



ActiveBase Security Rule Example Package

Example scenarios for AB*Security

ActiveBase Security (AB*Security) rule sample include examples that can be easily customized to fulfill your needs:

- Auditing Sensitive and Personal Information (SPI),
 - Blocking of offensive requests,
 - Hiding columns values
 - Data masking
 - Scrambling
 - Dynamically restricting Results
 - Identify and control offensive requests
 - Control Toad Users
- ◆ All examples use simple SQL requests into the Scott/tiger schema for testing AB's security capabilities. Table descriptions can be found in Appendix B
 - ◆ Each scenario is presented in two stages – 1st stage without using AB*Security and 2nd stage with AB*Security.

Notes:

1) All of the rules described bellow can be easily added to your existing rules by saving the "ABSecurity_Example_Pack.xml" attached to this document and simply importing it into your routing action.

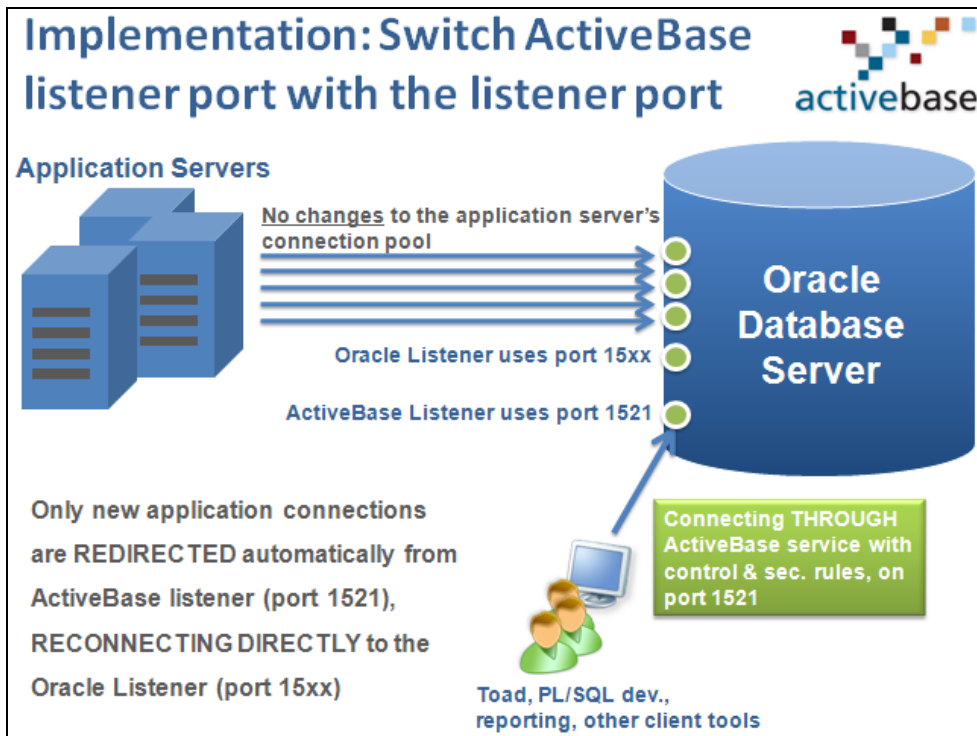
2) Scott schema SQL creation script can be found under:

```
SCOTT.SQL SCRIPT FROM \ORACLE_HOME\RDBMS\ADMIN\SCOTT.SQL
```

Simply open SQL*Plus and execute the SCOTT.SQL file to your database. Table description can be found in Appendix B.

Prerequisites

1. Define several oracle users. AB*Security rule sample include capability to allow or block access based on Oracle user names and other v\$session column data.
2. Create the PL/SQL functions found in the appendix A
3. Download and install AB*Security
4. Switch Oracle listener to port 15XX
5. Configure ActiveBase Listener port to 1521



6. Run the following scenarios:

1. Audit

- The 1st rule found in the “ABSecurity_Example_Pack.xml” is called “Log All”, ensuring that every SQL request from the clients that touch SPI data is audited. AB*Security enables to categorize these SQL requests into
 - DML requests (SQL requests that change data in an object) or
 - DDL requests (SQL requests that create, change or remove a database object).
 DML and DDL commands that include sensitive objects will be audited.
- These audited SQL requests are saved to a cyclic log files for uploading into an auditing database. NOTE: ActiveBase provides a utility for uploading the log files.

Action: run any query and show that it was audited and the log file captures it.

In SQL*Plus:

```

Command Prompt - sqlplus system/o
SQL> select sal from emp;

   SAL
-----
   800
  1600
  1250
  2975
  1250
  2850
  2450
  3000
  5000
  1500
  1100

   SAL
-----
   950
  3000
  1300

14 rows selected.

SQL>

```

ActiveBase Logs the following information:

1. **Time stamp:** 01/18 16:01:54,296
2. **Group of users:** DBA_Team
3. **Database name:** WProd
4. **Rule name:** Log All
5. **SQL full text:** `select sal from emp`
6. **Client Info:**[User=alon, Host=MERCURY,
application=C:\app\alon\product\11.1.0\db_1\bin\sqlplus.exe]
7. **SessionID**=139,92
8. **Oracle user name:** system
9. **Instance number:** 1

2. Prevent Data Leakage

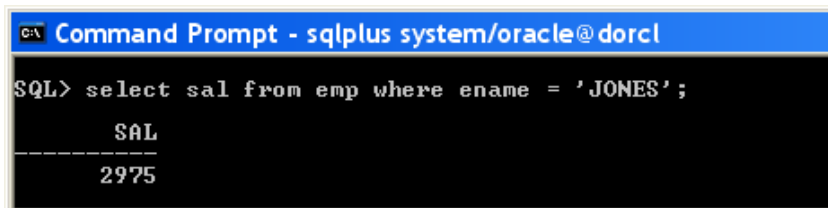
★ Apply a Mask to a column from the Result Set.

- **Salary masking rule.**

Action: Run the statement “`select sal salary from scott.emp where ename = 'JONES'`”.

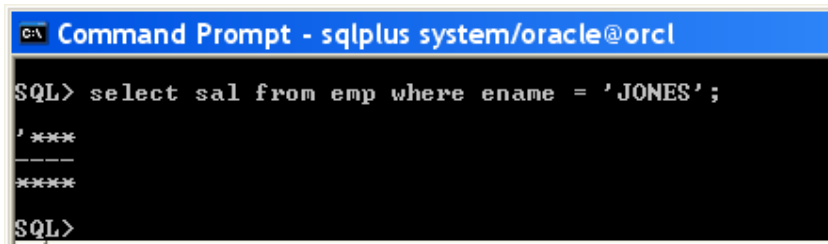
The salary column will be masked with “***”.

Before:



```
C:\ Command Prompt - sqlplus system/oracle@dorcl
SQL> select sal from emp where ename = 'JONES';
      SAL
-----
      2975
```

With AB*Security:



```
C:\ Command Prompt - sqlplus system/oracle@orcl
SQL> select sal from emp where ename = 'JONES';
' ***
-----
*****
SQL>
```

As seen in Toad.exe program:

ActiveBase can use any function required for masking or replacing SPI by the customer. In the following example, a rule was created to replace the first character with a '_' followed by the three middle characters only.

Action: Run the statement "select ename employee, sal salary from scott.emp".

The ENAME column will be scrambled.

Before:

```
C:\> Command Prompt - sqlplus system/oracle@orcl
SQL> select ename employee, sal salary from scott.emp;
EMPLOYEE          SALARY
-----
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
```

With AB*Security:

```
C:\> Command Prompt - sqlplus system/oracle@orcl
SQL> select ename employee, sal salary from scott.emp;
EMPL SALA
-----
MIT ****
LLE ****
ARD ****
ONE ****
ART ****
LAK ****
LAR ****
COT ****
ING ****
URN ****
DAM ****
```

★ Dynamically restrict the Result of a query according to contextual or user properties.

- Check user with a PL/SQL function and block accordingly.

Action: Run any CREATE/DROP/ALTER/etc. statements with the relevant oracle user and see it is being blocked.

- Check OSuser with a PL/SQL function and block accordingly.

The *Check_OSuser* PL/SQL function verifies that the OS_User name is 'SAM' and blocks all DML commands accordingly.

To define more restrictions for the users identified by the PL/SQL function, simply add more blocking rule inside the 'Check User' folder in the rule sample.

Action: Create the PL/SQL function when changing the OS_User name variable in the PL/SQL function to match your OS_User name (as found in your session by running 'select osuser from v\$session').

After the function is created, run any CREATE/DROP/ALTER/etc. as they will be blocked before reaching the database, while returning a message to the user accordingly.

You can notify the user by adding your own message in the blocking rule text field. In the rule set example, the following message is provided to the user:

```
Command Prompt - sqlplus system/oracle@orcl
SQL> CREATE TABLE DEPT
 2      (DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
 3      DNAME VARCHAR2(14) ,
 4      LOC VARCHAR2(13) ) ;
CREATE TABLE DEPT
*
ERROR at line 1:
ORA-00900: You are not allowed to run DDL commands. Please contact DBA team.
```

3. Identify and control offensive requests

- Alert the security officer (SMS & email) and Block the request.

Action: Run any statement matched by the rule and verify the message is emailed and the statement is blocked. ActiveBase sends an email using a PL/SQL function that connects to the database and send an email to the relevant distribution list with a detailed message. The email PL/SQL function ab_send_mail is described in the appendix A.

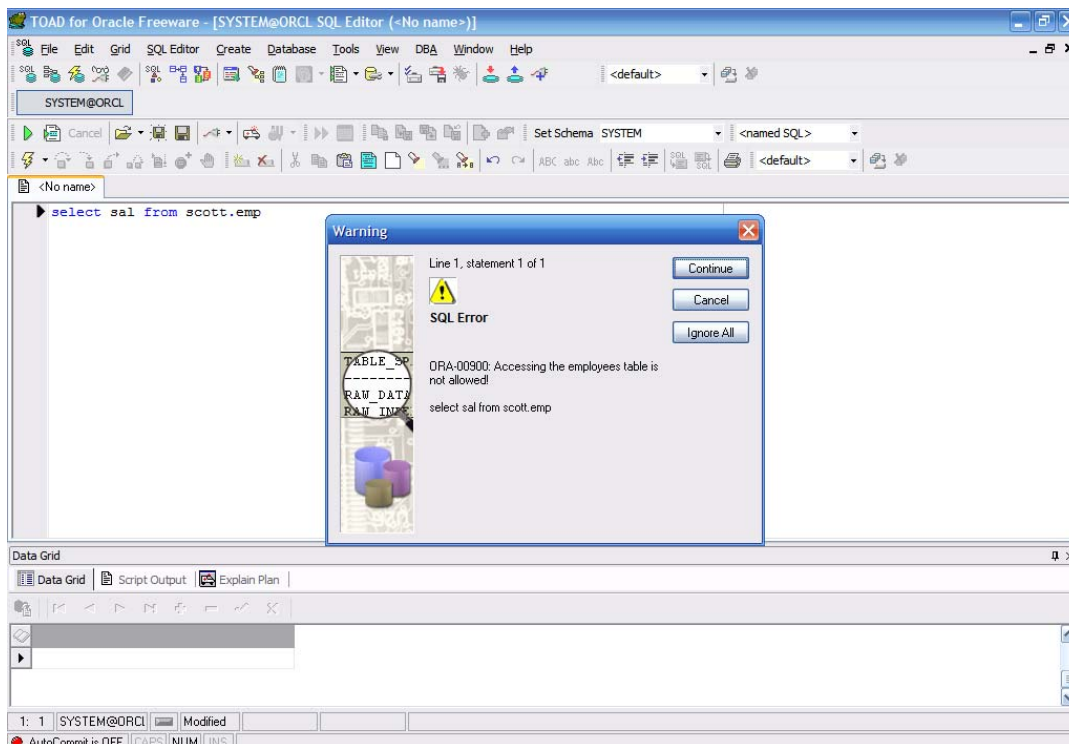
4. Control Toad Users

Identifying toad users can be done according to the toad.exe program (possibly creating a specific routing action to these users), or based on toad module name, as specified by toad using dbms_application_info settings (which cannot be changed by the user). Identifying toad module name setting example is found in the rules and in the PL/SQL function: check_toad. Import the rules to the relevant location.

- Block Toad from accessing / modifying specific tables or database objects

- Block Employees table access rule.

Action: run any statement from Toad which involves 'emp' table (for example- "select from scott.emp"). The user will get the relevant error.



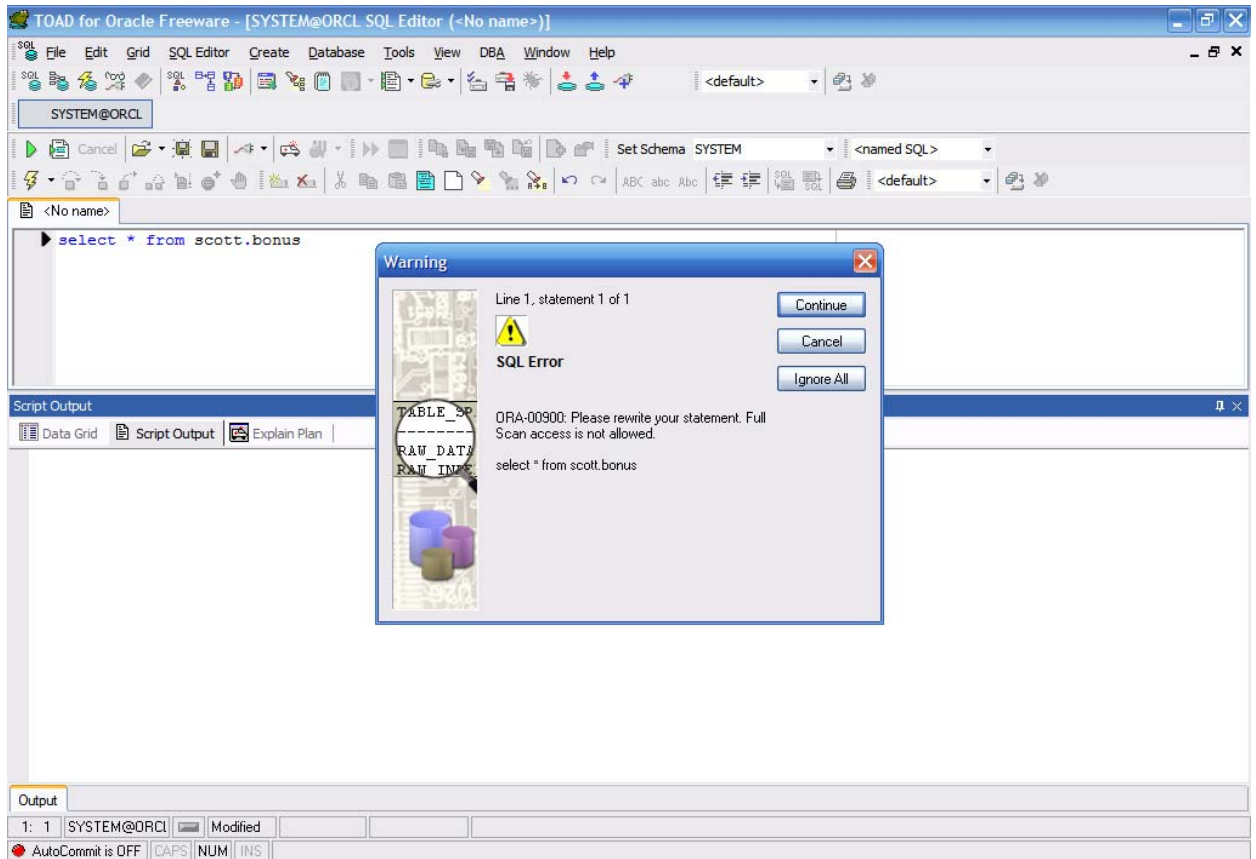
- Block Toad from escalating user privileges on tables.

- DCL commands control rule.

Action: Run any grant or revoke statement and verify it's being blocked.

- Block Toad according to time of day rule.
- Block Toad from running long queries by using the "Full Scan control" rule.

Action: run any statement which involves full scan (for example- any "select * from..." statement) and verify it is being blocked.



Appendix A: PL/SQL functions

1. PL/SQL Function Check_OSUser

Purpose: identifying specific users for more in-depth auditing and proactive user session control such as blocking DDL/DML commands.

```
CREATE OR REPLACE FUNCTION check_OSUser(pSid varchar2) return integer as
  vOSUser varchar2(100);
Begin
  select osuser into vOSUser from v$session where sid = pSid;
  if upper(vOSUser) like 'SAM' then
    return 1;
  end if;
  return 0;
End;
```

Note:

1. In some cases, system user cannot run a PL/SQL that 'select from v\$session'. To enable you need to run the following:

```
conn sys as sysdba
grant select on v_$session to system;
```

2. 'Keep matcher result' attribute has been selected. This rule matcher attribute defines that the matcher (PL/SQL function in this example) is NOT to be check for every incoming SQL statement, but only ONCE EVERY 3,600 seconds (every hour) per session. After the matcher is checked, the result will be kept valid for all concurrent SQL requests executed in the session, preventing the execution of the identification PL/SQL function only once per hour for each session.

2. PL/SQL Function Check_toad

Purpose: Toad users cannot be identified by merely the toad.exe program, because users might change the program name. This PL/SQL function identifies toad using the Module name, which cannot be changed by the user.

```
CREATE OR REPLACE FUNCTION check_toad(pSid varchar2) return integer as
  vModule varchar2(100);
Begin
  select module into vModule from v$session where sid = pSid;
  if upper(vModule) like 'TOAD%' then
    return 1;
  end if;
  return 0;
End;
```

Note:

In some cases, system user cannot run a PL/SQL that 'select from v\$session'. To enable you need to run the following:

```
conn sys as sysdba
grant select on v_$session to system;
```

3. PL/SQL Function ab_send_mail

Purpose: general purpose send email function that enables ActiveBase to send email alerts to administrators.

```
CREATE OR REPLACE FUNCTION ab_send_mail(p_oracle_user varchar2, p_os_user
varchar2, p_host_name varchar2, p_program varchar2, p_stmt varchar2, p_subject
varchar2)
return number
```

IS

```
Result NUMBER :=0;
```

begin

```
utl_mail.send (
activesecurity,
security@active-base.com,
null,
null,
p_subject,
details oracle user = p_oracle_user os user = p_os_user host name = p_host program
name = p_program statement text = p_stmt,
'text/plain; charset=us-ascii',
null);
result :=1;
return(result);
end ab_send_mail;
/
```

Note:

The function will receive subject text and all SQL parameters and will send an email notification accordingly
UTL_MAIL is not installed by default because of the SMTP_OUT_SERVER configuration, you must both install UTL_MAIL and define the SMTP_OUT_SERVER.

To install UTL_MAIL, run the following in your SQL*Plus session:

```
sqlplus sys/<pwd>
```

```
SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql
```

```
SQL> @$ORACLE_HOME/rdbms/admin/prvtmail.sql
```

Set smtp_server information in init.ora or spfile.ora alter system set smtp_out_server = 'SMTP_SERVER_IP_ADDRESS:SMTP_PORT' scope=both; 25 = Default SMTP Port

Appendix B: Table descriptions

CREATE TABLE dept

```
(deptno number(2) constraint pk_dept primary key,  
  dname varchar2(14) ,  
  loc varchar2(13) ) ;
```

CREATE TABLE emp

```
(empno number(4) constraint pk_emp primary key,  
  ename varchar2(10),  
  job varchar2(9),  
  mgr number(4),  
  hiredate date,  
  sal number(7,2),  
  comm number(7,2),  
  deptno number(2) constraint fk_deptno references dept);
```

```
INSERT INTO DEPT VALUES
```

```
(10,'ACCOUNTING','NEW YORK');
```

```
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
```

```
INSERT INTO DEPT VALUES
```

```
(30,'SALES','CHICAGO');
```

```
INSERT INTO DEPT VALUES
```

```
(40,'OPERATIONS','BOSTON');
```

```
INSERT INTO EMP VALUES
```

```
(7369,'SMITH','CLERK',7902,TO_DATE('17-12-1980','DD-MM-YYYY'),800,NULL,20);
```

```
INSERT INTO EMP VALUES
```

```
(7499,'ALLEN','SALESMAN',7698,TO_DATE('20-2-1981','DD-MM-YYYY'),1600,300,30);
```

```
INSERT INTO EMP VALUES
```

```
(7521,'WARD','SALESMAN',7698,TO_DATE('22-2-1981','DD-MM-YYYY'),1250,500,30);
```

```
INSERT INTO EMP VALUES
```

```
(7566,'JONES','MANAGER',7839,TO_DATE('2-4-1981','DD-MM-YYYY'),2975,NULL,20);
```

```
INSERT INTO EMP VALUES
```

```
(7654,'MARTIN','SALESMAN',7698,TO_DATE('28-9-1981','DD-MM-YYYY'),1250,1400,30);
```

```
INSERT INTO EMP VALUES
```

```
(7698,'BLAKE','MANAGER',7839,TO_DATE('1-5-1981','DD-MM-YYYY'),2850,NULL,30);
```

```
INSERT INTO EMP VALUES
```

```
(7782,'CLARK','MANAGER',7839,TO_DATE('9-6-1981','DD-MM-YYYY'),2450,NULL,10);
```

```
INSERT INTO EMP VALUES
```

```
(7788,'SCOTT','ANALYST',7566,TO_DATE('13-JUL-87')-85,3000,NULL,20);
```

```
INSERT INTO EMP VALUES
```

```
(7839,'KING','PRESIDENT',NULL,TO_DATE('17-11-1981','DD-MM-YYYY'),5000,NULL,10);
```

```
INSERT INTO EMP VALUES
```

```
(7844,'TURNER','SALESMAN',7698,TO_DATE('8-9-1981','DD-MM-YYYY'),1500,0,30);
```