Practice-Using SQL Firewall

Practice Target

This practice guides you in using SQL Firewall in Oracle Database 23ai to protect SQL workloads. You will configure users and roles, simulate trusted and untrusted workloads, capture allowed SQL, enforce block mode, test violations, and review logs.

Practice Overview

In this practice, you will perform the following:

- Enable SQL Firewall
- Generate and enforce allow-list
- Test violations with unexpected SQL
- Switch allow-list to block mode and observe behavior
- Review violation and audit logs

Demonstrating Using SQL Firewall

1. Open SQL Developer and connect to the pdb as SYSTEM or SYS, then enable the SQL Firewall.

```
EXEC DBMS_SQL_FIREWALL.ENABLE;
```

2. Start a capture for **HR** user.

```
EXEC DBMS_SQL_FIREWALL.CREATE_CAPTURE('HR');
```

3. Open Putty, invoke SQL*Plus, and connect as HR and execute known safe queries and procedures.

```
sql hr/oracle@//localhost/freepdb1
SELECT first_name, last_name FROM hr.employees WHERE department_id = 90; SELECT job_id, job_title FROM
hr.jobs;
SELECT d.department_name, e.last_name FROM hr.departments d JOIN hr.employees e ON d.manager_id =
e.employee_id;
```

4. In the SYSTEM session, stop the capture.

```
EXEC DBMS_SQL_FIREWALL.STOP_CAPTURE('HR');
```

5. View session and capture logs.

   Observe that internal SQL queries were logged.

```
SELECT username, ip_address FROM dba_sql_firewall_session_logs WHERE username='HR';
SELECT sql_text FROM dba_sql_firewall_capture_logs WHERE username='HR';
```

6. Generate allow list from captured SQL and enable it.

```
EXEC DBMS_SQL_FIREWALL.GENERATE_ALLOW_LIST('HR'); EXEC
DBMS_SQL_FIREWALL.ENABLE_ALLOW_LIST('HR');
```

7. In Putty session, re-login as HR and run the previously allowed statements.
   All should execute without violations.

```
conn hr/oracle@//localhost/freepdb1

SELECT first_name, last_name FROM hr.employees WHERE department_id = 90; SELECT job_id, job_title FROM
hr.jobs;
```

```
SELECT d.department_name, e.last_name FROM hr.departments d JOIN hr.employees e ON d.manager_id =
e.employee_id;
```

8. Run unexpected queries to trigger firewall violations.

   It should trigger a violation (but not be blocked yet).

```
SELECT * FROM hr.employees ORDER BY hire_date;
```

9. In the SYSTEM session, flush and query violation logs.

   The violated query should be retrieved.

```
EXEC DBMS_SQL_FIREWALL.FLUSH_LOGS;
SELECT cause, firewall_action, sql_text FROM dba_sql_firewall_violations WHERE username='HR';
```

10. Update the allow list enforcement to block mode.

```
exec DBMS_SQL_FIREWALL.UPDATE_ALLOW_LIST_ENFORCEMENT('HR', block => TRUE);
```

11. As HR, retry unmatched statements and observe blocked results.

    The query was blocked without waiting for the session to re-login.

```
SELECT * FROM employees ORDER BY hire_date;
```

12. As a cleanup, disable and delete the allow-list, drop the HR capture, and disable the firewall.

```
EXEC DBMS_SQL_FIREWALL.DISABLE_ALLOW_LIST('HR'); EXEC
DBMS_SQL_FIREWALL.DROP_ALLOW_LIST('HR'); EXEC
DBMS_SQL_FIREWALL.DROP_CAPTURE('HR');
EXEC DBMS_SQL_FIREWALL.DISABLE;

-- as HR, run the unmatched statements again and make sure it is not blocked: SELECT * FROM employees ORDER BY
hire_date;
```

Summary

You have completed a full basic SQL Firewall lifecycle in Oracle database. You learned how to simulate a workload, enforce known SQL behavior, capture violations, and escalate to block mode.