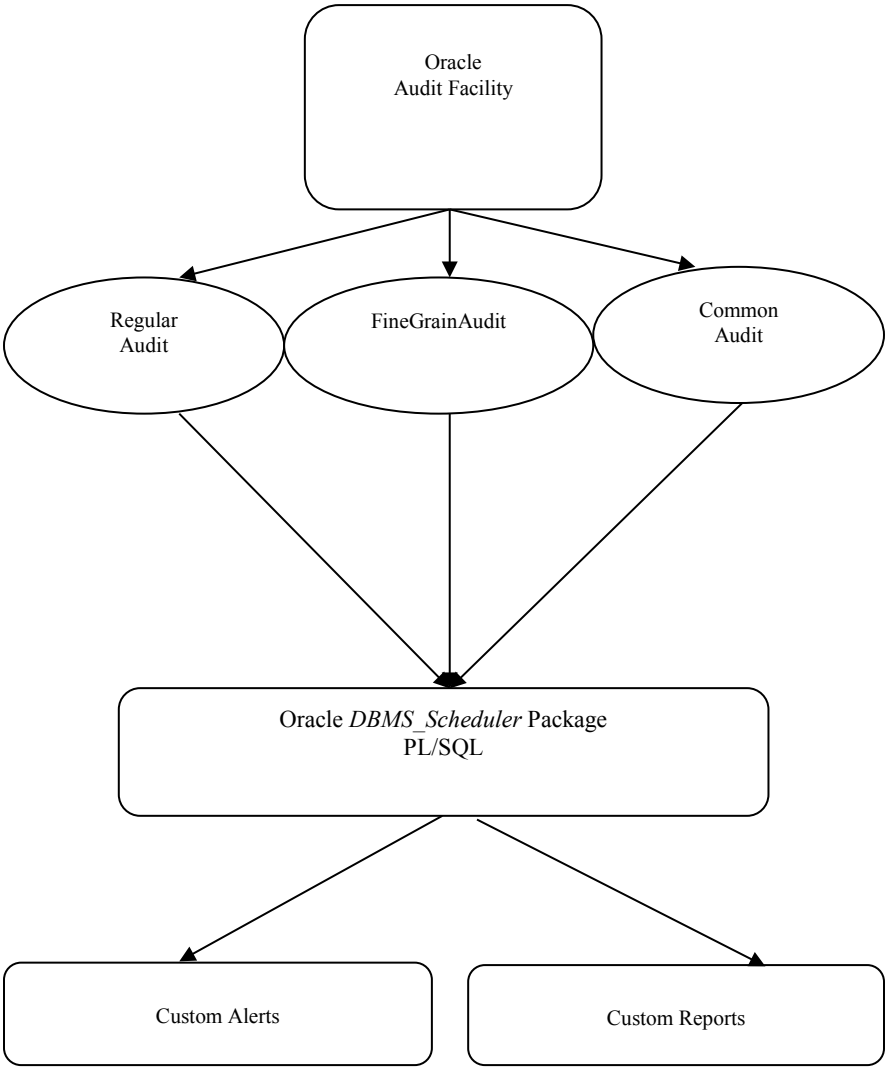

Figure 2: System Process Diagram [1]

Below is the sequence to specify the audit policies once enabled:

```
CONNECT sys/password AS SYSDBA
SQL>AUDIT ALL BY GRADE_DEPOT BY ACCESS;
SQL>AUDIT CREATE SESSION WHENEVER NOT SUCCESSFUL;
SQL>AUDIT CREATE SESSION WHENEVER SUCCESSFUL;SQL>AUDIT SELECT TABLE,
UPDATE TABLE, INSERT TABLE, DELETE TABLE BY GRADE_DEPOT BY ACCESS;
```

For *fine-grained audit*, to audit the grade column, the policy can be set using a PL/SQL built-in package as follows:

```
CONN sys/password AS SYSDBA
SYS>
BEGIN
 DBMS_FGA.add_policy(
   object_schema   => 'GRADE_DEPOT',
   object_name     => 'physics101',
   policy_name     => 'check_grade',
   audit_condition => ' ',
   audit_column    => 'GRADE');
END;
```

The utility uses two schedulers. The first scheduler consisting of *scheduler_api_sys.sql* and *scheduler_run_sys.sql* operates using the SYS schema. The sole purpose of this scheduler is to automatically collect and update the two master custom log tables, *gen_report* and *access_report*. In the *DBA_ADMIN* schema, the second scheduler consisting of *scheduler_api_dba.sql* and *scheduler_run_dba.sql* generates specific jobs and programs according to the audit policy.

Some of these jobs and programs generate and update the predefined custom log tables, while others manipulate the data and print custom security reports. In addition, in the *DBA_ADMIN* schema, a series of triggers are set on these custom log tables.

It is worth mentioning that these custom logs have many table columns occurring in the 18 pre-defined views in Oracle® audit package. *DBA_ACCESS_ALERT* holds all failed login information. *DBA_GEN_REPORT* contains all DDL and DML statements performed on the audit schema objects, and so on.
Oracle® defines a trigger as "a named PL/SQL unit that is stored in the database and executed (fired) in response to a specified event that occurs in the database [10]." Therefore, data access or other audited action is required for the triggers to be defined on these tables. Explicit cursors are used to fetch into temporary tables in schema DBA_ADMIN the data on which triggers are created.

The schedulers used in this work are implemented with built-in Oracle package *dbms_schedule*r. This package enables creation and execution of jobs that extract oracle audit content into the privileged schema *DBA_ADMIN*. Extraction is accomplished by executing standard SQL user-defined cursors for row-at-a-time extraction. The jobs are usually database stored procedures, but in the Oracle case,

could also be other kinds of modules. The additional two schedulers generate the jobs that create reports and alerts using DML SQL triggers.

Our proposed scheduling/extraction approach forms a database-independent audit post-processor. However, database systems vary considerably regarding the details of scheduling flexibility, extraction options, data storage organization and access method support. Database dependencies arise at this level of detail.

For example, an intense database workload, thus high auditing rate, might result in audit data extraction becoming an operational bottleneck. As pointed in Section 3.1, this would limit scalability of our approach. To deal with this, a given database system might or might not provide advanced cursor capabilities such as bulk (not row-at-a-time) extraction, shared cursors, parallelizability of the SQL engine, and so on. Moreover, SQL execution efficiency and scalability ultimately depend on the SQL query optimizer. Specifically, the ability of the optimizer to quickly determine the least-cost execution plan for SQL statements is database system dependent. We note that relational database vendor SQL query optimizer capabilities vary considerably.

3.3 Test

The utility was tested on Oracle® Database version 11g Express Edition. We performed incremental life cycle and unit testing. The tests were based on a hypothetical schema called *GRADE_DEPOT* including two objects, CHEM301 and PHYSICS101. Each one has three columns namely *NAME*, *Id*, and *GRADE*. Our verification of the proposed utility covered the test cases listed in Table 1.

Table 1: Test cases for validating the Proposed Utility

Verification Test Cases
Independent operation of both schedulers
Correct data collection from Oracle audit views
DBA_ADMIN schema's scheduler executing all the jobs and programs as scheduled
Validity of information from the custom log table, gen_report
UPDATE, DELETE, and DROP TABLE commands
All the custom log tables being updated correctly
Proper functionality of Event triggers to alert the database administrator of suspicious activity
Correct generation of security alert and audit report when failed login attempts exceed a threshold
Functionality of DML statement audit alert and reporting

Complete test results are provided in [1] (Figures 5 – 15). Here, we provide two of the figures illustrating our testing. Figure 3 illustrates the audited actions. Figure 4 shows the exact information from the custom log table, gen_report.

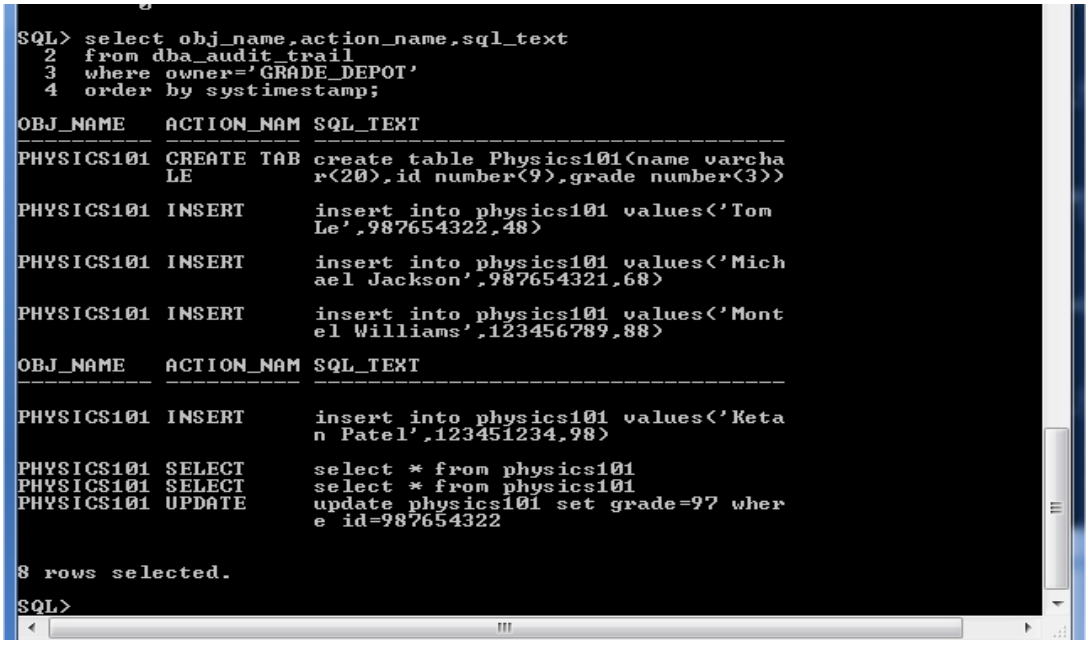

Figure 3: Oracle's dba_audit_trail view showing audited actions [1]

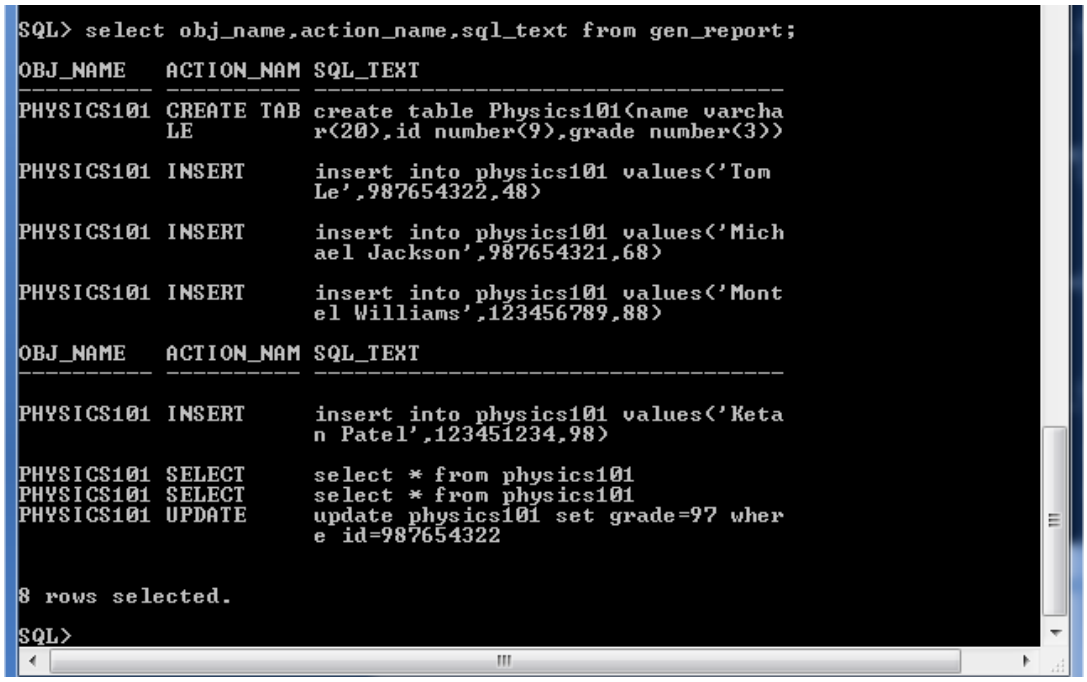

Figure 4: Custom log table, gen_report, showing the same audited actions [1]

4. Results and Analysis

The test results show that the utility effectively monitors the database. Working independently of one another, both schedulers in the utility work as expected. The scheduler in *SYS* schema correctly collects the information from the corresponding Oracle® audit, *AUD$*. We verified that the correct data are collected from Oracle® audit views. *DBA_ADMIN* schema's scheduler also executes all the jobs and programs as scheduled. All the custom log tables are updated with their respective data types. Event triggers that alert the database administrator of suspicious activity are working properly. They fire when conditions in the specific tables met the preset security policy. Test results show that the correct security alert and audit report are generated once the failed login attempts exceed the threshold of two failed attempts. DML statement audit alert and report also work.

This paper does not include performance analysis (via benchmarks) of audit log data extraction. Alerts can be implemented in near real time using triggers as database audit record inserts occur. Alternatively, if triggers are not deployed, alerts can be deferred until stored procedures are executed on extracted audit. Choosing between triggers and stored procedures depends on how quickly a site wishes to generate alerts as well as site security policy requirements.

Experience with very large databases having significant data modification rates suggests that triggers should be used with caution [11]. This applies especially for systems with significant concurrent table processing since triggers add overhead to transactional data changes. However, our use of triggers occurs in only one security administration process associated with audit extraction. Thus, concurrency is not a cost factor for our configuration of one centralized database and its audit.

There remains the question of the relative speed of the trigger approach. Testing on VMware virtual machines running Redhat Centos version 6 with identical Oracle 11g release 2.0.1.0 (4 GB memory, Intel(R) Xeon(R) CPU, E5620 @ 2.40GHz, 12 MB cache size (as reported from /proc/cpuinfo)) yields typical results. For example audit extractions of 55,500 rows and 918,843 rows, respectively, the triggered executions took more than 3 times the elapsed time taken by stored procedures using row-at-a-time cursors. In both cursor scenarios, the additional overhead of creating tables used by the stored procedure was included. This table creation step contributes increasingly diminished cost for increasing table size compared with stored procedure execution time on extracted audit...

Although the above elapsed time comparisons do not address scalability for increasing audit extraction sizes, they confirm that generating alerts via triggers has acceptable, albeit it, significant overhead. However, a site with high audit rate would require database-specific optimizations when the trigger approach become a performance bottleneck. In relational databases, speed improvements depend on support for SQL bulk (not row-at-a-time) row insert, SQL statement parallelization, query optimization sophistication, and available physical storage options such as multi-level table and index partitioning. The tradeoff is between using triggers and priority of timely alert creation.

We note in passing that commercial grade database audit toolkits are gradually becoming available [2]. As is the case with many management aspects of full-featured database systems, the audit component of a system install is usually treated as a black box [13]. However, we have illustrated (for Oracle database) how to build a customized audit enhancement while leaving the native audit subsystem untouched.

5. Conclusions

Audit logging is commonly used in database systems for monitoring security. However, even for relational databases, audit content is not standardized among databases. In addition, individual sites using the same database product usually need customized audit functionality. This paper outlines a process architecture for customized auditing that has some measure of database independence. The approach uses task and job scheduling that accesses and manipulates a given database's audit log. Such tasks deal with policy compliance, reporting, log mining/analysis, and security incident handling. The level of database independence supported by our task structure depends on isolating database-specific dependencies and task modularity achievable on a given database system. This paper demonstrates reporting and incident tracking of specified audited actions by ordinary Oracle® database users. Customization is applied to audit data sets extracted from the actual database audit log. This approach leaves the internal Oracle® audit log untouched.

References

[1] Thomas Le, Customized Intrusion Detection Based on a Database Audit Log, MS Project Report, California State University, Sacramento 2014.
[2] Oracle® Database Concepts 10g Release 2 [Online].
Available:http://docs.Oracle®.com/cd/B19306_01/server.102/b14220/intro.htm
[3] Oracle® Database 12c White Paper (2013, June)
[Online].Available:http://www.Oracle®.com/technetwork/database/Oracle®-database-editions-wp-12c-1896124.pdf?ssSourceSiteId=ocomen
[4] Oracle® Database Licensing Information 11g Release 2 [Online]. Available:
http://docs.Oracle®.com/cd/E11882_01/license.112/e47877/editions.htm#DBLIC109
[5] Oracle® Database Security Checklist White Paper (2008, June) [Online]. Available:
http://www.Oracle®.com/technetwork/database/security/twp-security-checklist-database-1-132870.pdf
[6] Oracle® Database 12c [Online]. Available:
http://www.Oracle®.com/technetwork/database/features/plsql/index.html
[7] Oracle® 9i Database Concepts Release 2 [Online].
Available:http://docs.Oracle®.com/cd/B10500_01/server.920/a96524/c25audit.htm 66
[8] Oracle® Database Express Edition 11g [Online].
Available:http://docs.Oracle®.com/cd/E17781_01/license.112/e18068.pdf
[9] Oracle® Database 2 day + Security Guide [Online].
Available:http://docs.Oracle®.com/cd/B28359_01/server.111/b28337.pdf
[10] Oracle® Database PL/SQL Language Reference 11g
Release[Online].Available:http://docs.Oracle®.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS020
[11] Tom Kyte, 'The Trouble with Triggers', , Oracle Corporation, Oracle Magazine, Sept/Oct 2008
[2] P. Finnigan, 'Delete from AUD$, P. Finnigan's Oracle Security Weblog, February 2017
[13] 'db2audit – Audit facility administrator tool command', IBM Db2 11.1,
https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0002072.html