SUPPLEMENT

Scenarios

This supplement to Expert Oracle and Java Security: Programming Secure Oracle Database Applications with Java includes material that will help you implement security, as described in the book. You can use this material to take the security structures that we have built out for a test drive, so to speak. Then you can use these same procedures to apply our security structures in your computing environment.

Please browse to my blog for more frequent updates and other information and resources, at:

http://oraclejavasecure.blogspot.com/

Note: This document is Under Construction!

Why is this supplement separate from the book? You might ask. Well, there are a few reasons. First is that books are written and published on a schedule, and this material was not ready on time. Second is that books have a pre-planned structure and content, and just because you have good ideas for things you can add to a book doesn't mean those ideas can be included. And the third reason is that this gives me a bit of flexibility to add to the book, after publication.

One area that I feel the book has left a topic prematurely is the area of passwords. We want to turn off remote `SYSDBA` access, and we want to reset the administrative passwords without exposing them.

Alternative Installations

I realize that not everyone will be able or inclined to install these security structures to two databases named `orcl` and `apver`. It may also be that some readers will not want to install and manage the interim structures as each chapter defines them; but will, rather, want to install the end state structures just in order to examine them as they read the book. For those instances, I hope to provide some helpful hints about how that can be accomplished in the following sections.

Installing All At Once

I have included a set of scripts and code that will allow the SYS user on a database to install a single-instance version of the end state of the structures presented in this book. You can find these scripts and code in the directory *Supplement/FromScratch*.

This installation requires an instance of Oracle 11g with standard, default pl/sql packages and an installation of the sample HR schema. Note that this will install a single-Oracle instance version of these structures. Refer to Chapter 11 to harden these structures by establishing an alternate instance for application verification.

As the SYS user on your database, run commands in *AllSys.sql* to build all structures. There are notes at the top of the script that indicate the items you need to modify in the script before running it. Those items include your local IP subnet addresses, the exact username of your operating system users and the passwords for the users you will be creating – 7 occurrences of "password". Note that if you have already built some of the structures from this book, you should uncomment the drop commands at the top of the script to clear those structures.

When completed with *AllSys.sql*, follow the instructions in Chapter 12, *Administration*, in the section named *Bootstrap OJSAdmin*. In the process, you will use these structures and set up administration of the same.

Afterwards, I recommend you run through the addition of apps/users/administrators in the next section of this supplement, and run through the exercise in this Supplement: *Responding to an Application Security Emergency*.

Installing On One Oracle Instance

The code directory *Chapter12Single* contains the code described in Chapter 12 of the book, except that only a single Oracle database instance is used. Follow the same directions given in the section named *Scripts Execution and Code Compilation*, except that you will skip the file *ApverSys.sql*.

Following that, you will also need to follow the instructions given in the section named *Bootstrap OJSAdmin* to set up administration of these services.

Using Oracle Instances Besides ORCL and APVER

For a dual-instance installation, I recommend `apver` as your second instance, so I'm not documenting how to change that, but perhaps you have a different instance name that you want to use as your primary Oracle instance. You could establish a SQLnet TNSNames entry to call (refer to) the existing instance by the name `orcl`. But if you did not want to do that, you can edit the code to use a different instance.

The chapter-by-chapter code would be more difficult; actually, it would just be *OracleJavaSecure.java*, *TestOracleJavasSecure.java* and other Java test files that you would need to direct at your instance name for their connections. However, if you install the from-scratch version, described in *Installing All At Once* above, then you can jump right to the end state (on a single-instance installation.) You will still need to edit the connection-string generator in the `main()` method of `OracleJavaSecure` to create an encoded connection string that references your instance name. You will

also need to edit the code for the GUI classes named PickAppManage, RegNewApp, AssignApp, AdminUsers and AddUser to point their connections at your instance.

Using Employees Data from Outside HR Schema

If you deploy these application security solutions in your organization, you are not going to want to use the sample HR schema for data related to single sign-on. In this case, you will need to create two tables similar to those described in this book: sms_carrier_host and emp_mobile_nos. In your version of the emp_mobile_nos table, you might include some columns that we are referencing in the HR.EMPLOYEES table: first_name, last_name and email. Don't forget to grant select on these tables to appsec user and ojs_adm_admin role.

You will also need to change the appsec.appsec_only_pkg.f_is_cur_cached_cd function and appsec.appsec_only_pkg.p_get_emp_2fact_nos procedure to read from your tables. Also, the hr.hr_pub_pkg package defined in the book will need to be rewritten to use your tables.

Using Structures without Two-Factor Authentication

I'm going to give you the quickest path to disabling the 2-factor authentication. This will not remove references to 2-factor authentication from the code, nor will it remove the exchange of authentication codes between the client and server; it's just that, the blank codes will be exchanged and accepted. No codes will be required, and no codes will be sent to mobile devices or e-mail.

Modify the appsec.appsec_public_pkg.p_get_app_conns procedure, commenting the 2-factor authentication code test and distribute lines, as shown below:

```
--IF( m_two_factor_cd IS NULL )
--THEN
--    m_err_txt := appsec_only_pkg.f_send_2_factor( return_user, m_application_id );
--ELSIF( appsec_only_pkg.f_is_cur_cached_cd(
--    return_user, m_application_id, m_two_factor_cd ) = 'Y' )
--THEN
    secret_pass_salt :=
        app_sec_pkg.f_get_crypt_secret_salt( ext_modulus, ext_exponent );
    secret_pass_count :=
        app_sec_pkg.f_get_crypt_secret_count( ext_modulus, ext_exponent );
    secret_pass :=
        app_sec_pkg.f_get_crypt_secret_pass( ext_modulus, ext_exponent );
    secret_pass_algorithm :=
        app_sec_pkg.f_get_crypt_secret_algorithm(ext_modulus, ext_exponent);
    m_crypt_connections := appsec_only_pkg.f_get_crypt_conns( m_class_instance );
--ELSE
--    -- Wrong 2-Factor code entered
--    RAISE NO_DATA_FOUND;
--END IF;
```

Make that same change to the appsec.appsec_admin_pkg.p_copy_app_conns procedure.

Using Structures in a Web Application

This paragraph is not yet written, but the intent is to get the web application passwords from the `appver` structures. Application verification is still in effect, but SSO is possibly not used and 2-factor authentication is definitely not used.

SSO may still be used if the user http session can be identified and authenticated, perhaps to an Active Directory.

Test Applications, Users and Administrators

The test applications and users we add in this section will be used to demonstrate levels of access and restrictions. None of these is needed outside the demonstrations and tests conducted here.

Adding Applications

We will add three applications for demonstration purposes. The code for these applications is in *Supplement/testojs*. The *testa* directory contains an application that demonstrates the implementation of SSO and 2-factor authentication without registering the application. For *testa* to run, no Oracle passwords are needed (nothing is stored in Application Verification). So no application inner class needs to be registered and no secure application role needs to be configured.

The *testb* directory is simply another instance of our `HR.EMPLOYEES` secure application, showing sensitive and public data in a table. The *testc* directory contains a bad example of an application to access the sample `OE` (Order Entry) schema, getting customer detail. The reason it is a bad example is that there is some data in the customer detail table that is sensitive. We will implement secure access to that data later in this supplement in the section named *Responding to an Application Security Emergency*.

Using SSO and 2-Factor Authentication in an Unregistered Application

Did you realize that we have been doing SSO, proxying to `appver` user and 2-factor authentication before we use application authentication and a stored connection string list. Applications can use our SSO and 2-factor authentication without registering and storing their connection strings with us.

Don't have an Oracle instance we can acquire, but have used connection internal to `OracleJavaSecure`. Found OS User ID for SSO. Now want to display it on application. Modified `OracleJavaSecure` to retain OS User ID, and created public method to return the ID, `getOSUserName()`.

Just in general terms, we can say that we are doing SSO and 2-factor authentication without being tied to a specific application. At least we can say that we do not even attempt to read the list of connection strings for a registered application until after we have passed 2-factor authentication with SSO and all that entails. Even if there is no list of connection strings registered for the application in question, if the user has passed 2-factor authentication, we will return an empty list of connection strings.

We will use that feature to our advantage. Let's say we have a Java application that does not have any Oracle connections. And for this application, we still need to know for certain who the current user is. Beyond SSO, we want to enforce 2-factor authentication. We can do this with the `Login` class. We should designate this application with a new application ID (name) and a different package. If not, we may gain the connection strings list for a different application, if the current user is granted proxy through the application user for that application. Don't get them if you don't need them! Additionally, a change in the inner class of the original application will break the other application. Also, if you are not careful with `CLASSPATH`, you may end up with two different implementations of `Login` class with the same name in your `CLASSPATH`, which can cause headaches in application use and troubleshooting.

So, you've got a copy of *Login.java* in a new package, and you've designated a new application ID in *Login.java* for this new application. In the default constructor for your new application, just instantiate a `new Login()` class. That will give you SSO and 2-factor authentication. Make sure you edited *Capter12/orajavsec/OracleJavaSecure.java* and that *ojdbc6.jar* is on your `CLASSPATH`, then from the Chapter12 directory, compile and run `TestAppA` as follows.

```
javac testojs/testa/TestAppA.java
java testojs.testa.TestAppA
```

You will get the `Login` prompt for the `TestAppA` application, and you will receive a 2-factor code. Once you've entered the 2-factor code, you will see the current OS username displayed. This is to demonstrate that your application, `TestAppA` has achieved SSO and 2-factor authentication, and that you can easily get the logged-in user. We have added a simple utility method in `OracleJavaSecure` to return the SSO user name, `getOSUserName()`, the code for which is given in Listing 12-71. To implement that, we modified the SSO process to capture the OS username and retain it as a static member of `OracleJavaSecure`.

Registering a New Application

TestAppB uses appusr to read hrview_role
TestAppC uses oeview to read customers

Adding Users

Administrative Users

osadmin has all
nonead has none
userad has employee/user admin
proxyad has administrator and user administration
apverad has connection string administration

Application Users

All have TestAppA - no getAAConnRole() to get secure application role
user1 has TestAppB
user2 has TestAppC
user3 has TestAppB and TestAppC

Application and Administrative Access

Assigning Specific Administrators to Applications

Can only be done by OSADMIN or other user given INSERT to application_admins

Granting Access to Applications

Responding to an Application Security Emergency

This is only an exercise: It has come to our attention that we have historically collected customer data that we must begin protecting according to more stringent data security requirements. This scenario should sound familiar, because we have all heard of so many high-profile data losses of customer data. We should be setting higher security standards and proactively protecting any customer data that we handle.

In particular, our new policy states that we protect customer Personally Identifying Information (PII) and other sensitive data from data loss. Further, our policy states that we will not archive those customer attributes unless they are encrypted.

We have presented our customer data to the Chief Information Officer and representatives of the Legal Department, and they have identified several columns of our customer data that must have enhanced protection: `DATE_OF_BIRTH`, `MARITAL_STATUS` and `INCOME_LEVEL`. There are several other fields that they want to consider for enhanced protection in the future.

In the sample oracle database, these fields occur in 2 places: the Order Entry (`OE`) schema contains current data and the Sales History (`SH`) schema contains an archive of our data. On review, we find that none of these fields are represented in any views – that is, we have views of our `OE.CUSTOMERS` table, but those views do not contain our sensitive columns. There are, however, applications that use the `OE.CUSTOMERS` table directly, without a view.

We have formulated this response to resolve the problem:

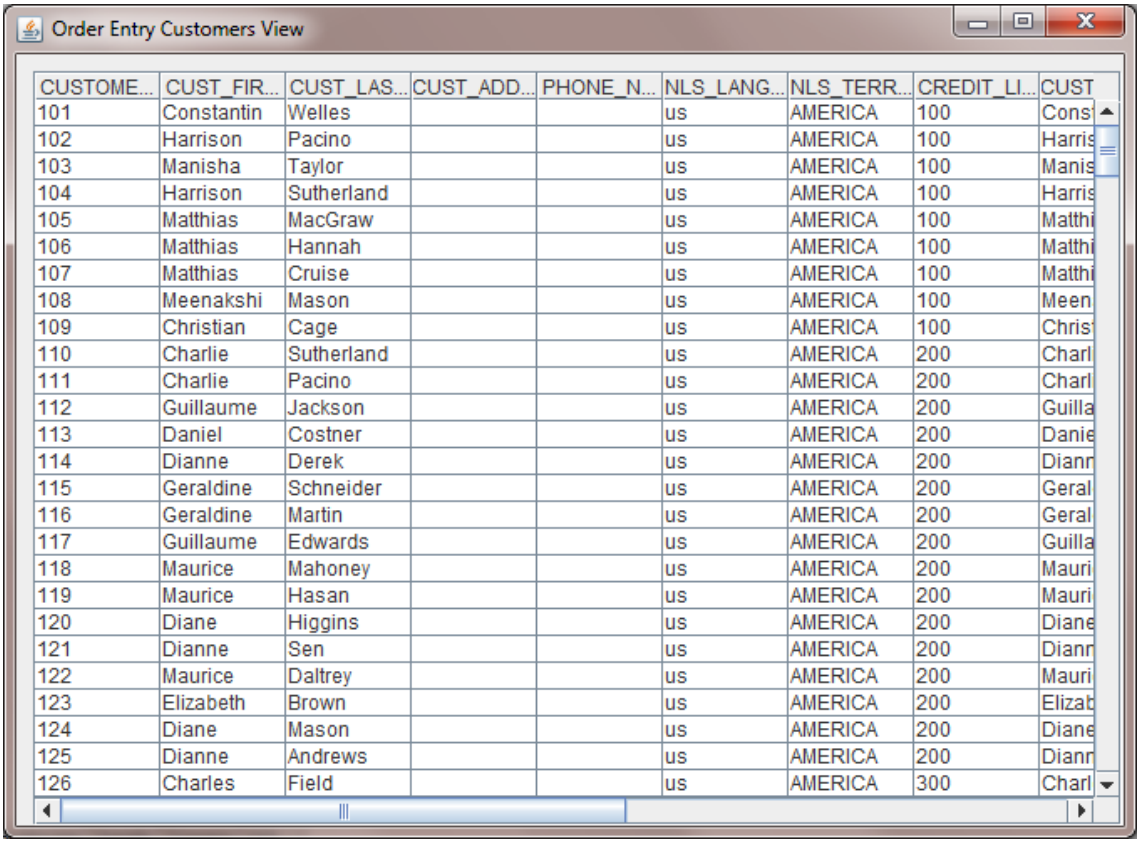
- Remove the `CUST_MARITAL_STATUS` and `CUST_INCOME_LEVEL` data from records in our archive, `SH.CUSTOMERS` table.

- For some reason, our purchasing department feels it is necessary to retain the `CUST_YEAR_OF_BIRTH` field in the `SH.CUSTOMERS` archive table, so we will need to encrypt it.
- For existing records in `SH.CUSTOMERS` we will remove the indicated sensitive data and encrypt the `CUST_YEAR_OF_BIRTH` data.
- We will also create an on-insert trigger to null and encrypt future data being stored in `SH.CUSTOMERS`.
- For our active data, the developers who maintain the application that currently reads and writes to the `OE.CUSTOMERS` table are willing to update the current application to select and update `CUSTOMERS` data using secure procedures, and to implement our new encrypted data transfer. They are willing to include our SSO, 2-Factor Authentication and Application Authorization processes, with our help.
- We will help the developers create the secure procedures and the Java code to execute them.

The Original Application

There is an existing application that we will need to assist in upgrading to use our security structures. The application consists of a table of customer records and a form for updating those records. (I realize this is unrealistic, but it is sufficient for the purposes of this exercise in the supplement.) Figures depicting the 2 application screens are shown below.

When the application user clicks on a row of the table, the form screen opens up and allows data from that customer record (row) to be edited. The **Save** button updates data in the table, and the **Cancel** button returns without updating.



CUSTOME...	CUST_FIR...	CUST_LAS...	CUST_ADD...	PHONE_N...	NLS_LANG...	NLS_TERR...	CREDIT_LI...	CUST
101	Constantin	Welles			us	AMERICA	100	Const
102	Harrison	Pacino			us	AMERICA	100	Harris
103	Manisha	Taylor			us	AMERICA	100	Manis
104	Harrison	Sutherland			us	AMERICA	100	Harris
105	Matthias	MacGraw			us	AMERICA	100	Matthi
106	Matthias	Hannah			us	AMERICA	100	Matthi
107	Matthias	Cruise			us	AMERICA	100	Matthi
108	Meenakshi	Mason			us	AMERICA	100	Meen
109	Christian	Cage			us	AMERICA	100	Chris
110	Charlie	Sutherland			us	AMERICA	200	Charl
111	Charlie	Pacino			us	AMERICA	200	Charl
112	Guillaume	Jackson			us	AMERICA	200	Guilla
113	Daniel	Costner			us	AMERICA	200	Danie
114	Dianne	Derek			us	AMERICA	200	Diann
115	Geraldine	Schneider			us	AMERICA	200	Geral
116	Geraldine	Martin			us	AMERICA	200	Geral
117	Guillaume	Edwards			us	AMERICA	200	Guilla
118	Maurice	Mahoney			us	AMERICA	200	Mauri
119	Maurice	Hasan			us	AMERICA	200	Mauri
120	Diane	Higgins			us	AMERICA	200	Diane
121	Dianne	Sen			us	AMERICA	200	Diann
122	Maurice	Daltrey			us	AMERICA	200	Mauri
123	Elizabeth	Brown			us	AMERICA	200	Elizab
124	Diane	Mason			us	AMERICA	200	Diane
125	Dianne	Andrews			us	AMERICA	200	Diann
126	Charles	Field			us	AMERICA	300	Charl

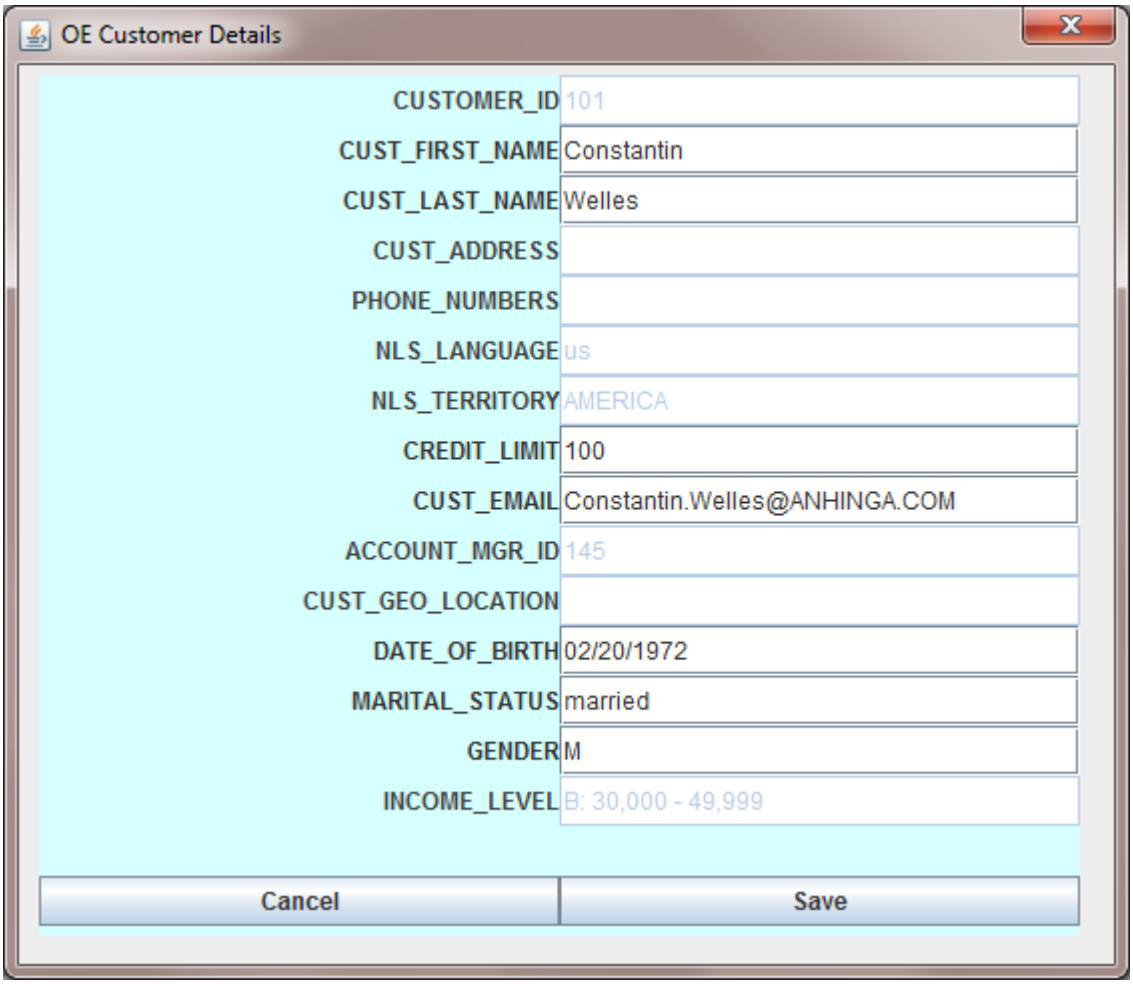


Table of Customers

Much of the code used to present the GUI table of customers is already familiar to us. Here is the code for the `jbInit()` and `dataInit()` methods that are called by the constructor.

In the `jbInit()` method, I only want to point out a couple features. First of all, setting the `AutoResizeMode` for the table to `AUTO_RESIZE_OFF` allows the columns to be set to a default, constant width. Otherwise, preferred widths should be defined for each column using the `TableColumnModel`. The second point of interest in `jbInit()` is the `mouseClicked()` event handler which calls the `oeViewTable_mouseClicked()` method.

```
    private void jbInit() throws Exception {
        this.getContentPane().setLayout(null);
```

```
        this.setSize(new Dimension(677, 498));
        this.setTitle("Order Entry Customers View");
        this.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    this_windowClosing(e);
                }
            });
        jScrollPane1.setBounds(new Rectangle(10, 10, 650, 450));
        jScrollPane1.getViewport().add(oeViewTable, null);
        this.getContentPane().add(jScrollPane1, null);
        oeViewTable.setModel(oeViewTM);
        oeViewTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        oeViewTable.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                    oeViewTable_mouseClicked(e);
                }
            });
    }
```

In the `dataInit()` method, you will see areas where we need to address application security. The first area of concern is the password used in the `getConnection()` method. Notice that everyone using the application is connecting as the `OE` schema owner account. We have each application user enter the password in a Dialog screen. That means that everyone who needs to use the old application also needs to know the OE schema account password. This password / account is used to get complete access to the schema to manage and design the data. We don't want the password to be widely known; rather, we need to protect it. The second area of concern is, of course, the query to acquire our sensitive data, e.g. `DATE_OF_BIRTH`, in clear text. The new PII protection policy directs us not to do that.

```
    void dataInit() {
        Statement stmt = null;
        ResultSet rs = null;
        try {
            if (conn == null) {
                String password =
                    JOptionPane.showInputDialog(this, "Enter OE Customers Password: ");
                conn =
(OracleConnection)DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
                                              "oe", password);
            }
            stmt = conn.createStatement();
            rs =
 stmt.executeQuery("SELECT CUSTOMER_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_ADDRESS, " +
                  "PHONE_NUMBERS, NLS_LANGUAGE, NLS_TERRITORY, CREDIT_LIMIT, CUST_EMAIL, "
+
                  "ACCOUNT_MGR_ID, CUST_GEO_LOCATION, TO_CHAR( DATE_OF_BIRTH, 'MM/DD/YYYY'
), MARITAL_STATUS, " +
                  "GENDER, INCOME_LEVEL FROM OE.customers ORDER BY CUSTOMER_ID");
            // dataVector must be Vector of Vectors
            Vector dataVector = new Vector();
            Vector itemVector;
            while (rs.next()) {
```

```
                itemVector = new Vector();
                itemVector.add(rs.getString(1));
                itemVector.add(rs.getString(2));
                itemVector.add(rs.getString(3));
                itemVector.add(rs.getString(4));
                itemVector.add(rs.getString(5));
                itemVector.add(rs.getString(6));
                itemVector.add(rs.getString(7));
                itemVector.add(rs.getString(8));
                itemVector.add(rs.getString(9));
                itemVector.add(rs.getString(10));
                itemVector.add(rs.getString(11));
                itemVector.add(rs.getString(12));
                itemVector.add(rs.getString(13));
                itemVector.add(rs.getString(14));
                itemVector.add(rs.getString(15));
                dataVector.add(itemVector);
            }
            if (rs != null)
                rs.close();
            oeViewTM.setDataVector(dataVector, columnIdentifiers);
        } catch (Exception x) {
            x.printStackTrace();
            JOptionPane.showMessageDialog(this, x.toString());
            this_windowClosing(null);
        } finally {
            try {
                if (stmt != null)
                    stmt.close();
            } catch (Exception y) {
            }
        }
    }
```

When the application user clicks on a row in the table, the mouse event is handled by a call to the `oeViewTable_mouseClicked()` method. In that method, we create a new instance of our form, `OEViewDialog` and hand it (via the `setValues()` method) this current table instance so the data from the currently selected row can be used to populate the form.

```
    private void oeViewTable_mouseClicked(MouseEvent e) {
        OEViewDialog mOEVD = new OEViewDialog();
        mOEVD.setValues(this);
    }
}
```

Customer Details

We will skip reviewing most of the code for the form, `OEViewDialog`. To populate the form, recall from the previous discussion, we hand the instance of the table view to the `OEViewDialog.setValues()`

method. In that method, each `JTextField` on the form is populated by getting the value of the currently selected table row, at the appropriate column. This example sets the value of the `CUSTOMER_ID` text field, `jTextField1` to the value from column 0 (zero) of the selected row in the table.

```
jTextField1.setText((String)parent.oeViewTable.getValueAt(
        parent.oeViewTable.getSelectedRow(), 0));
```

When we are done with the form, by clicking either the **Cancel** or **Save** button, we call the `this_windowClosing()` method. In that method, as is typical, we dispose of the form and make the table visible once again. It is convenient, so we also call the `dataInit()` method on the `OEView`, table class to refresh that data after a potential update.

```
    private void this_windowClosing(WindowEvent e) {
        parent.setVisible(true);
        parent.dataInit();
        this.dispose();
    }
```

The area of the form that causes us the most concern is the submission of data in the `saveButton_actionPerformed()` method, on two accounts. First of all, there is nothing in the form that restrains the application user from accomplishing SQL Injection. The statement simple concatenates data from the form fields into the update command, like this:

```
    stmt.executeUpdate("update OE.customers set CUST_FIRST_NAME='" +
        jTextField2.getText() + "'”
```

What if by malicious action, that text field perchance contained the following text. It is conceivable that with one trip through the form, a user might delete all our customer data.

```
    NoName’;delete from customers;--
```

We will want to correct that insecure programming approach; but our primary goal is to correct the transmission of sensitive data in clear text. For example, we are currently simply concatenating the sensitive fields, like marital_status into the update command like this:

```
    MARITAL_STATUS='" + jTextField13.getText() + "'”
```

Running the old application

Before running the old application, you will need to compile the code. Here are the commands you will use to compile and run:

```
    cd Supplement\OEView
    javac –cp .;ojdbc6.jar com\org\oeview\OEView.java
    java –cp .;ojdbc6.jar com.org.oeview.OEView
```

Security Administrator Script

The assumption here, for those following this supplemental section, is that you have the Oracle sample schemas installed on your database instance. One of the sample schemas is named OE, which stands for Order Entry. That is where our sensitive data is located. Another associated sample schema is named SH, which stands for Sales History. SH holds an archive of the live customer data in OE.

Both of these sample schemas may be installed by default, or after the fact, in an Oracle 11g instance. When they are installed, the schema owner user account is disabled. We are going to enable those accounts and give them a password. We accomplish that as the Security Administrator, secadm user with the commands listed below.

We also create a user, oeview who will access the OE data in our applications – a "one big application user" approach (change the password for oeview to something complex). The privileges for access will be granted to a role, oeview_role which is a secure application role identified by our p_check_role_access procedure. Any user who can SSO and 2-factor authenticate to Oracle must also be granted the privilege to proxy through the oeview user in order to run our applications.

Note: The following commands can be found in the script file, Supplement/OEView/SecAdm.sql. You may have already executed several of these commands if you ran code to implement the testc application, described in a previous section.

```
EXEC sys.p_check_secadm_access;

-- From here, may have already been accomplished for TestAppC in Supplement
ALTER USER oe ACCOUNT UNLOCK;
ALTER USER oe IDENTIFIED BY password;

GRANT create_session_role TO oeview IDENTIFIED BY password;

CREATE ROLE oeview_role IDENTIFIED USING appsec.p_check_role_access;
-- To here

GRANT EXECUTE ON appsec.app_sec_pkg TO oe;

ALTER USER sh ACCOUNT UNLOCK;
ALTER USER sh IDENTIFIED BY password;

GRANT EXECUTE ON appsec.f_mask TO sh;
GRANT EXECUTE ON appsec.f_unmask TO sh;
```

Take a look at the grants in the script above. We grant the OE application schema user the right to execute procedures and functions in our app_sec_pkg. With this privilege, OE can incorporate data encryption over the network into procedures used for accessing the OE data. We will build those secure procedures a bit later. Additionally, we grant the privilege to execute f_mask and f_unmask functions to

the SH schema user. SH will be encrypting data for storage in the archive tables and will use our appsec functions to accomplish encryption.

Cleaning and Encrypting Archive Records

Our first order of business is to clean up the archive records. Removing or encrypting the sensitive data in our primary table is, as you will see, a simple matter. To accomplish due diligence in this respect, we will have to clean (purge or destroy) any copies or reflections of the sensitive of the data. We need to look in these places:

- Personal schemas and hard drives of data owners and users
- Oracle redo logs
- Oracle archive logs
- Replica, acceptance and development Oracle instances
- On-line and archive backups, both on-site and remote

This search and destroy task will not be easy, and should serve as a reminder to avoid storing potentially sensitive data.

Now the task at hand is to remove the sensitive data from our on-line archive table, SH.CUSTOMERS; and in the case of the customer year of birth, store it in encrypted form. The encrypted column will be a different type from the original year of birth column, so we will add a column, cust_yob_encr to the archive table to hold the encrypted data. Accomplish these commands as the SH schema user.

```
-- Add column to customers table to hold encrypted cust_year_of_birth
ALTER TABLE SH.customers
add
   cust_yob_encr RAW(2000);
```

Note: This code can be found in the script file *Supplement/OEView/SH.sql*

Now to automatically expunge sensitive data from records being inserted or updated in our archive table, we will create a trigger. Our assurance is that once this trigger is created and enabled, no unencrypted sensitive data will exist in the archive. We will create a BEFORE INSERT OR UPDATE trigger on SH.CUSTOMERS to replace the values being inserted / updated for marital_status and income_level with null strings. We also call our wrapped encryption function, appsec.f_mask to encrypt the value provided for cust_year_of_birth. We set the encrypted value for the new column, cust_yob_encr and replace the value for cust_year_of_birth with a zero.

```
-- Create trigger to expunge future archive records
CREATE OR REPLACE TRIGGER SH.t_customers_biur BEFORE INSERT OR UPDATE
```

```
    ON SH.customers FOR EACH ROW
BEGIN
    :new.cust_marital_status := '';
    :new.cust_income_level := '';
    :new.cust_yob_encr := appsec.f_mask(
        SYS.UTL_RAW.CAST_TO_RAW(TO_CHAR(:new.cust_year_of_birth)||'SufficientLength'),
        :new.cust_last_name,
        :new.cust_first_name );
    :new.cust_year_of_birth := 0;
END;
/

ALTER TRIGGER SH.t_customers_biur ENABLE;
```

You might recall that our `f_mask` function does not do cookie-cutter encryption of the field in question, but makes each encryption unique to other related aspects of the record. In our Application Verification tasks, we encrypt with a reference to the application ID and the inner class name. Here, we are encrypting specifically in relation to the `cust_last_name` and `cust_first_name` values.

It is always a good plan to test our code. The next block of code inserts a record then examines the results of the trigger. After that examination, the test record should be deleted.

```
-- Test update trigger
INSERT INTO SH.customers
( CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_GENDER, CUST_YEAR_OF_BIRTH,
    CUST_MARITAL_STATUS, CUST_STREET_ADDRESS, CUST_POSTAL_CODE, CUST_CITY,
    CUST_CITY_ID, CUST_STATE_PROVINCE, CUST_STATE_PROVINCE_ID, COUNTRY_ID,
    CUST_MAIN_PHONE_NUMBER, CUST_INCOME_LEVEL, CUST_CREDIT_LIMIT, CUST_EMAIL,
    CUST_TOTAL, CUST_TOTAL_ID, CUST_SRC_ID, CUST_EFF_FROM, CUST_EFF_TO, CUST_VALID )
VALUES
( 200000, 'David', 'Coffin', 'M', 1999, 'Married', '1212 Second St.',
    '68524', 'Glasco', 51583, 'KS', 52630, 52790, '800-555-1212',
    'K: 250,000 - 299,999', 15000, 'coffin@org.com', 'Customer total',
    52772, NULL, SYSDATE, NULL, 'A' );

COMMIT;

SELECT * FROM SH.customers WHERE cust_id = 200000;
-- Observe the values of cust_marital_status, cust_income_level,
--  cust_year_of_birth and cust_yob_encr columns

DELETE FROM SH.customers WHERE cust_id = 200000;

COMMIT;
```

They say the shortest distance between two points is a straight line, except in the novel A Wrinkle in Time. The straight line for us to get all our archive records cleaned up will be to apply our trigger to each row. We can do that with an anonymous PL/SQL block that updates each row. Execute the following code to accomplish this task, and be aware that it might run for a little while.

Note: This task places the sensitive data into the redo logs which will have to be separately cleansed.

```
-- Apply trigger to all existing records
-- This takes a few minutes!  Only run this once!
DECLARE
  TYPE CustCurTyp  IS REF CURSOR;
  v_cust_cursor    CustCurTyp;
  cust_record      SH.customers%ROWTYPE;
  v_stmt_str       VARCHAR2(200);
BEGIN
  v_stmt_str := 'SELECT * FROM SH.customers';
  OPEN v_cust_cursor FOR v_stmt_str;
  v_stmt_str := 'UPDATE SH.customers set cust_year_of_birth = :y where cust_id = :i';
  LOOP
    FETCH v_cust_cursor INTO cust_record;
    EXECUTE IMMEDIATE v_stmt_str USING cust_record.cust_year_of_birth, cust_record.cust_id;
    EXIT WHEN v_cust_cursor%NOTFOUND;
  END LOOP;
  CLOSE v_cust_cursor;
END;
/
```

One notable aspect of that block of code that we have not explored previously is the use of the `ROWTYPE` variable, `cust_record`. Using the `ROWTYPE` saves us having to handle each column individually. Notice that we are setting the value for `cust_year_of_birth` equal to the same value we read from the archive table. We are counting on the trigger we created to encrypt the year of birth data and to remove the sensitive data from each record.

We can observe the resultant records by selecting from the table. And if we are satisfied with the cleansed results, we should commit the transaction.

```
SELECT * FROM SH.customers;
-- Observe the values of cust_marital_status, cust_income_level,
--  cust_year_of_birth and cust_yob_encr columns

COMMIT;
```

For our own reference sake, the following view selects from the `SH.CUSTOMERS` table, decrypting the `cust_yob_encr` column and returning it as `cust_year_of_birth`. This code is perhaps something that your marketing department needs for reporting or analysis; but if not, it would be best to not define the view. This view is like a key to decrypt the sensitive data.

```
CREATE OR REPLACE FORCE VIEW SH.v_customer_detail_sensitive
    AS SELECT
        CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_GENDER,
        to_number(
            substr(
```

```
                SYS.UTL_RAW.CAST_TO_varchar2(
                    appsec.f_unmask(
                        cust_yob_encr,
                        cust_last_name,
                        cust_first_name
                    )
                ),
                1,
                4
            )
        ) CUST_YEAR_OF_BIRTH,
        CUST_MARITAL_STATUS, CUST_STREET_ADDRESS, CUST_POSTAL_CODE, CUST_CITY,
        CUST_CITY_ID, CUST_STATE_PROVINCE, CUST_STATE_PROVINCE_ID, COUNTRY_ID,
        CUST_MAIN_PHONE_NUMBER, CUST_INCOME_LEVEL, CUST_CREDIT_LIMIT, CUST_EMAIL,
        CUST_TOTAL, CUST_TOTAL_ID, CUST_SRC_ID, CUST_EFF_FROM, CUST_EFF_TO, CUST_VALID
    FROM SH.customers;

-- Do not grant this view to anyone - only use internally

SELECT * FROM SH.v_customer_detail_sensitive;
-- Observe the values of cust_year_of_birth
```

Secure Oracle Structures for Order Entry Customers Table

To our knowledge, there is not a view of the customer data that returns the sensitive columns. We are going to create a placeholder for that (perhaps default) view, `v_customer_detail` that actually returns null strings or the number 0 (zero) instead of the sensitive data.

```
-- Swap null for DATE_OF_BIRTH, MARITAL_STATUS and INCOME_LEVEL
CREATE OR REPLACE FORCE VIEW OE.v_customer_detail
    AS SELECT
                customer_id,
                cust_first_name,
                cust_last_name,
                cust_address,
                phone_numbers,
                nls_language,
                nls_territory,
                credit_limit,
                cust_email,
                account_mgr_id,
                cust_geo_location,
        null    date_of_birth,
        ''      marital_status,
                gender,
        ''      income_level
    FROM OE.customers;

SELECT * FROM oe.v_customer_detail ;
```

We envision access to the CUSTOMERS table being provided in general through a view that simply excludes the sensitive columns. The following definition of `v_customer_detail_public` is what we intend to provide unless specific justification for access to the sensitive data is produced. We grant access to this public-data view to the `oeview_role` role. We will ask developers who are currently accessing data through the customers table to use this view instead. If they need to continue querying for the sensitive fields but can live with null values instead of the sensitive data, then we would give them access to the `v_customer_detail` view, defined above.

```
CREATE OR REPLACE FORCE VIEW OE.v_customer_detail_public
    AS SELECT
        customer_id,
        cust_first_name,
        cust_last_name,
        cust_address,
        phone_numbers,
        nls_language,
        nls_territory,
        credit_limit,
        cust_email,
        account_mgr_id,
        cust_geo_location,
        gender
    FROM OE.customers;

SELECT * FROM oe.v_customer_detail_public ;

GRANT SELECT ON OE.v_customer_detail_public TO oeview_role;
```

Another view that we intend to reserve for special access circumstances will give access to all the data columns, both sensitive and public, the `v_customer_detail_sensitive` view. Primarily, we will be using this view in our secure data access procedures that will acquire the data and encrypt the sensitive columns before transmission to the client application. Our preference is to provide access through a view rather than directly through the table. That way we can redefine the table as needed without breaking the accessing applications, because the view definition will remain consistent even if it accesses the data differently.

```
CREATE OR REPLACE FORCE VIEW OE.v_customer_detail_sensitive
    AS SELECT * FROM OE.customers;

SELECT * FROM oe.v_customer_detail_sensitive ;

-- No grants to v_customer_detail_sensitive, use it internally only
```

Just as we saw for secure `HR` data access in Chapter 7 of the book, we will create a package in the `OE` schema that will provide procedures to query the sensitive data and update it, all the while passing the sensitive data in encrypted form. In the `OE` schema, we will call this package `oe_sec_pkg`. It will have, for this exercise, two procedures: `p_select_customers_sensitive` and `p_update_customers_sensitive`. These are analogous to the procedures `p_select_employees_sensitive` and `p_update_employees_sensitive` that we saw in Chapter 7.

The code for the package body is shown below. The first procedure in the package, p_select_customers_sensitive returns the customer data. As in the book, this procedure takes the locally generated RSA public key modulus and exponent, then calculates the DES shared password key and returns the artifacts of that DES key. It also returns a result set / CURSOR with all the public data columns and the sensitive columns in encrypted form.

```
CREATE OR REPLACE PACKAGE BODY oe.oe_sec_pkg IS

    PROCEDURE p_select_customers_sensitive(
        ext_modulus               VARCHAR2,
        ext_exponent              VARCHAR2,
        secret_pass_salt      OUT RAW,
        secret_pass_count     OUT RAW,
        secret_pass_algorithm OUT RAW,
        secret_pass           OUT RAW,
        resultset_out         OUT RESULTSET_TYPE,
        m_err_no              OUT NUMBER,
        m_err_txt             OUT VARCHAR2 )
    IS BEGIN
        m_err_no := 0;
        secret_pass_salt :=
            appsec.app_sec_pkg.f_get_crypt_secret_salt( ext_modulus, ext_exponent );
        secret_pass_count :=
            appsec.app_sec_pkg.f_get_crypt_secret_count( ext_modulus, ext_exponent );
        secret_pass :=
            appsec.app_sec_pkg.f_get_crypt_secret_pass( ext_modulus, ext_exponent );
        secret_pass_algorithm :=
            appsec.app_sec_pkg.f_get_crypt_secret_algorithm(ext_modulus, ext_exponent);
        OPEN resultset_out FOR SELECT
            CUSTOMER_ID,
            CUST_FIRST_NAME,
            CUST_LAST_NAME,
            CUST_ADDRESS,
            PHONE_NUMBERS,
            NLS_LANGUAGE,
            NLS_TERRITORY,
            CREDIT_LIMIT,
            CUST_EMAIL,
            ACCOUNT_MGR_ID,
            CUST_GEO_LOCATION,
            appsec.app_sec_pkg.f_get_crypt_data( TO_CHAR( DATE_OF_BIRTH, 'MM/DD/YYYY' ) ),
            appsec.app_sec_pkg.f_get_crypt_data( MARITAL_STATUS ),
            GENDER,
            appsec.app_sec_pkg.f_get_crypt_data( INCOME_LEVEL )
        FROM OE.v_customer_detail_sensitive ORDER BY CUSTOMER_ID;
    EXCEPTION
        WHEN OTHERS THEN
            m_err_no := SQLCODE;
            m_err_txt := SQLERRM;
            appsec.app_sec_pkg.p_log_error( m_err_no, m_err_txt,
                'OE p_select_customers_sensitive' );
```

```
END p_select_customers_sensitive;
```

The second procedure, `p_update_customers_sensitive` takes all the data columns from the client for insert or update into the `OE.CUSTOMERS` table. The sensitive data is transmitted in encrypted form and will be decrypted before storage in the database table. We select records matching this customer ID, and if there are none, we insert a new record; else we update the existing record.

```
PROCEDURE p_update_customers_sensitive(
    m_customer_id           customers.customer_id%TYPE,
    m_cust_first_name       customers.cust_first_name%TYPE,
    m_cust_last_name        customers.cust_last_name%TYPE,
    --m_cust_address          customers.cust_address%TYPE,
    --m_phone_numbers         customers.phone_numbers%TYPE,
    --m_nls_language          customers.nls_language%TYPE,
    --m_nls_territory         customers.nls_territory%TYPE,
    m_credit_limit          customers.credit_limit%TYPE,
    m_cust_email            customers.cust_email%TYPE,
    --m_account_mgr_id        customers.account_mgr_id%TYPE,
    --m_cust_geo_location     customers.cust_geo_location%TYPE,
    crypt_date_of_birth     RAW,
    crypt_marital_status    RAW,
    m_gender                customers.gender%TYPE,
    --crypt_income_level      RAW,
    m_err_no            OUT NUMBER,
    m_err_txt           OUT VARCHAR2 )
IS
    test_cust_ct     NUMBER(6);
    v_date_of_birth  customers.date_of_birth%TYPE;
    v_marital_status customers.marital_status%TYPE;
    v_income_level   customers.income_level%TYPE;
BEGIN
    -- Note:  Use of this procedure assumes you have already done a select
    -- and that you are using the same Session Secret PassPhrase
    m_err_no := 0;
    v_date_of_birth := TO_DATE(
        appsec.app_sec_pkg.f_get_decrypt_data( crypt_date_of_birth ),
        'MM/DD/YYYY' );
    v_marital_status :=
        appsec.app_sec_pkg.f_get_decrypt_data( crypt_marital_status );
    --v_income_level :=
    --    appsec.app_sec_pkg.f_get_decrypt_data( crypt_income_level );
    SELECT COUNT(*) INTO test_cust_ct FROM OE.v_customer_detail_sensitive WHERE
        customer_id = m_customer_id;
    IF test_cust_ct = 0
    THEN
        -- Insert will not be called, since only updating existing
        INSERT INTO OE.v_customer_detail_sensitive
            ( CUSTOMER_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_ADDRESS,
                PHONE_NUMBERS, NLS_LANGUAGE, NLS_TERRITORY, CREDIT_LIMIT,
                CUST_EMAIL, ACCOUNT_MGR_ID, CUST_GEO_LOCATION,
                DATE_OF_BIRTH, MARITAL_STATUS, GENDER, INCOME_LEVEL )
```

```
            VALUES
                --( m_customer_id, m_cust_first_name, m_cust_last_name, m_cust_address,
                --m_phone_numbers, m_nls_language, m_nls_territory, m_credit_limit,
                --m_cust_email, m_account_mgr_id, m_cust_geo_location,
                --v_date_of_birth, v_marital_status, m_gender, v_income_level );
                ( m_customer_id, m_cust_first_name, m_cust_last_name, null,
                null, null, null, m_credit_limit,
                m_cust_email, null, null,
                v_date_of_birth, v_marital_status, m_gender, null );
        ELSE
            UPDATE OE.v_customer_detail_sensitive
            SET CUST_FIRST_NAME = m_cust_first_name,
                CUST_LAST_NAME = m_cust_last_name,
                --CUST_ADDRESS = m_cust_address,
                --PHONE_NUMBERS = m_phone_numbers,
                --NLS_LANGUAGE = m_nls_language,
                --NLS_TERRITORY = m_nls_territory,
                CREDIT_LIMIT = m_credit_limit,
                CUST_EMAIL = m_cust_email,
                --ACCOUNT_MGR_ID = m_account_mgr_id,
                --CUST_GEO_LOCATION = m_cust_geo_location,
                DATE_OF_BIRTH = v_date_of_birth,
                MARITAL_STATUS = v_marital_status,
                GENDER = m_gender --,
                --INCOME_LEVEL = v_income_level
            WHERE customer_id = m_customer_id;
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
            m_err_no := SQLCODE;
            m_err_txt := SQLERRM;
            appsec.app_sec_pkg.p_log_error( m_err_no, m_err_txt,
                'OE p_update_customers_sensitive' );
    END p_update_customers_sensitive;

END oe_sec_pkg;
/

GRANT EXECUTE ON oe.oe_sec_pkg TO oeview_role;
```

We are not attempting to update certain columns of the customer data. Any columns that are calculated or based on foreign keys, we just leave as is. We grant execute on this new package to the `oeview_role` role, which will be acquired by all users of this application who have passed our authentication requirements and proxied through the `oeview` user.

The Secured Application

Now we will look at the changes to the existing application that are required in order to comply with our secure computing initiatives and policies. We will not be adding much in the way of functionality, but

will be adding the features of application authentication and security that we have built into `OracleJavaSecure` and in the Oracle structures, described above.

Implementing OracleJavaSecure

The first modification we want to make to the existing application is to incorporate `OracleJavaSecure` features. In order to do that, we have to make `OracleJavaSecure` available to the **OEView** application. We will package `OracleJavaSecure`, as described in Chapter 7, *Java Archive for Use by Applications*. The jar file (named for example, OJS.jar) will include at least `OracleJavaSecure.class`, `RevLvlClassIntfc.class` and `OJSC.class`.

Then for application-specific use, we need a copy of `Login` for this application. Copy *Login.java* from Chapter12, and change the package name in that file to be the same as for `OEView` -- *com.org.oeview*. Also change the application ID in *Login.java* to be "OEVIEW".

Changes for Table of Customers

Make a copy of *OEView.java*, and rename it *OEView2.java*. You will be editing *OEView2.java* to build the replacement, secure application. In order to use our security enhancements, you will need to import `OracleJavaSecure`, like this:

```
import orajavsec.OracleJavaSecure;

public class OEView2 extends JFrame {
    JFrame thisComponent = this;
```

Also, for reference later from within extraneous threads, we need a reference to the current instance of `OEView2`. Create a class member, `thisComponent` to hold that reference.

Of course, your `main()` method and the constructor need to refer to this class by its new name, `OEView2`.

```
    public static void main(String[] args) {
        new OEView2();
    }

    public OEView2() {
        try {
            jbInit();
            dataInit();
            this.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

In the `dataInit()` method, we will tap into the features of `Login` and `OracleJavaSecure`. We will, simply, remove our previous `Dialog` request for the `OE` password and new `Connection`, and replace it with the instantiation of the `Login` class and a call to `getAAConnRole()`.

```
    void dataInit() {
        Statement stmt = null;
        ResultSet rs = null;
        try {
            if (conn == null) {
                new Login(this);
                Login.center(this);
                conn = OracleJavaSecure.getAAConnRole("orcl", "oeview");
            }
```

We also replace our simple, clear-text query with a call to our new secure procedure, `p_select_customers_sensitive`, shown below. Notice how we also wrap this call in a delayed thread using the `SwingUtilities.invokeLater()` method, as described in Chapter 12. See where we pass the RSA public key modulus and exponent to the procedure?

```
            SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        int errNo;
                        String errMsg;
                        OracleCallableStatement stmt = null;
                        OracleResultSet rs = null;
                        try {
                            stmt =
(OracleCallableStatement)conn.prepareCall("CALL
oe.oe_sec_pkg.p_select_customers_sensitive(?,?,?,?,?,?,?,?,?)");
                            stmt.registerOutParameter(3, OracleTypes.RAW);
                            stmt.registerOutParameter(4, OracleTypes.RAW);
                            stmt.registerOutParameter(5, OracleTypes.RAW);
                            stmt.registerOutParameter(6, OracleTypes.RAW);
                            stmt.registerOutParameter(7, OracleTypes.CURSOR);
                            stmt.registerOutParameter(8, OracleTypes.NUMBER);
                            stmt.registerOutParameter(9, OracleTypes.VARCHAR);
                            stmt.setString(1,
                                          OracleJavaSecure.getLocRSAPubMod());
                            stmt.setString(2,
                                          OracleJavaSecure.getLocRSAPubExp());
                            stmt.setNull(3, OracleTypes.RAW);
                            stmt.setNull(4, OracleTypes.RAW);
                            stmt.setNull(5, OracleTypes.RAW);
                            stmt.setNull(6, OracleTypes.RAW);
                            stmt.setInt(8, 0);
                            stmt.setNull(9, OracleTypes.VARCHAR);
                            stmt.executeUpdate();

                            errNo = stmt.getInt(8);
                            if (errNo != 0) {
                                errMsg = stmt.getString(9);
                                JOptionPane.showMessageDialog(thisComponent,
                                    "Oracle error p_select_employee_by_id_sens) " +
                                                                errNo + ", " +
```

```
                                                    errMsg);
                            } else {
                                rs = (OracleResultSet)stmt.getCursor(7);

                                // Following code from original, with decryption added
                                // dataVector must be Vector of Vectors
                                Vector dataVector = new Vector();
                                Vector itemVector;
                                while (rs.next()) {
                                    itemVector = new Vector();
                                    itemVector.add(rs.getString(1));
                                    itemVector.add(rs.getString(2));
                                    itemVector.add(rs.getString(3));
                                    itemVector.add(rs.getString(4));
                                    itemVector.add(rs.getString(5));
                                    itemVector.add(rs.getString(6));
                                    itemVector.add(rs.getString(7));
                                    itemVector.add(rs.getString(8));
                                    itemVector.add(rs.getString(9));
                                    itemVector.add(rs.getString(10));
                                    itemVector.add(rs.getString(11));
                                    //itemVector.add(rs.getString(12));
itemVector.add(OracleJavaSecure.getDecryptData(rs.getRAW(12),
                                        stmt.getRAW(6),
                                        stmt.getRAW(5),
                                        stmt.getRAW(3),
                                        stmt.getRAW(4)));
                                    //itemVector.add(rs.getString(13));
itemVector.add(OracleJavaSecure.getDecryptData(rs.getRAW(13),
                                        stmt.getRAW(6),
                                        stmt.getRAW(5),
                                        stmt.getRAW(3),
                                        stmt.getRAW(4)));
                                    itemVector.add(rs.getString(14));
                                    //itemVector.add(rs.getString(15));
itemVector.add(OracleJavaSecure.getDecryptData(rs.getRAW(15),
                                        stmt.getRAW(6),
                                        stmt.getRAW(5),
                                        stmt.getRAW(3),
                                        stmt.getRAW(4)));
                                    dataVector.add(itemVector);
                                }
                                if (rs != null)
                                    rs.close();
                                oeViewTM.setDataVector(dataVector,
                                                       columnIdentifiers);
                            }

                        } catch (Exception x) {
                            System.out.println(x.toString());
                        } finally {
```

```
                        try {
                            if (stmt != null)
                                stmt.close();
                            } catch (Exception y) {
                            }
                    }
                    Login.sayWaitDialog.setVisible(false);
                }
            });
        // It may take a while to get the user data, esp while we set up encryption
        // So ask the user to be patient while working
        Login.sayWaitDialog.setVisible(true);
```

In the result set / `CURSOR` that is returned, notice that the sensitive data is being returned as encrypted `RAW` type. For each of those values, we call the `OracleJavaSecure.getDecryptData()` method to decrypt the data. Instead of creating local members for each attribute of the DES secret password key, we pass the `RAW` values for those attributes directly to the `getDecryptData()` method. In this new code, we also use the "Wait While Processing" modal `Dialog` screen from `Login` to ask the user to be patient.

Changes for Customer Details

We also copy and rename the class used for data update, the form class. We will now call it `OEViewDialog2`. It also has to import `OracleJavaSecure` so we can use those features.

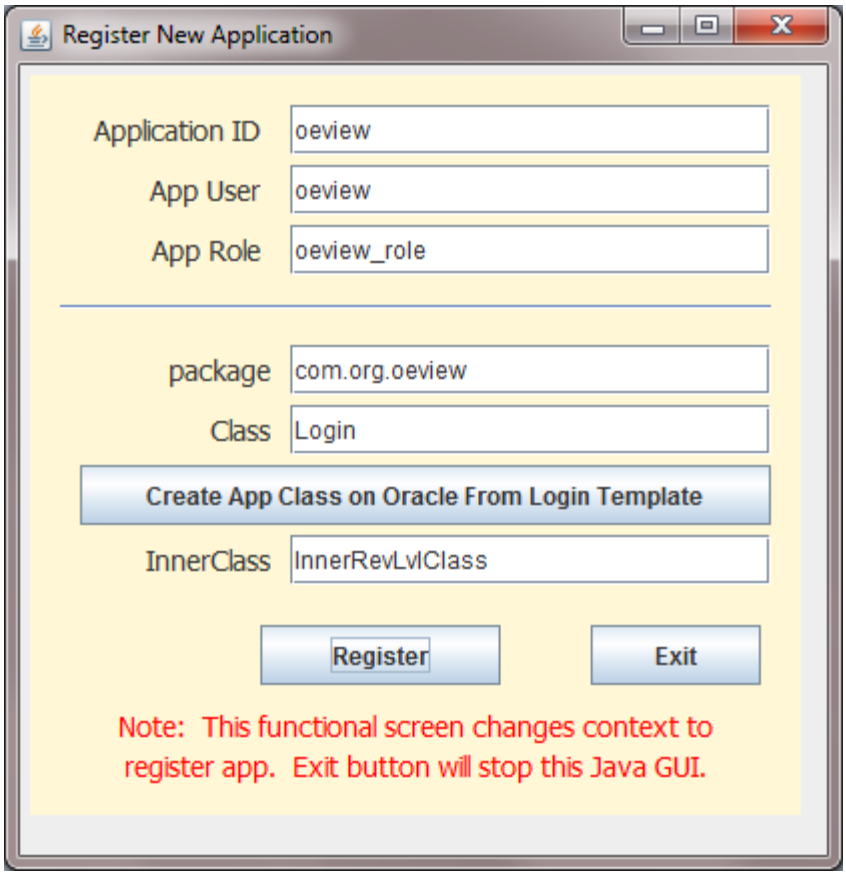
There is nothing much new in this procedure until we get to the data update command in the `saveButton_actionPerformed()` method. In the updated code, we now call the `p_update_customers_sensitive` procedure to accomplish the encrypted data transmission. It looks pretty ordinary since key exchange must have already taken place, except that the sensitive fields will be submitted in encrypted form as `RAW` types.

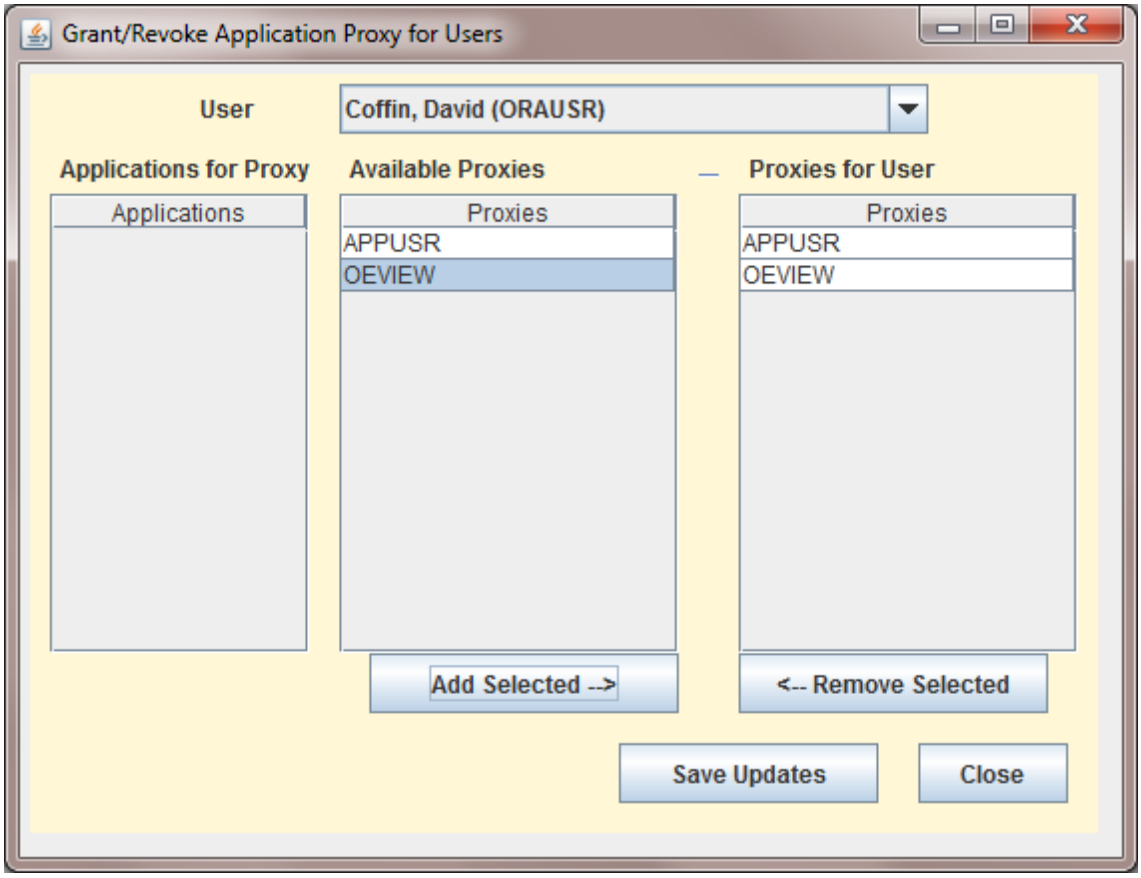
```
        // Call new procedures in oe.oe_sec_pkg to get and set data.
        SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    int errNo;
                    String errMsg;
                    OracleCallableStatement stmt = null;
                    try {
                        stmt =
(OracleCallableStatement)parent.conn.prepareCall("CALL
oe.oe_sec_pkg.p_update_customers_sensitive(?,?,?,?,?,?,?,?,?,?)");
                        stmt.registerOutParameter(9, OracleTypes.NUMBER);
                        stmt.registerOutParameter(10, OracleTypes.VARCHAR);
                        stmt.setString(1, jTextField1.getText());
                        stmt.setString(2, jTextField2.getText());
                        stmt.setString(3, jTextField3.getText());
                        stmt.setString(4, jTextField8.getText());
                        stmt.setString(5, jTextField9.getText());
                        stmt.setRAW(6,
                                OracleJavaSecure.getCryptData(jTextField12.getText()));
                        stmt.setRAW(7,
```

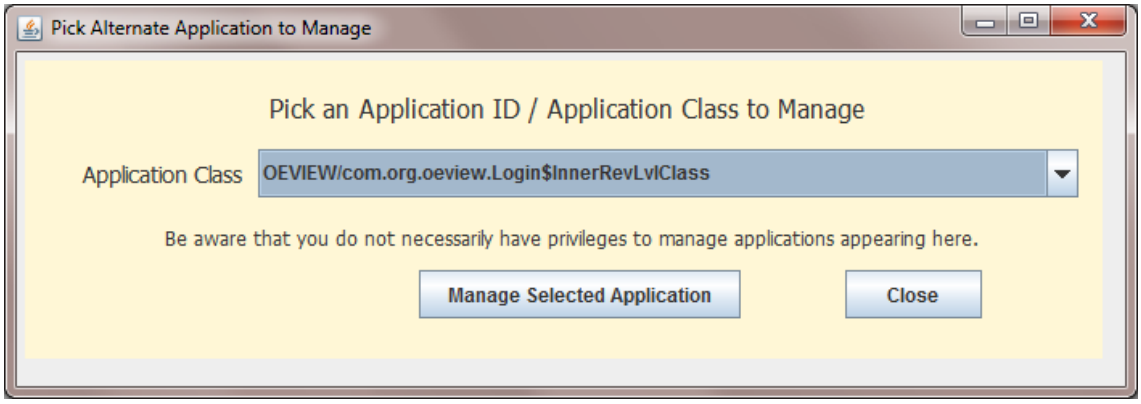
```
                                    OracleJavaSecure.getCryptData(jTextField13.getText()));
                        stmt.setString(8, jTextField14.getText());
                        stmt.setInt(9, 0);
                        stmt.setNull(10, OracleTypes.VARCHAR);
                        stmt.executeUpdate();
                        errNo = stmt.getInt(9);
                        if (errNo != 0) {
                            errMsg = stmt.getString(10);
                            JOptionPane.showMessageDialog(thisComponent,
                                "Oracle error p_update_customers_sensitive) " +
                                errNo + ", " + errMsg);
                        }
                    } catch (Exception x) {
                        JOptionPane.showMessageDialog(thisComponent,
                                                      x.toString());
                    } finally {
                        try {
                            if (stmt != null)
                                stmt.close();
                        } catch (Exception y) {
                        }
                    }
                    orajavsec.Login.sayWaitDialog.setVisible(false);
                    this_windowClosing(null);
                }
            });
        // Ask the user to be patient while working
        orajavsec.Login.sayWaitDialog.setVisible(true);
    }
}
```

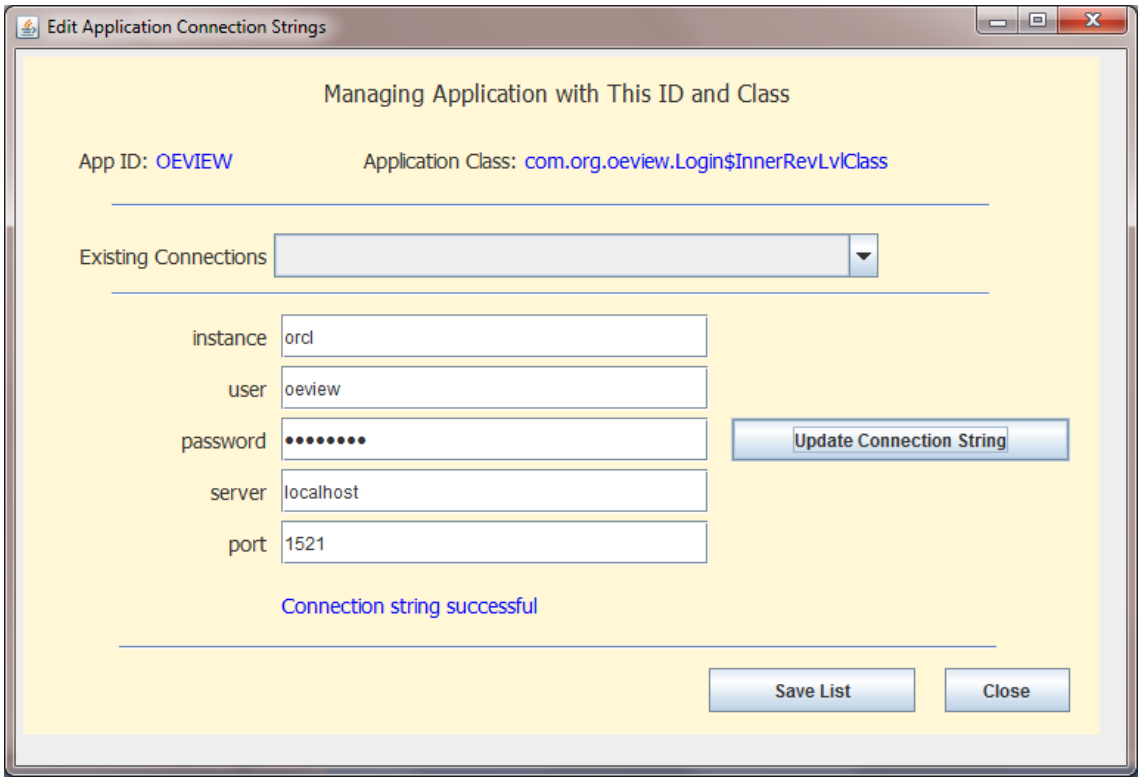
Registering Secured Application

The following screen captures show what must be configured to use this new app. Refer to Chapter 12 for instructions on running and using the `OJSAdmin` application


Register New Application
Application ID
oeview
App User
oeview
App Role
oeview_role
package
com.org.oeview
Class
Login
Create App Class on Oracle From Login Template
InnerClass
InnerRevLvlClass
Register
Exit
Note: This functional screen changes context to register app. Exit button will stop this Java GUI.


Grant/Revoke Application Proxy for Users
User
Coffin, David (ORAUSR)
Applications for Proxy
Available Proxies
Proxies for User
Applications
Proxies
APPUSR
OEVIEW
Proxies
APPUSR
OEVIEW
Add Selected -->
<-- Remove Selected
Save Updates
Close


Pick Alternate Application to Manage
Pick an Application ID / Application Class to Manage
Application Class
OEVIEW/com.org.oeview.Login$InnerRevLvlClass
Be aware that you do not necessarily have privileges to manage applications appearing here.
Manage Selected Application
Close



Asset-Centric Risk Assessment

This section is still in a state of development. My idea here is to show fences of security that we have erected around the data that we are protecting, specifically `HR.EMPLOYEES.SALARY`. If we can itemize the access points and assure proper and sufficient access control and security for this column, then all identically protected data will be assured of the same protection.

My quandary is in how to show this. At first I was thinking of using a table, with cross-references. But I'm considering the value of using text (sentences and paragraphs). Right now, what I've got is a beginning list, not really complete.

*Employee Salary
who can see
sys mitigate complex pwd, guard pwd, change pwd, (password security) also guard hrdwr
dbas pwd security, audit activities
hr schema pwd security, audit grants and access
employees table / views (including v_employees_sensitve_view) audit grants and access
legacy hr applications evaluate and remediate security, as required

thieves backups and archives
appsec-role (by grant on any table) guard grant, grantee pwd security
network capture encrypted on network
*hr_sec_pkg guard grant execute

network capture
who can read
only current session, encryption

hr_sec_pkg
who can execute
*hrview_role identified by procedure, p_get_role_somthing

hrview_role
who can acquire
appusr on network by oracle user proxy by sso by 2-factor ????? (by p_get_role_somthing)

appusr
who has user access
*those with password pwd security
*those with proxy grant
*those with application authorization

Passwords
who can acquire
encrypted over network
encrypted on client

proxy appusr
who has proxy grant
*those granted in database, further identified by sso and 2-factor

sso
who can
oracle users equal to os users,

2-factor
who can
sso user with mobile device

application authorization
who can
those with appusr proxy grant
must produce app inner class
must exchange encryption keys.

General Maintenance

Monitor and Research Auditing and Logging

note - Logging entries may be slightly out of order, since procedure is autonomous
audit failed attempts to login as appver or to login trigger

Most of the auditing we set was already existing as part of the default settings for Oracle 11g.

Change All Passwords

Remove remote SYSDBA password file – require connect from local prompt.
use sqlplus password command as opposed to alter user - protects password at command prompt and across the network
appver user password
encode appver password and paste into OracleJavaSecure, compile and distribute the jar
Oracle application user passwords
passwords in application connection strings
database link passwords
Oracle security administrator user passwords

There is also new data dictionary view DBA_USERS_WITH_DEFPWD which indicates which users have default passwords. Prior to 11g, there was no straight forward way to check the users with default passwords. Querying this view, we can know what users have default passwords which could be a security threat.

```
SQL> SELECT username FROM dba_users_with_defpwd;
USERNAME
----------
DIP
MDSYS
WK_TEST
CTXSYS
OUTLN
EXFSYS
SCOTT
MDDATA
ORDPLUGINS
ORDSYS
XDB
```

Once we change the user's default password to a non-default password, it will no longer display in this view.

Recheck Grants

Further Security Discussions

Discovering Database Link Security Weakness

There is a requirement that if my schema has been granted SELECT privilege on a view in another schema, that I cannot further grant that privilege to a friend by, for example, creating a view of my own that includes a SELECT from the granted view. For example, if SYS grants me the privilege to SELECT from sys.dba_users, which is not normally granted to PUBLIC, and I create a view that uses a SELECT from sys.dba_users, then I can't grant SELECT on my view to others. Here is the code for this scenario:

SYS:
```
GRANT SELECT ON sys.dba_users TO me;
```

ME:
```
CREATE OR REPLACE VIEW me.my_dba_users AS SELECT * FROM sys.dba_users;
GRANT SELECT ON me.my_dba_users to friend;
```

As that set of commands stands, the final GRANT will fail. The me schema was not granted the privilege to allow other schemas (e.g., friend) to also SELECT from sys.dba_users. However with a slight modification, the final grant to friend can succeed. If the original SYS grant includes the WITH GRANT OPTION modifier, then the me user can allow other schemas to SELECT from sys.dba_users. The original GRANT would look like this:

SYS:
```
GRANT SELECT ON sys.dba_users TO me WITH GRANT OPTION;
```

Perhaps there is no way I can talk SYS or the database administrators into granted me select on sys.dba_users WITH GRANT OPTION. You may think my options are gone. But suppose I have been able to acquire CREATE DATABASE LINK system privilege as a user on another instance. Then I will be able to break out of the constraints and grant SELECT on sys.dba_users to another schema on that instance. The WITH GRANT OPTION requirement cannot be enforced across a database link. The code that follows demonstrates this loophole:

SYS@origin:
```
GRANT SELECT ON sys.dba_users TO me;
```

ME@origin;
```
CREATE OR REPLACE VIEW me.my_dba_users AS SELECT * FROM sys.dba_users;
```

SYS@alternate:
```
GRANT CREATE DATABASE LINK TO metoo;
GRANT CREATE VIEW TO metoo;
```

METOO@alternate;
```
CREATE DATABASE LINK origin_me_link CONNECT TO me IDENTIFIED BY password
USING 'origin';
CREATE OR REPLACE VIEW metoo.origin_dba_users AS SELECT *
FROM sys.proxy_users@origin_me_link;
GRANT SELECT ON metoo.origin_dba_users to friendalso;
```

I have just succeeded in granting SELECT on the sys.dba_users view on origin to the user friendalso on the alternate database, even though I was not granted WITH GRANT OPTION on the sys.dba_users view. That is the security weakness.

We are benefitting from this weakness in the two-instance configuration of our structures in this book. We create a UNION view named ojsaadm.instance_proxy_users on the orcl instance that selects from both the local sys.proxy_users view (which was granted WITH GRANT OPTION) and the view of sys.proxy_users across a database link to apver, which was not granted WITH GRANT OPTION. We are able to grant SELECT on this UNION view to the appsec user.

Back to our original discussion, to fully exploit the weakness and grant select on sys.dba_users to other schemas on the origin database instance, I would need the privilege to create a link from the origin. Here are additional commands that would enable this full-circle database link security weakness exploitation:

```
SYS@origin:
GRANT CREATE DATABASE LINK TO me;

ME@origin;
CREATE DATABASE LINK alternate_metoo_link CONNECT TO metoo IDENTIFIED BY password
    USING 'alternate';
CREATE OR REPLACE VIEW me.my_dba_users AS SELECT *
    FROM metoo.origin_dba_users@alternate_metoo_link;
GRANT SELECT ON me.my_dba_users to friend;
```

Now the friend user on the origin database instance has been, essentially, granted SELECT on the local sys.dba_users view by the me user, even though the me user was not granted SELECT on sys.dba_users WITH GRANT OPTION.