CSC12001

Security Issues in Information Systems

C6 – Auditing

Dr. Phạm Thị Bạch Huệ
MSc. Lương Vĩ Minh

Information System Department – Faculty of Information Technology
Univeristy of Science, VNU-HCM

Outline

1. Security cycle
2. Why to audit
3. Audit methods

Introduction

- Security cycle
- Goal of auditing

Security Cycle

- Prevention → Detection → Response
  - Access control is a way of prevention.
  - Detection means finding the security breach.
  - Response relies on the detection.

Auditing for Accountability

- People use auditing to provide the detection capabilities just described to support overall database security efforts.
- You want to ensure users are doing only what they are supposed to. You want to capture privilege abuse and misuse. Auditing allows you to hold your users responsible for their actions by tracking their behavior.
- When a person commits a crime, a picture that has captured them in the act can serve as very compelling evidence.
- An important aspect to auditing is that it can also serve as a deterrent for would-be bad guys. If you know that someone is watching, you are less likely to do something bad.
- Database auditing can be thought of as the security cameras that capture the actions and diabolical deeds as they unfold. Note that the cameras and auditing may capture both good or expected actions as well as the bad and unexpected actions.

- Security begins with a clear and concise security policy.
- The policy is enforced through proper design and implementation complemented with varying access control capabilities.
- There is no way to build a computer application that is 100 percent secure.
- You have to rely on auditing for the detection mechanisms to support the overall database security

Auditing Provides the Feedback Loop

- One thing worse than being robbed is not even knowing that you've been robbed. Auditing can help. The audit records are the means of capturing the robbery. If you're viewing the records and you see something has happened, then you can properly respond. Response may result in readjusting your access control mechanisms and expelling the user.
- Two important things have to happen for effective auditing.
  - First, you have to be auditing and auditing on the correct thing. This is analogous to saying you have to have security cameras turned on and facing the right direction.
  - Second, you have to read and interpret the audit records.
- The audit records act as a feedback mechanism into the prevention and access control mechanisms you've already established. They also play an important role in any investigative activities that occur either as a result of a breach or in anticipation of one.
- Without auditing, you may have no way of knowing whether your security is sound or whether your data has been read or modified by an unauthorized user.

Auditing Is Not Overhead

- Some people feel that auditing introduces overhead which, however, compare to the valuable information derived from the audit records, it's worth doing it.
- Auditing needs to be performed to determine what users can see and do. It is necessary to ensure the privacy and security of the data.
- However, auditing all actions by all users on all data is not useful and *will* make a system perform miserably.
- Auditing must be selective and, when done correctly, it targets the correct data, processes, and users. This means the audit records are, by definition, very useful.
- Auditing is simply the last phase in your security cycle, and it should not be discarded.

Audit Methods

- Application server logs.
- Application auditing.
- Trigger auditing.
- Standard auditing.
- Fine-grained auditing.
- These auditing techniques aren’t mutually exclusive. They can and should, as necessary, be combined to build complementary layers of defense within the auditing realm.

Application Server Logs

- Application server access logs + all associated log files for the application server and web server are often considered a basic form of auditing.
  - they will list the resources that have been accessed, when and how the access occurred, and the result by way of a status code—for example, success, failure, and unknown.
- The logs are very useful. The records contained in the log files are often direct indicators of the actions the user performed. For example, an update posted from a web page would have a distinct URL signature. As such, the user (or rather the user's IP address) can be audited as having invoked some program.

- Application server logs are very useful in determining suspicious behavior.
  - Denial of service (DoS) attacks.
  - Many administrators actually use the logs to track a user's behavior as they navigate a website. This is similar to studying the shopping patterns of customers in department stores.
- The challenge with using the application log files is that the information is indirect. It's only useful when combined with other data that links IP addresses to users and the URLs with actual programs. For this reason, application auditing is usually performed in addition to gathering server log files because it can directly audit who is acting on what.

Trigger setting IP Address to Client Identifier

```
sec_mgr@KNOX10g> CREATE OR REPLACE TRIGGER set_ip_in_id
  2    AFTER LOGON ON DATABASE
  3  BEGIN
  4    DBMS_SESSION.set_identifier
  5                          (SYS_CONTEXT ('userenv',
  6                                        'ip_address'));
  7  END;
  8  /
```

Application Auditing

- Application auditing is the auditing *manually programmed* into the application.
- One of the most frequently used auditing techniques.
- Regardless of the implementation language, application auditing is a natural to use, because it can meet most auditing requirements.
- Many people are quite familiar with application auditing. This technique is often seen when the developers don't understand or can't take advantage of the database auditing.
- As users perform actions, the application code selectively audits. Various aspects of auditing are generally seen. User logins, data manipulations, and administration tasks can be easily audited.

Application auditing

- The program could be invoked from an application running either within or outside the database.
- This program will be explicitly called by the application at the appropriate time, which will vary from application to application. The example will invoke this auditing when the user performs an update to the SAL column of our table.

```
scott@KNOX10g> CREATE TABLE aud_emp (
  2    username    VARCHAR2(30),
  3    action      VARCHAR2(6),
  4    empno       NUMBER(4),
  5    column_name VARCHAR2(255),
  6    call_stack  VARCHAR2(4000),
  7    client_id   VARCHAR2(255),
  8    old_value   VARCHAR2(10),
  9    new_value   VARCHAR2(10),
 10    action_date DATE DEFAULT SYSDATE
 11  )
```

```
scott@KNOX10g> CREATE OR REPLACE PROCEDURE audit_emp (
  2    p_username      IN  VARCHAR2,
  3    p_action        IN  VARCHAR2,
  4    p_empno         IN  NUMBER,
  5    p_column_name   IN  VARCHAR2,
  6    p_old_value     IN  VARCHAR2,
  7    p_new_value     IN  VARCHAR2)
  8  AS
  9  BEGIN
 10  -- check data format and length
 11  -- not shown here
 12    INSERT INTO aud_emp
 13                (username,
 14                 action,
 15                 empno,
 16                 column_name,
 17                 call_stack,
 18                 client_id,
 19                 old_value,
 20                 new_value,
 21                 action_date)
 22         VALUES (p_username,
 23                 p_action,
 24                 p_empno,
 25                 p_column_name,
 26                 DBMS_UTILITY.format_call_stack,
 27                 SYS_CONTEXT ('userenv',
 28                              'client_identifier'),
 29                 p_old_value,
 30                 p_new_value,
 31                 SYSDATE);
 32  END;
 33  /
```

```
scott@KNOX10g> CREATE OR REPLACE PROCEDURE update_sal (
  2    p_empno   IN  NUMBER,
  3    p_salary  IN  NUMBER)
  4  AS
  5    l_old_sal  VARCHAR2 (10);
  6  BEGIN
  7    SELECT     sal
  8          INTO l_old_sal
  9          FROM emp_copy
 10         WHERE empno = p_empno
 11    FOR UPDATE;
 12    UPDATE emp_copy
 13       SET sal = p_salary
 14     WHERE empno = p_empno;
 15    audit_emp
 16      (p_username       => USER,
 17       p_action         => 'UPDATE',
 18       p_empno          => p_empno,

 19       p_column_name    => 'SAL',
 20       p_old_value      => l_old_sal,
 21       p_new_value      => p_salary);
 22  END;
 23  /
```

```
scott@KNOX10g> CREATE OR REPLACE PROCEDURE show_aud_emp
  2  AS
  3  BEGIN
  4    FOR rec IN (SELECT   *
  5                  FROM aud_emp
  6                ORDER BY action_date DESC)
  7    LOOP
  8      DBMS_OUTPUT.put_line (   'User:       '
  9                            || rec.username);
 10      DBMS_OUTPUT.put_line (   'Client ID:  '
 11                            || rec.client_id);
 12      DBMS_OUTPUT.put_line (   'Action:     '
 13                            || rec.action);
 14      DBMS_OUTPUT.put_line (   'Empno:      '
 15                            || rec.empno);
 16      DBMS_OUTPUT.put_line (   'Column:     '
 17                            || rec.column_name);
 18      DBMS_OUTPUT.put_line (   'Old Value:  '
 19                            || rec.old_value);
 20      DBMS_OUTPUT.put_line (   'New Value:  '
 21                            || rec.new_value);
 22      DBMS_OUTPUT.put_line (   'Date:       '
 23                            || TO_CHAR
 24                                 (rec.action_date,
 25                                  'Mon-DD-YY HH24:MI'));
 26      DBMS_OUTPUT.put_line
 27                     ('------------------------------');
 28    END LOOP;
 29  END;
```

```
scott@KNOX10g> GRANT EXECUTE ON update_sal TO blake;

Grant succeeded.

scott@KNOX10g> GRANT SELECT ON emp_copy TO blake;

Grant succeeded.
blake@KNOX10g> SELECT empno, sal
  2    FROM scott.emp_copy
  3   WHERE ename = 'BLAKE';

     EMPNO        SAL
---------- ----------
      7698       2850

blake@KNOX10g> EXEC scott.update_sal(p_empno=>7698, p_salary=>3000);

PL/SQL procedure successfully completed.

blake@KNOX10g> COMMIT ;

Commit complete.

blake@KNOX10g> SELECT empno, sal
  2    FROM scott.emp_copy
  3   WHERE ename = 'BLAKE';

     EMPNO        SAL
---------- ----------
      7698       3000
```

```
scott@KNOX10g> EXEC show_aud_emp
User:        BLAKE
Client ID:   192.168.0.100
Action:      UPDATE
Empno:       7698
Column:      SAL
Old Value:   2850
New Value:   3000
Date:        Mar-24-04 13:34
------------------------------
```

Application Auditing

- Benefits:
  - Can be extended easily.
  - Application auditing can often be modified to meet the new and ever-changing requirements.
  - Can control *how* the auditing is done: to audit to a file on the midtier or audit to a separate database, which would protect the audit records from the administrators of the production database.
  - All aspects of the application can be audited, not only the data access. If the application interfaces with multiple databases, a few flat files, and a web service, all of that can be audited in a consistent way.
  - Application auditing requires no knowledge of database auditing.

Application Auditing

- Drawbacks:
  - Challenges of coding.
  - From the security angle, the real drawback occurs if the application is bypassed.

Trigger Auditing

- Using DML triggers (for inserts updates, deletes) for aditing is a very popular technique for auditing.
- Trigger auditing provides transparency, allowing you to enable auditing without requiring application modifications. Applications don’t have to be aware of the trigger auditing.

- Step 1: Create an auxiliary auditing table.
- Step 2: Create the trigger to insert data (what user to do the operation, new and old data for update operation, for example) to the previous table.

- For testing, perform the operation on the table, then display the records in auditing table.

```
Scott: CREATE OR REPLACE TRIGGER update_emp_sal_trig
 BEFORE UPDATE OF sal
 ON emp_copy
 FOR EACH ROW
 DECLARE
 BEGIN
  audit_emp (p_username       => USER,
               p_action         => 'UPDATE',
              p_empno          => :OLD.empno,
             p_column_name    => 'SAL',
            p_old_value      => TO_CHAR (:OLD.sal),
            p_new_value      => TO_CHAR (:NEW.sal));
 END;
```

```
blake@KNOX10g> UPDATE scott.emp_copy
  2     SET sal = sal * 1.1
  3   WHERE deptno = 20;

Scott: EXEC show_aud_emp;

User:       BLAKE
Client ID:  192.168.0.100
Action:     UPDATE
Empno:      7369
Column:     SAL
Old Value:  800
New Value:  880
Date:       Mar-24-04 14:23
```

Trigger Auditing

- **Benefits**

➢ Trigger auditing is transparent to the application.

➢ An application in which the code can't be modified, then trigger auditing may provide a robust mechanism for adding or augmenting what is already provided.

➢ The triggers also can be enabled only when specific columns are being manipulated, as seen in the previous example.

➢ The trigger can operate for each row or for each statement. This allows selectivity in auditing and reduces the number of unnecessary audit records.

➢ The trigger auditing will also be invoked for all applications regardless of language; that is, no matter how the user interacts with the data, the trigger will audit.

Trigger Auditing

- **Drawbacks**

➢ Triggers don't fire for certain actions, such as TRUNCATE statements.

➢ Triggers don't allow applications to pass additional parameters. They are constrained to the table columns. Outside of the new and old values of the data, the only other information the trigger can use are application contexts and environmental data, such as the user's name and connecting IP address.

➢ Also, just like application auditing, triggers have to be created and defined for every object. Calling procedures from within triggers can help if the procedures can be shared across several triggers.

Standard Database Auditing

- Mandatory Auditing
- Administrator (SYS) Auditing.
- Standard Auditing
- Fine-grained Auditing.

Madatory Auditing

- The database always records three important things:
  - database startup
  - database shutdown
  - users authenticated with the SYSDBA or SYSOPER roles
- For the database startup, the audit record also indicates whether standard auditing has been enabled. This allows one to determine if an administrator has disabled auditing (setting the AUDIT_TRAIL value to none or FALSE) and is now restarting the database.

- Audit records:
  - have to be stored on the operating system because the database isn't available (it's being started or stopped).
  - The actual location varies depending on OS platform—for example, on Windows, the records are written to the Event Logs; on Linux, the audits are generally found in the $ORACLE_HOME/rdbms/audit directory.

Auditing SYS

- Auditing actions performed by users authenticated as SYSDBA or SYSOPER.
- The audit records are again written to OS files.
  - First, these users have the most significant privileges in the database, such as the ability to see and modify all data, change passwords, log in to any schema, and drop schemas. As such, it's generally advisable to monitor their actions.
  - Second, they control the database auditing. If they want to disable it, they have the privileges to do so. They also have the privileges to delete the audit records. Therefore, auditing to the database is useless since the user would be able to modify or delete the audit records.
- You shouldn't allow the database user access to the operating system directories where the audits will be written, or else you will suffer from the same challenge as auditing to the database.

Auditing SYS

- To enable audits for the SYS user, you have to set the AUDIT_SYS_OPERATIONS initialization parameter to TRUE:

```
System: ALTER SYSTEM SET audit_sys_operations=TRUE
     SCOPE=SPFILE;

System altered.
```

Auditing SYS

- This change is written to the initialization file (init.ora), and the database has to be rebooted for the change to take effect.
- Trying to change this parameter at runtime results in the error: “ORA-02095: specified initialization parameter cannot be modified”.
  - This is a security feature. If the database could be modified at runtime without booting, the SYS user could turn off auditing, do something bad, and then re-enable auditing.
  - By forcing a reboot, you know that you’ll have captured both the disabling of the auditing as well as the reboot.

Standard Auditing in Oralce

- Oracle Database records audit activities in audit records. Audit records provide information about:
  - The operation that was audited,
  - The user performing the operation,
  - The date and time of the operation.
- Audit records can be stored in either a data dictionary table, called the **database audit trail**, or in operating system files, called an **operating system audit trail**.
- Oracle Database also provides a set of data dictionary views that you can use to track suspicious activities.
- Oracle Database writes the audit records to either to DBA_AUDIT_TRAIL (the SYS.AUD$ table), the operating system audit trail, or to the DBA_COMMON_AUDIT_TRAIL view, which combines standard and fine-grained audit log records.

Standard Auditing

- For invoking the database standard auditing, you have to enable it by setting the AUDIT_TRAIL initialization parameter.
  - The AUDIT_TRAIL parameter allows standard database auditing to occur. By default, this parameter is also set to FALSE.
- AUDIT_TRAIL parameter can receive the following values:
  - **"OS"**: which enable the database audit records to be written to the operating system (OS).
    - As with auditing on SYS, this is a good idea if you're concerned with someone modifying or deleting the audit records that might otherwise be contained within the database.
    - Auditing to the OS can make it difficult to run reports because the data is in a text file.
  - **"DB" or "TRUE":** enable auditing for records stored in the database
    - In the SYS.AUD$ table.
  - **"DB_EXTENDED":** enables database auditing but captures additional information in the audit record: SQL text, some other useful information, …

Standard Auditing

- You can audit:
  - On objects such as tables and views: every time someone accesses the APP.USER_DATA table an audit will be recorded.
  - Procedure executions.
  - When someone exercises a system privilege, such as disabling a trigger or using the SELECT ANY TABLE privilege.
- You can restrict your audits to specific users.
- You can audit for successful actions, unsuccessful actions, or both.
- With all of the preceding, you can audit every time someone performs the action, or audit only once per session regardless of the number of times they perform the action within the session.

Auditing Practices

- Checking for the value of the AUDIT_TRAIL parameter:

  System: show parameter audit_trail;
- For auditing for logons and logoffs, simply audit the user's connection

  system: AUDIT SESSION;
- Auditing whenever Unsuccessful: sensitive data or with data that can be used to derive sensitive information, such as privacy-related data, encryption keys, passwords, and user preferences.
  - Ex: audit selects on T for failures

  sec_mgr: AUDIT SELECT ON T BY ACCESS WHENEVER NOT SUCCESSFUL;
- Auditing on the SCOTT.EMP object, for both successful and unsuccessful actions as well as auditing by access so you can capture a record for every SQL statement:

  sec_mgr: AUDIT SELECT ON scott.emp BY ACCESS;

- Deleting the existing records from the audit table:

  Sys: DELETE FROM aud$;

- Disable auditing (useful if you are following along), you have to issue the NOAUDIT command:

  Sys: NOAUDIT SELECT ON scott.emp;.

-

FGA (Fine-grained Auditing)

- Can be applied to all DML statements.
- To be an extension to the standard auditing.
- The auditing can be more selectively enabled to occur only when certain conditions arise and specific columns are queried.
- FGA offeres four major advantages over standard auditing: a Boolean condition check, the SQL capture ability, a column-sensitivity feature, and an event handler.

FGA

- Conditional auditing helps to eliminate unnecessary audits.
  - In standard auditing, you could audit anytime someone queried a table. However, you couldn't specify any exemptions to this based on specific conditions.
  - In standard auditing, by auditing SELECT statements, the audit records would only indicate a user selected from the table and would not indicate whether they were able to access another user's records (although you can derive the result set by looking at the SQL captured).
- With FGA, you can audit only when a user is accessing someone else's records.

- FGA doesn’t rely on any initialization parameters. Instead, you control FGA via the DBMS_FGA package.
- ADD-POLICY procedure and specifying the condition as seen next. If you leave the condition NULL, then the audit will always occur.

- Policy: audit SCOTT.EMP, but audit only when a user is accessing another user's records.

```
sec_mgr: BEGIN
      DBMS_FGA.add_policy
                  (object_schema      => 'SCOTT',
                   object_name        => 'EMP',
                   policy_name        => 'Example',
                   audit_condition    => 'ENAME != USER');
   END;
```

- Strange results from FGA can occur if you don't first issue an ANALYZE on the table. Prior to running this, the current audit records were deleted and the standard auditing on the EMP table was disabled:

```
scott@KNOX10g> ANALYZE TABLE emp COMPUTE STATISTICS;
scott@KNOX10g> SELECT sal, comm FROM emp
  WHERE ename = 'SCOTT';


scott@KNOX10g> SELECT sal, comm FROM emp
  WHERE deptno = 20;
```

```
sec_mgr@KNOX10g> EXEC show_aud

PL/SQL procedure successfully completed.

sec_mgr@KNOX10g> EXEC show_aud
Who:  SCOTT
What: SCOTT.EMP
Where: 192.168.0.100:SQLPLUS.EXE
When:  Mar-24 18:34
How:   SELECT sal, comm FROM emp
 WHERE deptno = 20
```

Column Sensitivity

- Users are allowed to access other users' records, except the salary field within those records.
- If the user's query doesn't touch the SAL column, then you don't audit.

Column Sensitivity

- The actual auditing will occur when both the sensitive column(s) is queried and the Boolean condition is met. In the case that no condition is specified, the auditing will occur only when the sensitive column is queried or manipulated.

```
sec_mgr: BEGIN
DBMS_FGA.drop_policy (object_schema    => 'SCOTT',
                      object_name      => 'EMP',
                      policy_name      => 'Example');
   DBMS_FGA.add_policy
               (object_schema      => 'SCOTT',
                object_name        => 'EMP',
                policy_name        => 'Example',
                audit_condition    => 'ENAME != USER',
               audit_column       => 'SAL');
END;
```

Key points

- We can audit:
  - Audit logon, logoff into the database.
  - Audit source of database usage.
  - Audit database usage outside normal operating hours.
  - Audit DDL activity.
  - Audit database errors.
  - Audit changes to sources of stored procedures and triggers.
  - Audit changes to privileges, user/login definitions, and orther security attributes.
  - Audit creations, changes, and usage of database links and replication.
  - Audit change to sensitive data.
  - Audit SELECT statements for privacy sets.
  - Audit any changes made to the definition of what to audit.

Q & A

Dr. Phạm Thị Bạch Huệ - ptbhue@fit.hcmus.edu.vn

MSc. Lương Vĩ Minh – lvminh@fit.hcmus.edu.vn