12

Managing Security for Application Developers

A security policy for application developers should encompass areas such as password management and securing external procedures and application privileges.

- About Application Security Policies
  An application security policy is a list of application security requirements and rules that regulate user access to database objects.
- Considerations for Using Application-Based Security
  An application security implementation should consider both application and database users and whether to enforce security in the application or in the database.
- Use of the DB_DEVELOPER_ROLE Role for Application Developers
  The `DB_DEVELOPER_ROLE` role provides most of the system privileges, object privileges, predefined roles, PL/SQL package privileges, and tracing privileges that an application developer needs.
- Securing Passwords in Application Design
  Oracle provides strategies for securely invoking password services, such as from scripts, and for applying these strategies to other sensitive data.
- Securing External Procedures
  An external procedure is stored in a `.dll` or an `.so` file, separately from the database, and can be through a credential authentication.
- Securing LOBs with LOB Locator Signatures
  You can secure large objects (LOB) by regenerating their LOB locator signatures.
- Managing Application Privileges
  Most database applications involve different privileges on different schema objects.
- Advantages of Using Roles to Manage Application Privileges
  Grouping application privileges in a role aids privilege management.
- Creating Secure Application Roles to Control Access to Applications
  A secure application role is only enabled through its associated PL/SQL package or procedure.
- Association of Privileges with User Database Roles
  Ensure that users have only the privileges associated with the current database role.
- Protecting Database Objects by Using Schemas
  A schema is a security domain that can contain database objects. Privileges granted to users and roles control access to these database objects.
- Object Privileges in an Application
  When you design an application, consider the types of users and the level access they need.
- Parameters for Enhanced Security of Database Communication
  Parameters can be used to manage security, such as handling bad packets from protocol errors or configuring the maximum number of authentication errors.

12.1 About Application Security Policies

An application security policy is a list of application security requirements and rules that regulate user access to database objects.

Creating an application security policy is the first step to create a secure database application. You should draft security policies for each database application. For example, each database application should have one or more database roles that provide different levels of security when executing the application. You then can grant the database roles to other roles or directly to specific users.

Applications that can potentially allow unrestricted SQL statement processing (through tools such as SQL*Plus or SQL Developer) also need security policies that prevent malicious access to confidential or important schema objects. In particular, you must ensure that your applications handle passwords in a secure manner.

12.2 Considerations for Using Application-Based Security

An application security implementation should consider both application and database users and whether to enforce security in the application or in the database.

- Are Application Users Also Database Users?
  Where possible, build applications in which application users are database users, so that you can use the intrinsic security mechanisms of the database.
- Is Security Better Enforced in the Application or in the Database?
  Oracle recommends that applications use the security enforcement mechanisms of the database as much as possible.

12.2.1 Are Application Users Also Database Users?

Where possible, build applications in which application users are database users, so that you can use the intrinsic security mechanisms of the database.

For many commercial packaged applications, application users are not database users. For these applications, multiple users authenticate themselves to the application, and the application then connects to the database as a single, highly-privileged user. This is called the *One Big Application User* model.

Applications built in this way generally cannot use many of the intrinsic security features of the database, because the identity of the user is not known to the database. However, you can use client identifiers to perform some types of tracking. For example, the `OCI_ATTR_CLIENT_IDENTIFIER` attribute of the Oracle Call Interface method `OCIAttrSet` can be used to enable auditing and monitoring of client connections. Client identifiers can be used to gather audit trail data on individual Web users, apply rules that restrict data access by Web users, and monitor and trace applications that each Web user users.

Table 12-1 describes how the One Big Application User model affects various Oracle Database security features:

Table 12-1 Features Affected by the One Big Application User Model

Oracle Database Feature	Limitations of One Big Application User Model
Auditing	A basic principle of security is accountability through auditing. If One Big Application User performs all actions in the database, then database auditing cannot hold individual users accountable for their actions. The application must implement its own auditing mechanisms to capture individual user actions.
Oracle strong authentication	Strong forms of authentication (such as client authentication over SSL, tokens, and so on) cannot be used if the client authenticating to the database is the application, rather than an individual user.
Roles	Roles are assigned to database users. Enterprise roles are assigned to enterprise users who, though not created in the database, are known to the database. If application users are not database users, then the usefulness of roles is diminished. Applications must then craft their own mechanisms to distinguish between the privileges which various application users need to access data within the application.
Enterprise user management	The Enterprise user management feature enables an Oracle database to use the Oracle Identity Management Infrastructure by securely storing and managing user information and authorizations in an LDAP-based directory such as Oracle Internet Directory. While enterprise users do not need to be created in the database, they do need to be known to the database. The One Big Application User model cannot take advantage of Oracle Identity Management.

12.2.2 Is Security Better Enforced in the Application or in the Database?

Oracle recommends that applications use the security enforcement mechanisms of the database as much as possible.

Applications, whose users are also database users, can either build security into the application, or rely on intrinsic database security mechanisms such as granular privileges, virtual private databases (fine-grained access control with application context), roles, stored procedures, and auditing (including fine-grained auditing).

When security is enforced in the database itself, rather than in the application, it cannot be bypassed. The main shortcoming of application-based security is that security is bypassed if the user bypasses the application to access data. For example, a user who has SQL*Plus access to the database can run queries without going through the Human Resources application. The user, therefore, bypasses all of the security measures in the application.

Applications that use the One Big Application User model must build security enforcement into the application rather than use database security mechanisms. Because it is the application, and not the database, that recognizes users; the application itself must enforce security measures for each user.

This approach means that each application that accesses data must re-implement security. Security becomes expensive, because organizations must implement the same security policies in multiple applications, and each new application requires an expensive reimplementation.

Related Topics

- Potential Security Problems of Using Ad Hoc Tools
  Ad hoc tools can pose problems if malicious users have access to such tools.

12.3 Use of the DB_DEVELOPER_ROLE Role for Application Developers

The `DB_DEVELOPER_ROLE` role provides most of the system privileges, object privileges, predefined roles, PL/SQL package privileges, and tracing privileges that an application developer needs.

An application developer needs a large number of these privileges to design, develop, and deploy applications. Oracle recommends that you grant the application developer the `DB_DEVELOPER_ROLE` role, rather than individually granting these privileges or granting the user the `DBA` role. Granting the application user the `DB_DEVELOPER_ROLE` role not only adheres to least-privilege principles and ensures greater security for the development environment, it facilitates the management of role grants and revokes for application developers. The `DB_DEVELOPER_ROLE` role can be used in either the CDB root or the PDB. Do not modify the `DB_DEVELOPER_ROLE`.

Generating a List of Privileges and Roles Granted by the DB_DEVELOPER_ROLE Role

To generate a full list of the system privileges, object privileges, and roles that are granted by the `DB_DEVELOPER_ROLE`, run the following statement. Ensure that you include the `set serveroutput on format wrapped` command, so that the indentation will be shown properly.

> **Note:**
>
> Be aware that the output will vary, depending on the version or patch release of Oracle Database that you are using.

```
set serveroutput on format wrapped;
DECLARE
    procedure printRolePrivileges(
      p_role             in varchar2,
      p_spaces_to_indent in number) IS
      v_child_roles   DBMS_SQL.VARCHAR2_TABLE;
      v_system_privs  DBMS_SQL.VARCHAR2_TABLE;
      v_table_privs   DBMS_SQL.VARCHAR2_TABLE;
      v_indent_spaces varchar2(2048);
    BEGIN
      -- Indentation for nested privileges via granted roles.
      for space in 1..p_spaces_to_indent LOOP
        v_indent_spaces := v_indent_spaces || ' ';
      end LOOP;
      -- Get the system privileges granted to p_role
      select PRIVILEGE bulk collect into v_system_privs
      from DBA_SYS_PRIVS
      where GRANTEE = p_role
      order by PRIVILEGE;
```

```
      -- Print the system privileges granted to p_role
      for privind in 1..v_system_privs.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(
          v_indent_spaces || 'System priv: ' || v_system_privs(privind));
      END LOOP;

      -- Get the object privileges granted to p_role
      select PRIVILEGE || ' ' || OWNER || '.' || TABLE_NAME
        bulk collect into v_table_privs
      from DBA_TAB_PRIVS
      where GRANTEE = p_role
      order by TABLE_NAME asc;

      -- Print the object privileges granted to p_role
      for tabprivind in 1..v_table_privs.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(
          v_indent_spaces || 'Object priv: ' || v_table_privs(tabprivind));
      END LOOP;

      -- get all roles granted to p_role
      select GRANTED_ROLE bulk collect into v_child_roles
      from DBA_ROLE_PRIVS
      where GRANTEE = p_role
      order by GRANTED_ROLE asc;

      -- Print all roles granted to p_role and handle child roles recursively.
      for roleind in 1..v_child_roles.COUNT LOOP
        -- Print child role
        DBMS_OUTPUT.PUT_LINE(
         v_indent_spaces || 'Role priv: ' || v_child_roles(roleind));

        -- Print privileges for the child role recursively. Pass 2 additional
        -- spaces to illustrate these privileges belong to a child role.
        printRolePrivileges(v_child_roles(roleind), p_spaces_to_indent + 2);
      END LOOP;

      EXCEPTION
        when OTHERS then
          DBMS_OUTPUT.PUT_LINE('Got exception: ' || SQLERRM );

    END printRolePrivileges;

BEGIN
    printRolePrivileges('DB_DEVELOPER_ROLE', 0);
END;
/
```

Output similar to the following appears:

```
System priv: CREATE ANALYTIC VIEW
System priv: CREATE ATTRIBUTE DIMENSION
System priv: CREATE CUBE
System priv: CREATE CUBE BUILD PROCESS
System priv: CREATE CUBE DIMENSION
System priv: CREATE DIMENSION
```

```
System priv: CREATE DOMAIN
System priv: CREATE HIERARCHY
System priv: CREATE JOB
System priv: CREATE MATERIALIZED VIEW
System priv: CREATE MINING MODEL
System priv: CREATE MLE
System priv: CREATE PROCEDURE
System priv: CREATE SEQUENCE
System priv: CREATE SESSION
System priv: CREATE SYNONYM
System priv: CREATE TABLE
System priv: CREATE TRIGGER
System priv: CREATE TYPE
System priv: CREATE VIEW
System priv: DEBUG CONNECT SESSION
System priv: EXECUTE DYNAMIC MLE
System priv: FORCE TRANSACTION
System priv: ON COMMIT REFRESH
Object priv: SELECT SYS.DBA_PENDING_TRANSACTIONS
Object priv: EXECUTE SYS.JAVASCRIPT
Object priv: READ SYS.V_$PARAMETER
Object priv: READ SYS.V_$STATNAME
Role priv: CTXAPP
  System priv: CREATE SEQUENCE
  Object priv: EXECUTE CTXSYS.CTX_ANL
  Object priv: EXECUTE CTXSYS.CTX_DDL
  Object priv: EXECUTE CTXSYS.CTX_ENTITY
  Object priv: EXECUTE CTXSYS.CTX_OUTPUT
  Object priv: EXECUTE CTXSYS.CTX_THES
  Object priv: EXECUTE CTXSYS.CTX_ULEXER
  Object priv: INSERT CTXSYS.DR$DICTIONARY
  Object priv: DELETE CTXSYS.DR$DICTIONARY
  Object priv: SELECT CTXSYS.DR$DICTIONARY
  Object priv: UPDATE CTXSYS.DR$DICTIONARY
  Object priv: INSERT CTXSYS.DR$THS
  Object priv: INSERT CTXSYS.DR$THS_BT
  Object priv: INSERT CTXSYS.DR$THS_FPHRASE
  Object priv: UPDATE CTXSYS.DR$THS_PHRASE
  Object priv: INSERT CTXSYS.DR$THS_PHRASE
  Object priv: EXECUTE CTXSYS.DRIENTL
  Object priv: EXECUTE CTXSYS.DRITHSL
Role priv: SODA_APP
  Object priv: EXECUTE XDB.DBMS_SODA_ADMIN
  Object priv: EXECUTE XDB.DBMS_SODA_USER_ADMIN
  Object priv: READ XDB.JSON$USER_COLLECTION_METADATA
```

Performing Grants and Revokes of the DB_DEVELOPER_ROLE Role

To grant the `DB_DEVELOPER_ROLE` to another user or role, use the `GRANT` statement, as you would with any role grant. For example:

```
GRANT DB_DEVELOPER_ROLE TO pfitch;
```

To check the grant:

```
SELECT GRANTED_ROLE FROM DBA_ROLE_PRIVS WHERE GRANTEE='pfitch';
```

Revoking the `DB_DEVELOPER_ROLE` is similar:

```
REVOKE DB_DEVELOPER_ROLE FROM pfitch;
```

12.4 Securing Passwords in Application Design

Oracle provides strategies for securely invoking password services, such as from scripts, and for applying these strategies to other sensitive data.

- General Guidelines for Securing Passwords in Applications
  Guidelines for securing passwords in applications cover areas such as platform-specific security threats.
- Use of an External Password Store to Secure Passwords
  You can store password credentials for connecting to a database by using a client-side Oracle wallet.
- Securing Passwords Using the ORAPWD Utility
  `SYSDBA` or `SYSOPER` users can use password files to connect to an application over a network.
- Example: Java Code for Reading Passwords
  You can create Java packages that can be used to read passwords.

12.4.1 General Guidelines for Securing Passwords in Applications

Guidelines for securing passwords in applications cover areas such as platform-specific security threats.

- Platform-Specific Security Threats
  You should be aware of potential security threats, which may not be obvious.
- Guidelines for Designing Applications to Handle Password Input
  Oracle provides guidelines for designing applications to handle password input.
- Guidelines for Configuring Password Formats and Behavior
  Oracle Database provides guidelines for configuring password formats and behavior.
- Guidelines for Handling Passwords in SQL Scripts
  Oracle provides guidelines for handling passwords in SQL scripts.

12.4.1.1 Platform-Specific Security Threats

You should be aware of potential security threats, which may not be obvious.

These security threats are as follows:

- **On UNIX and Linux platforms, command parameters are available for viewing by all operating system users on the same host computer.** As a result, passwords entered on the command line could be exposed to other users. However, do not assume that non-UNIX and Linux platforms are safe from this threat.

- **On some UNIX platforms, such as HP Tru64 and IBM AIX, environment variables for all processes are available for viewing by all operating system users.** However, do not assume that non-UNIX and Linux platforms are safe from this threat.
- **On Microsoft Windows, the command recall feature (the Up arrow) remembers user input across command invocations.** For example, if you use the `CONNECT SYSTEM/password` notation in SQL*Plus, exit, and then press the Up arrow to repeat the `CONNECT` command, the command recall feature reveals the connect string and displays the password. In addition, do not assume that non-Microsoft Windows platforms are safe from this threat.

12.4.1.2 Guidelines for Designing Applications to Handle Password Input

Oracle provides guidelines for designing applications to handle password input.

- **Design applications to interactively prompt for passwords.** For command-line utilities, do not force users to expose passwords at a command prompt.

  Check the APIs for the programming language you use to design applications (such as Java) for the best way to handle passwords from users.
- **Protect your database against code injection attacks.** Code injection attacks most commonly occur in the client application tool that sends SQL to the database (for example, SQL*Plus, or any Oracle Call Interface (OCI) or JDBC application. This includes database drivers that are built using these tools. A SQL injection attack causes SQL statements to behave in a manner that is not intended by the PL/SQL application. The injection attack takes place before the statement is sent to the database. For example, an intruder can bypass password authentication by setting a `WHERE` clause to `TRUE`.

  To address the problem of SQL injection attacks, use bind variable arguments or create validation checks. If you cannot use bind variables, then consider using the `DBMS_ASSERT` PL/SQL package to validate the properties of input values. You also should review any grants to roles such as `PUBLIC`.

  Note that client applications users may not always associate SQL injection with PL/SQL, because the injection could occur before the statement is sent to the database.
- **If possible, design your applications to defer authentication.** For example:
  - Use certificates for logins.
  - Authenticate users by using facilities provided by the operating system. For example, applications on Microsoft Windows can use domain authentication.
- **Mask or encrypt passwords.** If you must store passwords, then mask or encrypt them. For example, you can mask passwords in log files and encrypt passwords in recovery files.
- **Authenticate each connection.** For example, if schema A exists in database 1, then do not assume that schema A in database 2 is the same user. Similarly, the local operating system user `psmith` is not necessarily the same person as remote user `psmith`.
- **Do not store clear text passwords in files or repositories.** Storing passwords in files increases the risk of an intruder accessing them.
- **Use a single main password.** For example:
  - You can grant a single database user proxy authentication to act as other database users. In this case, only a single database password is needed.
  - Using the Oracle Database Enterprise User Security Wallet Manager, you can create a password wallet, which can be opened by the main password. The wallet then contains the other passwords.

> **Note:**
>
> Enterprise User Security (EUS) is deprecated with Oracle Database 23ai. Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

Related Topics

- Example: Java Code for Reading Passwords
  You can create Java packages that can be used to read passwords.
- *Oracle Database PL/SQL Language Reference*
- Proxy User Accounts and the Authorization of Users to Connect Through Them
  The `CREATE USER` statement enables you to create the several types of user accounts, all of which can be used as proxy accounts.
- *Oracle Database Enterprise User Security Administrator's Guide*

12.4.1.3 Guidelines for Configuring Password Formats and Behavior

Oracle Database provides guidelines for configuring password formats and behavior.

- **Limit the lifetime for passwords.** Use the `PASSWORD_LIFE_TIME`, `PASSWORD_GRACE_TIME`, and `PASSWORD_ROLLOVER_TIME` profile parameters to control lifetime of passwords.
- **Limit the ability of users to reuse old passwords.** Forcing users to create new, unique passwords can greatly deter intruders from guessing their passwords. You can control these factors by setting the `PASSWORD_REUSE_TIME`, `PASSWORD_REUSE_MAX`, and `PASSWORD_VERIFY_FUNCTION` parameters.
- **Force users to create strong, secure passwords.** You can customize password requirements for your site by using password complexity verification, which forces users to follow Oracle's guidelines for creating strong passwords.
- **Enable case sensitivity in passwords.** By default, new passwords are case sensitive.

Related Topics

- About Controlling Password Aging and Expiration
  You can specify a password lifetime, after which the password expires.
- Controlling the User Ability to Reuse Previous Passwords
  You can ensure that users do not reuse previous passwords for an amount of time or for a number of password changes.
- Guidelines for Securing Passwords
  Oracle provides guidelines for securing passwords in a variety of situations.
- About Password Complexity Verification
  Complexity verification checks that each password is complex enough to protect against intruders who try to guess user passwords.
- Managing Password Case Sensitivity
  You can manage the password case sensitivity for passwords from user accounts from previous releases.

12.4.1.4 Guidelines for Handling Passwords in SQL Scripts

Oracle provides guidelines for handling passwords in SQL scripts.

- **Do not invoke SQL*Plus with a password on the command line, either in programs or scripts.** If a password is required but omitted, SQL*Plus prompts the user for it and then automatically disables the echo feature so that the password is not displayed.

  The following examples are secure because passwords are not exposed on the command line. Oracle Database also automatically encrypts these passwords over the network.

  ```
  $ sqlplus system
  Enter password: password

  SQL> CONNECT SYSTEM
  Enter password: password
  ```

  The following example exposes the password to other operating system users:

  ```
  sqlplus system/password
  ```

  The next example poses two security risks. First, it exposes the password to other users who may be watching over your shoulder. Second, on some platforms, such as Microsoft Windows, it makes the password vulnerable to a command line recall attack.

  ```
  $ sqlplus /nolog
  SQL> CONNECT SYSTEM/password
  ```

- **For SQL scripts that require passwords or secret keys, for example, to create an account or to log in as an account, do not use positional parameters, such as substitution variables &1, &2, and so on.** Instead, design the script to prompt the user for the value. You should also disable the echo feature, which displays output from a script or if you are using spool mode. To disable the echo feature, use the following setting:

  ```
  SET ECHO OFF
  ```

  A good practice is to ensure that the script makes the purpose of the value clear. For example, it should be clear whether or not the value will establish a new value, such as an account or a certificate, or if the value will authenticate, such as logging in to an existing account.

  The following example is secure because it prevents users from invoking the script in a manner that poses security risks: It does not echo the password; it does not record the password in a spool file.

  ```
  SET VERIFY OFF
   ACCEPT user CHAR PROMPT 'Enter user to connect to: '
   ACCEPT password CHAR PROMPT 'Enter the password for that user: ' HIDE
   CONNECT &user/&password
  ```

  In this example:

  - `SET VERIFY OFF` prevents the password from being displayed. (`SET VERIFY` lists each line of the script before and after substitution.) Combining the `SET VERIFY OFF` command with the `HIDE` command is a useful technique for hiding passwords and other sensitive input data.
  - `ACCEPT password CHAR PROMPT` includes the `HIDE` option for the `ACCEPT password` prompt, which prevents the input password from being echoed.

  The next example, which uses positional parameters, poses security risks because a user may invoke the script by passing the password on the command line. If the user does not

enter a password and instead is prompted, the danger lies in that whatever the user types is echoed to the screen and to a spool file if spooling is enabled.

```
CONNECT &1/&2
```

- **Control the log in times for batch scripts.** For batch scripts that require passwords, configure the account so that the script can only log in during the time in which it is supposed to run. For example, suppose you have a batch script that runs for an hour each evening starting at 8 p.m. Set the account so that the script can only log in during this time. If an intruder manages to gain access, then they have less of a chance of exploiting any compromised accounts.
- **Be careful when using DML or DDL SQL statements that prompt for passwords.** In this case, sensitive information is passed in clear text over the network. You can remedy this problem by using Oracle strong authentication.

  The following example of altering a password is secure because the password is not exposed:

  ```
  password psmith
  Changing password for psmith
  New password: password
  Retype new password: password
  ```

  This example poses a security risk because the password is exposed both at the command line and on the network:

  ```
  ALTER USER psmith IDENTIFIED BY password
  ```

12.4.2 Use of an External Password Store to Secure Passwords

You can store password credentials for connecting to a database by using a client-side Oracle wallet.

An Oracle wallet is a secure software container that stores the authentication and signing credentials needed for a user to log in.

Related Topics

- Managing the Secure External Password Store for Password Credentials
  The secure external password store (SEPS) is a client-side wallet that is used to store password credentials.
- *Oracle Database Enterprise User Security Administrator's Guide*

12.4.3 Securing Passwords Using the ORAPWD Utility

`SYSDBA` or `SYSOPER` users can use password files to connect to an application over a network.

- To create the password file, use the `ORAPWD` utility.

Related Topics

- *Oracle Database Administrator's Guide*

12.4.4 Example: Java Code for Reading Passwords

You can create Java packages that can be used to read passwords.

Example 12-1 demonstrates how to create a Java package that can be used to read passwords.

Example 12-1 Java Code for Reading Passwords

```
// Change the following line to a name for your version of this package
package passwords.sysman.emSDK.util.signing;

import java.io.IOException;
import java.io.PrintStream;
import java.io.PushbackInputStream;
import java.util.Arrays;

/**
 * The static readPassword method in this class issues a password prompt
 * on the console output and returns the char array password
 * entered by the user on the console input.
 */
public final class ReadPassword {
  //----------------------------------
  /**
   * Test driver for readPassword method.
   * @param args the command line args
   */
  public static void main(String[] args) {
    char[] pass = ReadPassword.readPassword("Enter password: ");
    System.out.println("The password just entered is \""
      + new String(pass) + "\"");
    System.out.println("The password length is " + pass.length);
  }
   * Issues a password prompt on the console output and returns
   * the char array password entered by the user on the console input.
   * The password is not displayed on the console (chars are not echoed).
   * As soon as the returned char array is not needed,
   * it should be erased for security reasons (Arrays.fill(charArr, ' '));
   * A password should never be stored as a java String.
   *
   * Note that Java 6 has a Console class with a readPassword method,
   * but there is no equivalent in Java 5 or Java 1.4.
   * The readPassword method here is based on Sun's suggestions at
   * http://java.sun.com/developer/technicalArticles/Security/pwordmask.
   *
   * @param prompt the password prompt to issue
   * @return new char array containing the password
   * @throws RuntimeException if some error occurs
   */
  public static final char[] readPassword(String prompt)
  throws RuntimeException {
    try {
      StreamMasker masker = new StreamMasker(System.out, prompt);
      Thread threadMasking = new Thread(masker);
      int firstByte = -1;
      PushbackInputStream inStream = null;
      try {
        threadMasking.start();
        inStream = new PushbackInputStream(System.in);
        firstByte = inStream.read();
      } finally {
        masker.stopMasking();
      }
      try {
        threadMasking.join();
      } catch (InterruptedException e) {
        throw new RuntimeException("Interrupt occurred when reading password");
      }
```

```
    if (firstByte == -1) {
      throw new RuntimeException("Console input ended unexpectedly");
    }
    if (System.out.checkError()) {
      throw new RuntimeException("Console password prompt output error");
    }
    inStream.unread(firstByte);
    return readLineSecure(inStream);
  }
  catch (IOException e) {
    throw new RuntimeException("I/O error occurred when reading password");
  }
}
//----------------------------------
/**
 * Reads one line from an input stream into a char array in a secure way
 * suitable for reading a password.
 * The char array will never contain a '\n' or '\r'.
 *
 * @param inStream the pushback input stream
 * @return line as a char array, not including end-of-line-chars;
 *  never null, but may be zero length array
 * @throws RuntimeException if some error occurs
 */
private static final char[] readLineSecure(PushbackInputStream inStream)
throws RuntimeException {
  if (inStream == null) {
    throw new RuntimeException("readLineSecure inStream is null");
  }
  try {
    char[] buffer = null;
    try {
      buffer = new char[128];
      int offset = 0;
      // EOL is '\n' (unix), '\r\n' (windows), '\r' (mac)
      loop:
      while (true) {
        int c = inStream.read();
        switch (c) {
        case -1:
        case '\n':
          break loop;
        case '\r':
          int c2 = inStream.read();
          if ((c2 != '\n') && (c2 != -1))
            inStream.unread(c2);
          break loop;
        default:
          buffer = checkBuffer(buffer, offset);
          buffer[offset++] = (char) c;
          break;
        }
      }
      char[] result = new char[offset];
      System.arraycopy(buffer, 0, result, 0, offset);
      return result;
    }
    finally {
      if (buffer != null)
        Arrays.fill(buffer, ' ');
    }
  }
```

```
  catch (IOException e) {
    throw new RuntimeException("I/O error occurred when reading password");
  }
}
//----------------------------------
/**
 * This is a helper method for readLineSecure.
 *
 * @param buffer the current char buffer
 * @param offset the current position in the buffer
 * @return the current buffer if it is not yet full;
 *  otherwise return a larger buffer initialized with a copy
 *  of the current buffer and then erase the current buffer
 * @throws RuntimeException if some error occurs
 */
private static final char[] checkBuffer(char[] buffer, int offset)
throws RuntimeException
{
  if (buffer == null)
    throw new RuntimeException("checkBuffer buffer is null");
  if (offset < 0)
    throw new RuntimeException("checkBuffer offset is negative");
  if (offset < buffer.length)
    return buffer;
  else {
    try {
      char[] bufferNew = new char[offset + 128];
      System.arraycopy(buffer, 0, bufferNew, 0, buffer.length);
      return bufferNew;
    } finally {
      Arrays.fill(buffer, ' ');
    }
  }
}
//----------------------------------
/**
 * This private class prints a one line prompt
 * and erases reply chars echoed to the console.
 */
private static final class StreamMasker
extends Thread {
  private static final String BLANKS = StreamMasker.repeatChars(' ', 10);
  private String m_promptOverwrite;
  private String m_setCursorToStart;
  private PrintStream m_out;
  private volatile boolean m_doMasking;
  //----------------------------------
  /**
   * Constructor.
   * @throws RuntimeException if some error occurs
   */
  public StreamMasker(PrintStream outPrint, String prompt)
  throws RuntimeException {
    if (outPrint == null)
      throw new RuntimeException("StreamMasker outPrint is null");
    if (prompt == null)
      throw new RuntimeException("StreamMasker prompt is null");
    if (prompt.indexOf('\r') != -1)
      throw new RuntimeException("StreamMasker prompt contains a CR");
    if (prompt.indexOf('\n') != -1)
      throw new RuntimeException("StreamMasker prompt contains a NL");
    m_out = outPrint;
```

```
      m_setCursorToStart = StreamMasker.repeatChars('\010',
        prompt.length() + BLANKS.length());
      m_promptOverwrite = m_setCursorToStart + prompt + BLANKS
        + m_setCursorToStart + prompt;
    }
    //------------------------------------
    /**
     * Begin masking until asked to stop.
     * @throws RuntimeException if some error occurs
     */
    public void run()
    throws RuntimeException {
      int priorityOriginal = Thread.currentThread().getPriority();
      Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
      try {
        m_doMasking = true;
        while (m_doMasking) {
          m_out.print(m_promptOverwrite);
          if (m_out.checkError())
            throw new RuntimeException("Console output error writing prompt");
          try {
            Thread.currentThread().sleep(1);
          } catch (InterruptedException ie) {
            Thread.currentThread().interrupt();
            return;
          }
        }
        m_out.print(m_setCursorToStart);
      } finally {
        Thread.currentThread().setPriority(priorityOriginal);
      }
    }
    //------------------------------------
    /**
     * Instructs the thread to stop masking.
     */
    public void stopMasking() {
      m_doMasking = false;
    }
    //------------------------------------
    /**
     * Returns a repeated char string.
     *
     * @param c the char to repeat
     * @param length the number of times to repeat the char
     * @throws RuntimeException if some error occurs
     */
    private static String repeatChars(char c, int length)
    throws RuntimeException {
      if (length < 0)
        throw new RuntimeException("repeatChars length is negative");
      StringBuffer sb = new StringBuffer(length);
      for (int i = 0; i < length; i++)
        sb.append(c);
      return sb.toString();
    }
  }
}
```

12.5 Securing External Procedures

An external procedure is stored in a `.dll` or an `.so` file, separately from the database, and can be through a credential authentication.

- About Securing External Procedures
  For safety reasons, Oracle external procedures run in a process that is physically separate from the database.
- General Process for Configuring extproc for a Credential Authentication
  For better security, you can configure the `extproc` process to be authenticated through a credential.
- extproc Process Authentication and Impersonation Expected Behaviors
  The `extproc` process has a set of behaviors for authentication and impersonation.
- Configuring Authentication for External Procedures
  To configure a credential for `extproc` processes, you can use the `DBMS_CREDENTIAL` PL/SQL package.
- External Procedures for Legacy Applications
  For maximum security, set the `ENFORCE_CREDENTIAL` environment variable to `TRUE`.

12.5.1 About Securing External Procedures

For safety reasons, Oracle external procedures run in a process that is physically separate from the database.

In most cases, you configure this process to run as a user other than the Oracle software account. When your application invokes this external procedure—such as when a library of `.dll` or `.so` files must be accessed—then Oracle Database creates an operating system process called `extproc`. By default, the `extproc` process communicates directly through your server process. In other words, if you do not use a credential, then Oracle Database creates an `extproc` process for you in the default Oracle Database server configuration, and runs `extproc` as the `oracle` software account. Alternatively, it can communicate through the Oracle Database listener.

Related Topics

- Guideline for Securing External Procedures
  The `ENFORCE_CREDENTIAL` environment variable controls how an `extproc` process authenticates user credentials and callout functions.

12.5.2 General Process for Configuring extproc for a Credential Authentication

For better security, you can configure the `extproc` process to be authenticated through a credential.

The general process is as follows:

1. You create a credential and then configure your database to use it (that is, configure authentication for an external procedure).

   The credential is in an encrypted container. Both public and private synonyms can refer to this credential.

2. You make your initial connection to the database, which you are running in either a dedicated server or a shared server process.
3. Your application makes a call to an external procedure.

   If this is the first call, then Oracle Database creates an `extproc` process. Note that if you want to use a credential for `extproc`, then you cannot use the Oracle listener to spawn the `extproc` process.
4. The `extproc` process impersonates (that is, it runs on behalf of your supplied credential), loads the requisite `.dll`, `.so`, `.sl`, or `.a` file, and then sends your data between SQL and C.

Related Topics

- Configuring Authentication for External Procedures
  To configure a credential for `extproc` processes, you can use the `DBMS_CREDENTIAL` PL/SQL package.

12.5.3 extproc Process Authentication and Impersonation Expected Behaviors

The `extproc` process has a set of behaviors for authentication and impersonation.

Table 12-2 describes the expected behaviors of an `extproc` process based on possible authentication and impersonation scenarios.

In this table, `GLOBAL_EXTPROC_CREDENTIAL` is a reserved credential name for the default credential if the credential is not explicitly specified and if the `ENFORCE_CREDENTIAL` environment variable is set to `TRUE`. Therefore, Oracle strongly recommends that you create a credential by the that name if `ENFORCE_CREDENTIAL` is set to `TRUE`.

Table 12-2 Expected Behaviors for extproc Process Authentication and Impersonation Settings

ENFORCE_CREDENTIAL Environment Variable Setting	PL/SQL Library with Credential?	GLOBAL_EXTPROC_CREDENTIAL Credential Existence	Expected Behavior
`FALSE`	No	No	Uses the pre-release 12c authentication, which authenticates by operating system privilege of the owners of the Oracle listener or Oracle server process.
`FALSE`	No	Yes	Authenticates and impersonates with the Oracle Database instance-wide supplied `GLOBAL_EXTPROC_CREDENTIAL`. If only the `GLOBAL_EXTPROC_CREDENTIAL` credential is in use, then the `EXECUTE` privilege on this global credential is automatically granted to all users implicitly.
`FALSE`	Yes	No	Authenticates and impersonates with the credential defined in the PL/SQL library

Table 12-2 (Cont.) Expected Behaviors for extproc Process Authentication and Impersonation Settings

ENFORCE_CREDENTIAL Environment Variable Setting	PL/SQL Library with Credential?	GLOBAL_EXTPROC_CREDENTIAL Credential Existence	Expected Behavior
FALSE	Yes	Yes	Authenticates and impersonates. If both the PL/SQL library and the GLOBAL_EXTPROC_CREDENTIAL settings have credentials defined, then the credential of the PL/SQL library takes precedence.
TRUE	No	No	Returns error ORA-28575: unable to open RPC connection to external procedure agent
TRUE	No	Yes	Authenticates and impersonates with the Oracle system-wide supplied GLOBAL_EXTPROC_CREDENTIAL (footnote 1)
TRUE	Yes	No	Authenticates and impersonates with the credential defined in the PL/SQL library
TRUE	Yes	Yes	Authenticates and impersonates (footnote 2)

12.5.4 Configuring Authentication for External Procedures

To configure a credential for extproc processes, you can use the DBMS_CREDENTIAL PL/SQL package.

1. Log in to a PDB as a user who has been granted the CREATE CREDENTIAL or CREATE ANY CREDENTIAL privilege.

   In addition, ensure that you also have the CREATE LIBRARY or CREATE ANY LIBRARY privilege, and the EXECUTE object privilege on the library that contains the external calls.

   ```
   sqlplus psmith@hpdb
   Enter password: password
   Connected.
   ```

   To find the available PDBs in a CDB, log in to the CDB root container and then query the PDB_NAME column of the DBA_PDBS data dictionary view. To check the current container, run the show con_name command.

2. Using the DBMS_CREDENTIAL PL/SQL package, create a new credential.

   For example:

   ```
   BEGIN
     DBMS_CREDENTIAL.CREATE_CREDENTIAL (
       credential_name   => 'smith_credential',
       user_name         => 'tjones',
       password          => 'password')
   END;
   /
   ```

In this example:

- `credential_name`: Enter the name of the credential. Optionally, prefix it with the name of a schema (for example, `psmith.smith_credential`). If the `ENFORCE_CREDENTIAL` environment variable is set to `TRUE`, then you should create a credential with credential_name `GLOBAL_EXTPROC_CREDENTIAL`.
- `user_name`: Enter a valid operating system user name to be to used to run as the user.
- `password`: Enter the password for the `user_name` user.

3. Associate the credential with a PL/SQL library.

   For example:

   ```
   CREATE OR REPLACE LIBRARY ps_lib
    AS 'smith_lib.so' IN DLL_LOC
    CREDENTIAL smith_credential;
   ```

   In this example, `DLL_LOC` is a directory object that points to the `$ORACLE_HOME/bin` directory. Oracle does not recommend using absolute paths to the DLL.

   When the PL/SQL library is loaded by an external procedure call through the `extproc` process, `extproc` now can authenticate and impersonate on behalf of the defined `smith_credential` credential.

4. Register the external procedure by creating a PL/SQL procedure or function that tells PL/SQL how to call the external procedure and what arguments to pass to it.

   For example, to create a function that registers an external procedure that was written in C, only use the `AS LANGUAGE C`, `LIBRARY`, and `NAME` clauses of the `CREATE FUNCTION` statement, as follows:

   ```
   CREATE OR REPLACE FUNCTION getInt (x VARCHAR2, y BINARY_INTEGER)
   RETURN BINARY_INTEGER
   AS LANGUAGE C
   LIBRARY ps_lib
   NAME "get_int_vals"
   PARAMETERS (x STRING, y int);
   ```

Related Topics

- Guideline for Securing External Procedures
  The `ENFORCE_CREDENTIAL` environment variable controls how an `extproc` process authenticates user credentials and callout functions.
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Call Interface Developer's Guide*
- *Oracle Database Net Services Administrator's Guide*

12.5.5 External Procedures for Legacy Applications

For maximum security, set the `ENFORCE_CREDENTIAL` environment variable to `TRUE`.

However, if you must accommodate backward compatibility, then set `ENFORCE_CREDENTIAL` to `FALSE`. `FALSE` enables the `extproc` process to authenticate, impersonate, and perform user-defined callout functions on behalf of the supplied credential when either of the following occurs:

- The credential is defined with a PL/SQL library.
- The credential is not defined but the `GLOBAL_EXTPROC_CREDENTIAL` credential exists.

If neither of these credential definitions is in place, then setting the ENFORCE_CREDENTIAL parameter to FALSE sets the extproc process to be authenticated by the operating system privilege of the owners of the Oracle listener or Oracle server process.

For legacy applications that run on top of extproc processes, ideally you should change the legacy application code to associate all alias libraries with credentials. If you cannot do this, then Oracle Database uses the GLOBAL_EXTPROC_CREDENTIAL credential to determine how authentication will be handled. If the GLOBAL_EXTPROC_CREDENTIAL credential is not defined, then the extproc process is authenticated by the operating system privilege of the owners of the Oracle listener or Oracle server process.

12.6 Securing LOBs with LOB Locator Signatures

You can secure large objects (LOB) by regenerating their LOB locator signatures.

- About Securing LOBs with LOB Locator Signatures
  A LOB locator, which is a pointer to the actual location of a large object (LOB) value, can be assigned a signature, which can be used to secure the LOB.
- Managing the Encryption of a LOB Locator Signature Key
  You can use the ALTER DATABASE DICTIONARY SQL statement to encrypt a LOB locator signature key.

12.6.1 About Securing LOBs with LOB Locator Signatures

A LOB locator, which is a pointer to the actual location of a large object (LOB) value, can be assigned a signature, which can be used to secure the LOB.

When you create a LOB, Oracle Database automatically assigns a signature to the LOB locator. Oracle Database verifies the signature matches when it receives a locator from a client to ensure that the locator has not been tampered with. Signature-based security can be used for both persistent and temporary LOB locators. It is also used for distributed CLOBs, BLOBs, and NBLOBs that come from index organized table (IOT) locators.

In an Oracle Real Applications Clusters (Oracle RAC) environment, all instances will share the same signature key, which is persisted in the database. Each pluggable database (PDB) will have its own signature key. If a LOB locator has been tampered with, the signature verification rejects the LOB and raises an ORA-64219: invalid LOB locator encountered error.

You can encrypt, rekey, and delete the LOB signature key that was used to generate LOB signature for LOB locators that are sent from a standalone database or PDB to a client. If you plan to encrypt the signature key, then the database (or PDB) in which the key resides must have an open TDE keystore.

To enable the LOB signature feature, you must set the LOB_SIGNATURE_ENABLE initialization parameter to TRUE. By default, LOB_SIGNATURE_ENABLE is set to FALSE.

12.6.2 Managing the Encryption of a LOB Locator Signature Key

You can use the ALTER DATABASE DICTIONARY SQL statement to encrypt a LOB locator signature key.

1. Log in to the database as a user who has ALTER DATABASE DICTIONARY privileges.
2. If necessary, enable the LOB signature key feature by setting the LOB_SIGNATURE_ENABLE initialization parameter to TRUE.

```
ALTER SYSTEM SET LOB_SIGNATURE_ENABLE = TRUE;
```

Alternatively, you can set the `LOB_SIGNATURE_ENABLE` parameter in the `init.ora` initialization file before a database restart. This enables the LOB signature key feature for all PDBs.

3. If you plan to encrypt the signature key, then ensure that the database or PDB has an open TDE keystore.

   You must have the `SYSKM` administrative privilege to create a TDE keystore.

   For example, to create and open a software TDE keystore:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY password;
   ```

4. Run the `ALTER DATABASE DICTIONARY` statement to set the LOB signature key configuration.

   - To encrypt the LOB locator signature key instead of obfuscating it, run the following statement:

     ```
     ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
     ```

   - To regenerate the LOB locator signature key for LOB locators that will be sent to a client, use the following statement. If the database is in restricted mode, then Oracle Database regenerates a new LOB signature key to encrypt the regenerated signature key. If the database is in non-restricted mode, then a new signature key is not regenerated but instead, Oracle Database uses a new encryption key to encrypt the existing LOB signature key. Oracle recommends that a database administrator or PDB administrator run this statement in restricted mode on a periodic basis, preferably during database down time.

     ```
     ALTER DATABASE DICTIONARY REKEY CREDENTIALS;
     ```

   - To delete the encrypted LOB locator signature key and then regenerate a new LOB signature key in obfuscated form instead of encrypted form, run the following statement:

     ```
     ALTER DATABASE DICTIONARY DELETE CREDENTIALS;
     ```

Related Topics

- Configuring Transparent Data Encryption

12.7 Managing Application Privileges

Most database applications involve different privileges on different schema objects.

Keeping track of the privileges that are required for each application can be complex. In addition, authorizing users to run an application can involve many `GRANT` operations. To simplify application privilege management, create a role for each application and grant that role all the privileges a user must run the application. In fact, an application can have several roles, each granted a specific subset of privileges that allow greater or lesser capabilities while running the application. For example, suppose every administrative assistant uses the Vacation application to record the vacation taken by members of the department. To best manage this application, you should do the following:

1. Create a `VACATION` role.
2. Grant all privileges required by the Vacation application to the `VACATION` role.

Useful data dictionary views are `ROLE_TAB_PRIVS`, `ROLE_SYS_PRIVS`, and `DBA_ROLE_PRIVS`.

3. Grant the `VACATION` role to all administrative assistants. Better yet, create a role that defines the privileges the administrative assistants have, and then grant the `VACATION` role to that role.

Related Topics

- Creating a Role
  You can create a role that is authenticated with or without a password. You also can create external or global roles.
- User Privilege and Role Data Dictionary Views
  You can use special queries to find information about various types of privilege and role grants.

12.8 Advantages of Using Roles to Manage Application Privileges

Grouping application privileges in a role aids privilege management.

Consider the following administrative options:

- You can grant the role, rather than many individual privileges, to those users who run the application. Then, as employees change jobs, you need to grant or revoke only one role, rather than many privileges.
- You can change the privileges associated with an application by modifying only the privileges granted to the role, rather than the privileges held by all users of the application.
- You can determine the privileges that are necessary to run a particular application by querying the `ROLE_TAB_PRIVS` and `ROLE_SYS_PRIVS` data dictionary views.
- You can determine which users have privileges on which applications by querying the `DBA_ROLE_PRIVS` data dictionary view.

12.9 Creating Secure Application Roles to Control Access to Applications

A secure application role is only enabled through its associated PL/SQL package or procedure.

- Step 1: Create the Secure Application Role
  The `CREATE ROLE` statement with the `IDENTIFIED USING` clause creates a secure application role.
- Step 2: Create a PL/SQL Package to Define the Access Policy for the Application
  You can create a PL/SQL package that defines the access policy for your application.

12.9.1 Step 1: Create the Secure Application Role

The `CREATE ROLE` statement with the `IDENTIFIED USING` clause creates a secure application role.

You must have the `CREATE ROLE` system privilege to run this statement.

For example, to create a secure application role called `hr_admin` that is associated with the `sec_mgr.hr_admin` package:

1. Create the security application role as follows:

   ```
   CREATE ROLE hr_admin IDENTIFIED USING sec_mgr.hr_admin_role_check;
   ```

   This statement indicates the following:

   - The role `hr_admin` to be created is a secure application role.
   - The role can only be enabled by modules defined inside the PL/SQL procedure `sec_mgr.hr_admin_role_check`. At this stage, this procedure does not need to exist.

2. Grant the security application role the privileges you would normally associate with this role.

   For example, to grant the `hr_admin` role `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the `HR.EMPLOYEES` table, you enter the following statement:

   ```
   GRANT SELECT, INSERT, UPDATE, DELETE ON HR.EMPLOYEES TO hr_admin;
   ```

   Do not grant the role directly to the user. The PL/SQL procedure or package does that for you, assuming the user passes its security policies.

12.9.2 Step 2: Create a PL/SQL Package to Define the Access Policy for the Application

You can create a PL/SQL package that defines the access policy for your application.

- About Creating a PL/SQL Package to Define the Access Policy for an Application
  To enable or disable the secure application role, you must create the security policies of the role within a PL/SQL package.
- Creating a PL/SQL Package or Procedure to Define the Access Policy for an Application
  The PL/SQL package or procedure that you create must use invoker's rights to define the access policy.
- Testing the Secure Application Role
  As a user who has been granted the secure application role, try performing an action that requires the privileges the role grants.

12.9.2.1 About Creating a PL/SQL Package to Define the Access Policy for an Application

To enable or disable the secure application role, you must create the security policies of the role within a PL/SQL package.

You also can create an individual procedure to do this, but a package lets you group a set of procedures together. This lets you group a set of policies that, used together, present a solid security strategy to protect your applications. For users (or potential intruders) who fail the security policies, you can add auditing checks to the package to record the failure. Typically, you create this package in the schema of the security administrator.

The package or procedure must accomplish the following:

- **It must use invoker's rights to enable the role.**To create the package using invoker's rights, you must set the `AUTHID` property to `CURRENT_USER`. You cannot create the package by using definer's rights.
- **It must include one or more security checks to validate the user.** One way to validate users is to use the `SYS_CONTEXT` SQL function. To find session information for a user, you can use `SYS_CONTEXT` with an application context.

- **It must issue a SET ROLE SQL statement or DBMS_SESSION.SET_ROLE procedure when the user passes the security checks.** Because you create the package using invoker's rights, you must set the role by issuing the `SET ROLE` SQL statement or the `DBMS_SESSION.SET_ROLE` procedure. (However, you cannot use the `SET ROLE ALL` statement for this type of role enablement.) The PL/SQL embedded SQL syntax does not support the `SET ROLE` statement, but you can invoke `SET ROLE` by using dynamic SQL (for example, with `EXECUTE IMMEDIATE`).

Because of the way that you must create this package or procedure, you cannot use a logon trigger to enable or disable a secure application role. Instead, invoke the package directly from the application when the user logs in, before the user must use the privileges granted by the secure application role.

Related Topics

- *Oracle Database PL/SQL Language Reference*
- Using Application Contexts to Retrieve User Information
  An application context stores user identification that can enable or prevent a user from accessing data in the database.
- *Oracle Database PL/SQL Language Reference*

12.9.2.2 Creating a PL/SQL Package or Procedure to Define the Access Policy for an Application

The PL/SQL package or procedure that you create must use invoker's rights to define the access policy.

For example, suppose you wanted to restrict anyone using the `hr_admin` role to employees who are on site (that is, using certain terminals) and between the hours of 8 a.m. and 5 p.m. As the system or security administrator, you can create a procedure that defines the access policy for the application.

1. Create the procedure as follows:

```
CREATE OR REPLACE PROCEDURE hr_admin_role_check
 AUTHID CURRENT_USER
 AS
 BEGIN
  IF (SYS_CONTEXT ('userenv','ip_address')
    IN ('192.0.2.10' , '192.0.2.11')
     AND
    TO_CHAR (SYSDATE, 'HH24') BETWEEN 8 AND 17)
  THEN
    EXECUTE IMMEDIATE 'SET ROLE hr_admin';
  END IF;
 END;
/
```

In this example:

- `AUTHID CURRENT_USER` sets the AUTHID property to CURRENT_USER so that invoker's rights can be used.
- `IF (SYS_CONTEXT ('userenv','ip_address')` validates the user by using the `SYS_CONTEXT` SQL function to retrieve the user session information.

- `BETWEEN ... TO_CHAR` creates a test to grant or deny access. The test restricts access to users who are on site (that is, using certain terminals) and working between the hours of 8:00 a.m. and 5:00 p.m. If the user passes this check, the `hr_admin` role is granted.
- `THEN... EXECUTE` grants the role to the user by issuing the `SET ROLE` statement using the `EXECUTE IMMEDIATE` command, assuming the user passes the test.

2. Grant `EXECUTE` permissions for the `hr_admin_role_check` procedure to any user who was assigned it.

   For example:

   ```
   GRANT EXECUTE ON hr_admin_role_check TO psmith;
   ```

12.9.2.3 Testing the Secure Application Role

As a user who has been granted the secure application role, try performing an action that requires the privileges the role grants.

When you log in as a user who has been granted the secure application role, the role is then enabled.

1. As the user who has been granted the role, log in to the PDB where the application role was created.

   For example:

   ```
   CONNECT PSMITH@pdb_name
   Enter password: password
   ```

2. Perform an action that requires the privileges the secure application role grants.

   For example, if the role grants the `EXECUTE` privilege for a procedure called `sec_admin.hr_admin_role_check`:

   ```
   EXECUTE sec_admin.hr_admin_role_check;
   ```

12.10 Association of Privileges with User Database Roles

Ensure that users have only the privileges associated with the current database role.

- Why Users Should Only Have the Privileges of the Current Database Role
  A single user can use many applications and associated roles.
- Use of the SET ROLE Statement to Automatically Enable or Disable Roles
  You can use a `SET ROLE` statement at the beginning of each application to automatically enable its associated role and to disable all others.

12.10.1 Why Users Should Only Have the Privileges of the Current Database Role

A single user can use many applications and associated roles.

However, you should ensure that the user has only the privileges associated with the current database role.

Consider the following scenario:

- The ORDER role (for an application called Order) contains the UPDATE privilege for the INVENTORY table.
- The INVENTORY role (for an application called Inventory) contains the SELECT privilege for the INVENTORY table.
- Several order entry clerks were granted both the ORDER and INVENTORY roles.

In this scenario, an order entry clerk who was granted both roles can use the privileges of the ORDER role when running the INVENTORY application to update the INVENTORY table. The problem is that updating the INVENTORY table is not an authorized action for the INVENTORY application. It is an authorized action for the ORDER application. To avoid this problem, use the SET ROLE statement as explained in the following section.

12.10.2 Use of the SET ROLE Statement to Automatically Enable or Disable Roles

You can use a SET ROLE statement at the beginning of each application to automatically enable its associated role and to disable all others.

This way, each application dynamically enables particular privileges for a user only when required. The SET ROLE statement simplifies privilege management. You control what information users can access and when they can access it. The SET ROLE statement also keeps users operating in a well-defined privilege domain. If a user obtains privileges only from roles, then the user cannot combine these privileges to perform unauthorized operations.

Related Topics

- How Grants and Revokes Work with SET ROLE and Default Role Settings
  Privilege grants and the SET ROLE statement affect when and how grants and revokes take place.
- When Grants and Revokes Take Effect
  Depending on the privilege that is granted or revoked, a grant or revoke takes effect at different times.

12.11 Protecting Database Objects by Using Schemas

A schema is a security domain that can contain database objects. Privileges granted to users and roles control access to these database objects.

- Protecting Database Objects in a Unique Schema
  Think of most schemas as user names: the accounts that enable users to connect to a database and access the database objects.
- Protection of Database Objects in a Shared Schema
  For many applications, users only need access to an application schema; they do not need their own accounts or schemas in the database.

12.11.1 Protecting Database Objects in a Unique Schema

Think of most schemas as user names: the accounts that enable users to connect to a database and access the database objects.

However, a *unique schema* does not allow connections to the database, but is used to contain a related set of objects. Schemas of this sort are created as typical users, and yet are not granted the CREATE SESSION system privilege (either explicitly or through a role).

- To protect the objects, temporarily grant the `CREATE SESSION` and `RESOURCE` privilege to a unique schema if you want to use the `CREATE SCHEMA` statement to create multiple tables and views in a single transaction.

For example, a given schema might own the schema objects for a specific application. If application users have the privileges to do so, then they can connect to the database using typical database user names and use the application and the corresponding objects. However, no user can connect to the database using the schema set up for the application. This configuration prevents access to the associated objects through the schema, and provides another layer of protection for schema objects. In this case, the application could issue an `ALTER SESSION SET CURRENT_SCHEMA` statement to connect the user to the correct application schema.

12.11.2 Protection of Database Objects in a Shared Schema

For many applications, users only need access to an application schema; they do not need their own accounts or schemas in the database.

For example, users John, Firuzeh, and Jane are all users of the Payroll application, and they need access to the `payroll` schema on the `finance` database. None of them need to create their own objects in the database. They need to only access the `payroll` objects. To address this issue, Oracle Database provides the enterprise users, which are schema-independent users.

Enterprise users, users managed in a directory service, do not need to be created as database users because they use a shared database schema. To reduce administration costs, you can create an enterprise user once in the directory, and point the user at a shared schema that many other enterprise users can also access.

> **Note:**
>
> Enterprise User Security (EUS) is deprecated with Oracle Database 23ai.
> Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

Related Topics

- *Oracle Database Enterprise User Security Administrator's Guide*

12.12 Object Privileges in an Application

When you design an application, consider the types of users and the level access they need.

- What Application Developers Must Know About Object Privileges
  Object privileges enable end users to perform actions on objects such as tables, views, sequences, procedures, functions, or packages.
- SQL Statements Permitted by Object Privileges
  As you implement and test your application, you should create each necessary role.

12.12.1 What Application Developers Must Know About Object Privileges

Object privileges enable end users to perform actions on objects such as tables, views, sequences, procedures, functions, or packages.

Table 12-3 summarizes the object privileges available for each type of object.

Table 12-3 How Privileges Relate to Schema Objects

Object Privilege	Applies to Table?	Applies to View?	Applies to Sequence?	Applies to Standalone Stored Procedures, Functions, or Public Package Constructs
`ALTER`	Yes	No	Yes	No
`DELETE`	Yes	Yes	No	No
`EXECUTE`	No	No	No	Yes
`INDEX`	Yes (privilege that cannot be granted to a role)	No	No	No
`INSERT`	Yes	Yes	No	No
`REFERENCES`	Yes (privilege that cannot be granted to a role)	No	No	No
`SELECT`	Yes	Yes (can also be granted for snapshots)	Yes	No
`UPDATE`	Yes	Yes	No	No

Related Topics

- Auditing Object Actions
  You can use the `CREATE AUDIT POLICY` statement to audit object actions.

12.12.2 SQL Statements Permitted by Object Privileges

As you implement and test your application, you should create each necessary role.

Test the usage scenario for each role to ensure that the users of your application will have proper access to the database. After completing your tests, coordinate with the administrator of the application to ensure that each user is assigned the proper roles.

Table 12-4 lists the SQL statements permitted by the object privileges shown in Table 12-3.

Table 12-4 SQL Statements Permitted by Database Object Privileges

Object Privilege	SQL Statements Permitted
`ALTER`	`ALTER` object (table or sequence) `CREATE TRIGGER ON` object (tables only)

Table 12-4 (Cont.) SQL Statements Permitted by Database Object Privileges

Object Privilege	SQL Statements Permitted
DELETE	DELETE FROM object (table, view, or synonym)
EXECUTE	EXECUTE object (procedure or function)
	References to public package variables
INDEX	CREATE INDEX ON object (table, view, or synonym)
INSERT	INSERT INTO object (table, view, or synonym)
REFERENCES	CREATE or ALTER TABLE statement defining a FOREIGN KEY integrity constraint on object (tables only)
SELECT	SELECT...FROM object (table, view, synonym, or snapshot)
	SQL statements using a sequence

Related Topics

- About Privileges and Roles
  Authorization permits users to access, process, or alter data; it also creates limitations on user access or actions.
- Auditing Object Actions
  You can use the CREATE AUDIT POLICY statement to audit object actions.

12.13 Parameters for Enhanced Security of Database Communication

Parameters can be used to manage security, such as handling bad packets from protocol errors or configuring the maximum number of authentication errors.

- Bad Packets Received on the Database from Protocol Errors
  The SEC_PROTOCOL_ERROR_TRACE_ACTION initialization parameter controls how trace files are managed when protocol errors are generated.
- Controlling Server Execution After Receiving a Bad Packet
  The SEC_PROTOCOL_ERROR_FURTHER_ACTION initialization parameter controls server execution after the server receives a bad packet.
- Configuration of the Maximum Number of Authentication Attempts
  The SEC_MAX_FAILED_LOGIN_ATTEMPTS initialization parameter sets the number of authentication attempts before the database will drop a failed connection.
- Configuring the Display of the Database Version Banner
  The SEC_RETURN_SERVER_RELEASE_BANNER initialization parameter can be used to prevent the display of detailed product information during authentication.
- Configuring Banners for Unauthorized Access and Auditing User Actions
  The SEC_USER_UNAUTHORIZED_ACCESS_BANNER and SEC_USER_AUDIT_ACTION_BANNER initialization parameters control the display of banners for unauthorized access and for auditing users.

12.13.1 Bad Packets Received on the Database from Protocol Errors

The `SEC_PROTOCOL_ERROR_TRACE_ACTION` initialization parameter controls how trace files are managed when protocol errors are generated.

Networking communication utilities such as Oracle Call Interface (OCI) or Two-Task Common (TTC) can generate a large disk file containing the stack trace and heap dump when the server receives a bad packet, out-of-sequence packet, or a private or an unused remote procedure call.

Typically, this disk file can grow quite large. An intruder can potentially cripple a system by repeatedly sending bad packets to the server, which can result in disk flooding and Denial of Service (DOS) attacks. An unauthenticated client can also mount this type of attack.

You can prevent these attacks by setting the `SEC_PROTOCOL_ERROR_TRACE_ACTION` initialization parameter to one of the following values:

- `None`: Configures the server to ignore the bad packets and does not generate any trace files or log messages. Use this setting if the server availability is overwhelmingly more important than knowing that bad packets are being received.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = None
  ```

- `Trace` (default setting): Creates the trace files, but it is useful for debugging purposes, for example, when a network client is sending bad packets as a result of a bug.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Trace
  ```

- `Log`: Writes a short, one-line message to the server trace file. This choice balances some level of auditing with system availability.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Log
  ```

- `Alert`: Sends an alert message to a database administrator or monitoring console.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Alert
  ```

12.13.2 Controlling Server Execution After Receiving a Bad Packet

The `SEC_PROTOCOL_ERROR_FURTHER_ACTION` initialization parameter controls server execution after the server receives a bad packet.

After Oracle Database detects a client or server protocol error, it must continue execution. However, this could subject the server to further bad packets, which could lead to disk flooding or denial-of-service attacks.

- To control the further execution of a server process when it is receiving bad packets from a potentially malicious client, set the `SEC_PROTOCOL_ERROR_FURTHER_ACTION` initialization parameter to one of the following values:
  - `Continue`: Continues the server execution. However, be aware that the server may be subject to further attacks.

    For example:

```
SEC_PROTOCOL_ERROR_FURTHER_ACTION = Continue
```

- `(Delay,m)`: Delays the client *m* seconds before the server can accept the next request from the same client connection. This setting prevents malicious clients from excessively using server resources while legitimate clients experience a degradation in performance but can continue to function. When you enter this setting, enclose it in parentheses.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = (Delay,3)
  ```

  If you are setting `SEC_PROTOCOL_ERROR_FURTHER_ACTION` by using the `ALTER SYSTEM` or `ALTER SESSION` SQL statement, then you must enclose the `Delay` setting in either single or double quotation marks.

  ```
  ALTER SYSTEM SEC_PROTOCOL_ERROR_FURTHER_ACTION = '(Delay,3)';
  ```

- `(Drop,n)`: Forcefully terminates the client connection after *n* bad packets. This setting enables the server to protect itself at the expense of the client, for example, loss of a transaction. However, the client can still reconnect, and attempt the same operation again. Enclose this setting in parentheses. The default value of `SEC_PROTOCOL_ERROR_FURTHER_ACTION` is `(Drop,3)`.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = (Drop,10)
  ```

  Similar to the `Delay` setting, you must enclose the `Drop` setting in single or double quotation marks if you are using `ALTER SYSTEM` or `ALTER SESSION` to change this setting.

12.13.3 Configuration of the Maximum Number of Authentication Attempts

The `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter sets the number of authentication attempts before the database will drop a failed connection.

As part of connection creation, the listener starts the server process and attaches it to the client. Using this physical connection, the client is this able to authenticate the connection. After a server process starts, client authenticates with this server process. An intruder could start a server process, and then issue an unlimited number of authenticated requests with different user names and passwords in an attempt to gain access to the database.

You can limit the number of failed login attempts for application connections by setting the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter to restrict the number of authentication attempts on a connection. After the specified number of authentication attempts fail, the database process drops the connection and the server process is terminated. By default, `SEC_MAX_FAILED_LOGIN_ATTEMPTS` is set to `3`.

Remember that the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter is designed to prevent potential intruders from attacking your applications, as well as valid users who have forgotten their passwords. The `sqlnet.ora INBOUND_CONNECT_TIMEOUT` parameter and the `FAILED_LOGIN_ATTEMPTS` profile parameter also restrict failed logins, but the difference is that these two parameters only apply to valid user accounts.

For example, to limit the maximum attempts to 5, set `SEC_MAX_FAILED_LOGIN_ATTEMPTS` as follows in the `initsid.ora` initialization parameter file:

```
SEC_MAX_FAILED_LOGIN_ATTEMPTS = 5
```

12.13.4 Configuring the Display of the Database Version Banner

The SEC_RETURN_SERVER_RELEASE_BANNER initialization parameter can be used to prevent the display of detailed product information during authentication.

Detailed product version information should not be accessible before a client connection (including an Oracle Call Interface client) is authenticated. An intruder could use the database version to find information about security vulnerabilities that may be present in the database software.

- To restrict the display of the database version banner to unauthenticated clients, set the SEC_RETURN_SERVER_RELEASE_BANNER initialization parameter in the init*sid*.ora initialization parameter file to either TRUE or FALSE.

  By default, SEC_RETURN_SERVER_RELEASE_BANNER is set to FALSE.

For example, if you set it to TRUE, then Oracle Database displays the full correct database version. For example, for Release 19.1.0.0:

```
Oracle Database 19c Enterprise Edition Release 19.1.0.0 - Production
```

If a release number uses point release notation (for example, Oracle Database Release 19.1.0.1), then the banner displays as follows:

```
Oracle Database 19c Enterprise Edition Release 19.1.0.1 - Production
```

However, if in that same release, you set it to NO, then Oracle Database restricts the banner to display the following fixed text starting with Release 19.1, which instead of 19.1.0.1 is 19.1.0.0.0:

```
Oracle Database 19c Release 19.1.0.0.0 - Production
```

12.13.5 Configuring Banners for Unauthorized Access and Auditing User Actions

The SEC_USER_UNAUTHORIZED_ACCESS_BANNER and SEC_USER_AUDIT_ACTION_BANNER initialization parameters control the display of banners for unauthorized access and for auditing users.

You should create and configure banners to warn users against unauthorized access and possible auditing of user actions. The notices are available to the client application when it logs into the database.

- To configure these banners to display, set the following sqlnet.ora parameters on the database server side to point to a text file that contains the banner information:
  - SEC_USER_UNAUTHORIZED_ACCESS_BANNER. For example:

    ```
    SEC_USER_UNAUTHORIZED_ACCESS_BANNER = /opt/Oracle/12c/dbs/unauthaccess.txt
    ```

  - SEC_USER_AUDIT_ACTION_BANNER. For example:

    ```
    SEC_USER_AUDIT_ACTION_BANNER = /opt/Oracle/12c/dbs/auditactions.txt
    ```

By default, these parameters are not set. In addition, be aware that there is a 512-byte limitation for the number of characters used for the banner text.

After you set these parameters, the Oracle Call Interface application must use the appropriate OCI APIs to retrieve these banners and present them to the end user.