Development Security Guide
Oracle Banking Payments
Release 14.1.0.0.0
[May] [2018]

ORACLE
FINANCIAL SERVICES

ORACLE

Table of Contents

1. About this Manual

1.1 Introduction

Purpose:

This document provides security-related usage and configuration recommendations for Oracle Banking Payments 12.5.0.0.0. This guide may outline procedures required to implement or secure certain features, but it is also not a general-purpose configuration manual.

Audience:

This guide is primarily intended for Developers for Banking and third party or vendor software's. Some information may be relevant to IT decision makers and users of the application are also included. Readers are assumed to possess basic operating system, network, and system administration skills with awareness of vendor/third-party software's and knowledge of FCUBS application.

1.2 Scope

1.2.1 Read Sections Completely

Each section should be read and understood completely. Instructions should never be blindly applied. Relevant discussion may occur immediately after instructions for an action, so be sure to read whole sections before beginning implementation.

1.2.2 Understand the Purpose of this Guidance

The purpose of the guidance is to provide security-relevant code and configuration recommendations.

1.2.3 Limitations

This guide is limited in its scope to security-related guideline for developers.

2. How to address the OWASP Top10 in Oracle Banking Payments

2.1 Injection

Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or SQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code.

Banking uses Oracle database and it has adequate inbuilt techniques to prevent SQL injections as underlined below:-

1. **Use of prepared statements (parameterized queries)**—Banking uses PreparedStatement with bind variables to construct and execute SQL statements in JAVA.

2. **Use of Stored procedures**-- Stored procedures have the same effect as the use of prepared statements when implemented safely. 'Implemented safely' means the stored procedure does not include any unsafe dynamic SQL generation. Banking uses safe Java stored procedures calls.

   In addition to the above, wherever dynamic queries exist, Banking uses adequate defence to sanitize the untrusted input. The use of DBMS_ASSERT.SIMPLE_SQL_NAME and the use of bind variables justify the fact.

3. **Escaping all user supplied input--** This third technique is to escape user input before putting it in a query. If it's a concern that rewriting the dynamic queries as prepared statements or stored procedures might break the application or adversely affect performance, then this might be the best approach for the purpose. However, this methodology is frail compared to using parameterized queries and there's no guarantee that it will prevent all SQL Injection in all situations.
   Banking uses context specific escaping. It has a StringEscapeUtils.java file, where context specific escaping is handled.

2.2 Broken Authentication and Session Management

In Oracle Banking Payments application session interval will be validated against the session interval stored in the configurable file FCUBS.properties file. Validations are added to check the maximum time limit for the inactive session from being expired. Java API method javax.servlet.http.HTTPSession will set the max time out period for the session.

A maximum limit is imposed on the value passed to set the maximum limit of session interval. The maximum limit is a positive practical value. This validation is required to prevent long running sessions that can be actively targeted.

The default value for session time out is 30 minutes and it is configurable in FCUBS properties file.

The session used for login authentication will be invalidated (destroyed) and a new session will be created once the user logged-in successfully to the application .And the new session will be used to store the required variables.

A session attribute IsAuthenticated set to “Y” on successful login to the application. A new random token (Cross-site request forgery) also generates and same is available in the session attribute.

The entire subsequent request within the session will be having the Authenticated and Cross-site request forgery tokens .Every request send to the application from the browser is validated against the IsAuthenticated attribute and Cross-site request forgery token.

A hidden form is used to submit the logout request to the server, with the response resulting in a 302 redirect instead of client initiated redirect to the login page.

Session get expire once user log off from application or if idle for its maximum limit.

Cryptography used

PCI council defines **Strong Cryptography** as:

Cryptography based on industry-tested and accepted algorithms, along with strong key lengths and proper key-management practices. Cryptography is a method to protect data and includes both encryption (which is reversible) and hashing (which is not reversible, or “one way”). SHA-1 is an example of an industry-tested and accepted hashing algorithm. Examples of industry-tested and accepted standards and algorithms for encryption include AES (128 bits and higher), TDES (minimum double-length keys), RSA (1024 bits and higher), ECC (160 bits and higher), and ElGamal (1024 bits and higher).

Encryption algorithm: The application leverages AES encryption algorithm to store sensitive information into properties file. This algorithm uses 256 bit secret key for encryption and decryption which would be stored at property file.

Hashing algorithm: Oracle Banking Payments leverages SHA-512 hashing algorithm for user password authentication. This algorithm generates a password digest for the user password by using the SALT (Random number generated using SHA1PRNG algorithm) and the iteration number available in the property file.

Session storage

Oracle Banking Payments application does not store Http Session objects.

A unique sequence number generates and stored in current user table for the purpose of mapping server-side sessions with the entries in the current user table.

During session expiry (triggered by the container), the session listener provides the application with the sequence number of the session. The application makes checks as to whether the entry in current user table contains the same sequence number. Only in such a case should the entry be deleted.

When authentication of credentials (involving an incorrect user ID) is unsuccessful, the user id should not be logged in the audit logs (database table). The following possible scenarios will be accounted for:

Session logging

Unsuccessful attempt to login is stored in the database with terminal's ip address and timestamp. Invalid and expired session IDs submitted to the application are categorized as authentication failures and the same are logged in the database table.

2.3 Cross-Site Scripting (XSS)

XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. Banking is coded keeping in view the XSS prevention rules as below:-

1. **Technique#1—HTML Escape before inserting untrusted data into HTML element content**

   Across the Banking application, context specific escaping has been used to sanitize the untrusted data. For HTML content, the below function takes care of escaping the probable tainted data:
   public static String escapeHTML(String input);

   Escaping the following characters, with HTML entity encoding, to prevent switching into any execution context, such as script, style, or event handlers has been done. Use of recommended hex entities is in place. In addition to the 5 characters significant in XML (&, <, >, ", '), the forward slash is included as it helps to end an HTML entity.

   ```
   & --> &amp;
   < --> &lt;
   > --> &gt;
   " --> &quot;
   ' --> &#x27;
   / --> &#x2F;
   ```

2. **Technique #2-- JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values**
   Including untrusted data inside any other JavaScript context is quite dangerous, as it is extremely easy to switch into an execution context with characters including (but not limited to) semi-colon, equals, space, plus, and many more. For JavaScript context, the below function takes care of escaping the probable tainted data:
   public static String escapeJavaScript(String input);

3. **Technique #3—Escape JavaScript Characters**
   This works in conjunction with rule#2. Except for alphanumeric characters in Banking, all characters less than 256 are escaped with the \xHH format to prevent switching out of the data value into the script context or into another attribute. No use of any escaping shortcuts like \" ,because the quote character may be matched by the HTML attribute parser which runs first. These escaping shortcuts are also susceptible to "escape-the-escape" attacks where the attacker sends \" and the vulnerable code turns that into \\" which enables the quote.

4. **Technique #4--URL Escape And Strictly Validate Before Inserting Untrusted Data into HTML URL Parameters.**
   Banking encodes URL with the URLEncoder java class. It doesn't check for a valid URL, but directly does URL encoding, and that encoding is based on the context of display.

5. **Technique #5---Use of HttpOnly and secure cookie flag**
   Banking uses the HTTPOnly flag on the session cookie and any custom cookies that are not accessed by any JavaScript.

2.4 Insecure Direct Object References

1. **Use of prepared statements (parameterized queries)**
   Banking uses PreparedStatement with bind variables to construct and execute SQL statements in JAVA.

2. **Input Validation**
   Banking is a web based application, the request data from browser to server will be passed using request headers and request parameters. All the request fields coming from the client are validated using white list validation to prevent cross site scripting.
   User defined method validateParameter() is used for input validation which checks each character of the request field with a range of allowed characters.
   User defined methods escapeJavaScript(), escapeHTML() and escapeURL() will sanitize the output data before flushing it into client browser.
   escapeJavaScript() will escape all characters except immune JavaScript characters and alphanumeric characters in the ASCII character set. All other characters are encoded using the \\xHH or \\uHHHH notation for representing ASCII or Unicode sequences.
   escapeHTML() will escape the characters with equivalent HTML entities obtained from the lookup map. Lookup map will have entities such as amp, quot, lt, gt etc.
   escapeURL() will encode the URL using URLEncoder class.
   White list validation is also used to restrict Image/signature/excel upload and to check rights for every operation performed by user.

3. **Image Content validation**
   Signature upload will check for image type and image content using the inbuilt classes (ImageIO and JarFile) available in java.

4. **Field validation**
   Field level validations exist for all mandatory fields. Database too had limits on the type and the length of data. Blacklisted characters are not allowed in the mandatory fields. Nevertheless, Banking has free-text fields, which takes all data, entered by the user, as a String.

5. **Restriction on Blacklist characters**

   Similar to white list validation black list validation is also used for validating the request fields. Banking uses blacklist validation to check whether the request xml contains unwanted tags like scripting tag, html tag, anchor tag etc inside the xml content. It is also used for the advance summary field's validation to check whether proper request fields are coming from the browser.

Below table shows the list of bad characters which should not be allowed in URL path but the Banking operations requires many of the below characters to be passed in the request. So Banking will encode the below bad characters before sending them through the URL and same will be decoded at the server to prevent the hacker from modifying the request.

Bad URL Characters(Unsafe Characters)	
&	//
<	./
>	/.
;	/*
\"	*.
\'	~
%	\
)	25%
(	%25u
+	%25U
,	%00-%1f, %7f-%ff
" " (space)	%00-%1f and %7f-%ff
-	%25u and %25U

6. Restriction on Script/Html tags

Banking has blacklist validation for unwanted tag in xml like scripting tag or html tag inside xml content particularly in the header

2.5 Security Misconfiguration

1. Configuration files

Configuration files are securely placed inside the Classes folder of the WEB-INF folder which is not publicly accessible.

2. **Exception handling in java**

Different types of exceptions can rise in application. Java exceptions handled using try catch blocks available in java. Sometimes we use the Throw statement to throw an exception which is caught by the catch block. Caught exceptions will be written into the log files for the debug purpose when ever required. Whenever any exception occurs in application, proper information used to send to the front end user by showing alert.

3. **Exception handling in oracle database**

Database exceptions handled using EXCEPTION statement available in PL/SQL. Caught exceptions will be written into the log files for the debug purpose. And proper error message created to send the same in response to the user.

4. **Package lockout situation handled in backend**

Application will be hanged in an oracle system package lockout situation. Locked objects will be released manually using SQL scripts or through database restart.

We have handled cursor lock out problem in the required packages.

5. **Auto generated password:**

The password is generated by the system accordance to the password policy. The salt is also be generated every time the password is changed by using predefined algorithm.

The salt concatenated with auto generated password and SHA-512 hash applies on the resultant which results the password digest.

Once the successful generation of password digests both salt and password digest is stored in the DB.

6. **Custom password:**

The password is keyed in by the administrator / user accordance to the password policy. The salt is generated every time the password is changed by using predefined algorithm.

The salt concatenated with the password input and SHA-512 hash applies on the resultant which results the password digest.

Once the successful generation of password digests both salt and password digest is stored in the DB.

Oracle Banking Payments does not provide any default user/password. User and password needs to be created at the time of installation.

7. **Sand Box for File Upload**

The application uses a sandbox for placing files that are uploaded via the signature/image upload screen. The sandbox is placed in a specified location (the location will be specified in the properties file) on the server.

8. BI Publisher Reports – generation and access

The application uses a sandbox for placing the generated reports file into a sandbox area. The sandbox is placed in a specified location (the location will be specified in the properties file) on the server. The application validates if the user has explicit Rights to generate Reports.

2.6 Sensitive Data Exposure

1. Secure Transformation of Data (SSL)

The Oracle Banking Payments Installer allows a deployer to configure Oracle Banking Payments such that all HTTP connections to the Oracle Banking Payments application are over SSL/TLS. In other words, all HTTP traffic in the clear will be prohibited; only HTTPS traffic will be allowed. It is mandatory to enable this option in a production environment, especially when WebLogic Server acts as the SSL terminator.

A two-way SSL is used when the server needs to authenticate the client. In a two-way SSL connection the client verifies the identity of the server and then passes its identity certificate to the server. The server then validates the identity certificate of the client before completing the SSL handshake.

In order to establish a two-way SSL connection, need to have two certificates, one for the server and the other for client.This is required for de-centralized setup of application.

For Oracle Banking Payments, need to configure a single connector. This connector is related to SSL/TLS communication between host or browser and the branch which uses two-way authentication.

If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection, thereby preventing the cookie from being trivially intercepted by an attacker monitoring network traffic.

Below configuration has to be ensured in weblogic.xml within the deployed application ear.

- Cookies are set with Http only as true
- Cookie secure flag set to true
- Cookie path to refer to deployed application

```
<wls: session-descriptor>
 <wls: cookie-http-only>true</wls: cookie-http-only>
</wls: session-descriptor>

<wls: session-descriptor>
<wls: cookie-secure>true</wls: cookie-secure>
 <wls: url-rewriting-enabled>false</wls: url-rewriting-enabled>
</wls: session-descriptor>

 <session-descriptor>
  <cookie-name>JSESSIONID</cookie-name>
  <cookie-path>/<DeployedApplicationPath></cookie-path>
  <cookie-http-only>true</cookie-http-only>
  <cookie-secure>true</cookie-secure>
  <url-rewriting-enabled>false</url-rewriting-enabled>
        </session-descriptor>
```

Always make sure Cookies are set with always Auth Flag enabled by default for WebLogic server .

2. Sign-On messages

Below table shows the general Sign-On messages which would be displayed to the user during invalid authentication.

Message	Explanation
User Already Logged In	The user has already logged into the system and is attempting a login through a different terminal.
Invalid User ID/Login.	An incorrect user ID or password was entered.
User Status is Disabled. Please contact your System Administrator	The user profile has been disabled due to number of dormancy days allowed for the user has exceeded the dormancy days configured in the system.
User Status is Locked. Please contact your System Administrator	The user profile has been locked due to an excessive number of attempts to login, using an incorrect user ID or password. The number of attempts could have matched either the successive or cumulative number of login failures (configured for the system).

3. CACHE Control in Servlet and jsp

There are three basic HTTP response headers that prevent a page from being cached to disk. Different browsers handle them in slightly different ways, so they need to be used in combination to ensure all browsers do not cache the specific page. These headers are "Expires", "Pragma" and "Cache-control". In addition, these headers can either be sent directly by the server or placed in the HTML code as HTTP-EQUIV META tags within the HEAD section. The "Expire" header gives a date at which point the page should expire and no longer be cached. Internet Explorer supports a date of "0" for immediately and any negative number for already expired. The "Pragma: no-cache" header indicates that the page should not be cached.

4. Clickjacking/Frame-bursting

Banking uses the X-Frame-Options HTTP response header to indicate whether or not a browser should be allowed to render a page in a <frame> or <iframe>. This is used to avoid Clickjacking attacks, by ensuring that the content is not embedded into other sites.

2.7 Missing Function Level Access Control

It is likely that users working in the same department at the same level of hierarchy need to have similar user profiles. In such cases, you can define a Role Profile that includes access rights to the functions that are common to a group of users. A user can be linked to a Role Profile by which you give the user access rights to all the functions in the Role Profile.

Application level access has implemented via the Security Management System (SMS) module.
SMS supports “ROLE BASED” access of Screens and different types of operations.
Banking Payments supports dual control methodology, wherein every operation performed has to be authorized by another user with the requisite rights.
Please refer 2.6 section of the SMS user manual for more details.

Apart from the role based access control particular functions , products can be restricted for user as described below.

***Disallowed functions**:* Function IDs or UI level restrictions can be provided for the user by including the function Ids in the disallowed list. This will restrict the user from accessing the UI. When accessed, an error message dialogue box will pop up saying-“User not authorized to access the screen”.

Disallowed account class: The user could be restricted to perform any operation using a particular a/c class. When disallowed, no accounts could be created by the user using the account class.

Disallowed products: The user could be restricted to use product(s) of any module(s), if disallowed. This is really required when restricting users department wise. For example, staffs of accounts department need not be given access to view the loans of customers.

Disallowed branches: The user could be restricted to access branches other than his own branch (reporting branch). He can be given access to login from other branches of the bank at an approval from authenticated person, an action which again requires manual authorization.

2.8 Cross-Site Request Forgery (CSRF)

In case of XMLHttpRequest objects, the XMLHttpRequest object sets a custom HTTP header in the request, with the header value being the Cross-site request forgery token; the server then verifies for the presence of such a header and the Cross-site request forgery token. This serves as a protection at endpoints used for XMLHttpRequest requests, since only XMLHttpRequest objects can set HTTP headers (apart from Flash; and both cannot make cross-domain requests).

2.9 Using Components with Known Vulnerabilities

Source code scanning done using the latest fortify to identify the sources code issue and will provide the proper fix for the reported issues.

3rd party libraries scanning for every release has been done to validate if any security issues rise for any of the components or not. Update the 3PL with latest security patch or upgraded to latest version.

2.10 Unvalidated Redirects and Forwards Network Security

Application uses 302 redirect wherever required. Banking UBS uses response.sendRedirect(newURL);

3. Securing Gateway Services

Different applications deployed on disparate platforms and using different infrastructure need to be able to communicate and integrate seamlessly with Oracle Banking Payments in order to exchange data. The Oracle Banking Payments Integration Gateway will cater to these integration needs.

The integration needs supported by the Gateway can be broadly categorized from the perspective of the Gateway as follows:

- Inbound application integration – used when any external system needs to add, modify or query information within Oracle Banking Payments
- Outbound application integration – used when any external system needs to be notified of the various events that occur within Oracle Banking.

3.1 Inbound Application Integration

Oracle Banking Payments Inbound Application Gateway provides XML based interfaces thus enhancing the need to communicate and integrate with the external systems. The data exchanged between Oracle Banking Payments and the external systems will be in the form of XML messages. These XML messages are defined in FCUBS in the form of XML Schema Documents (XSD) and are referred to as 'FCUBS formats'.

FCUBS Inbound Application Integration Gateway uses the Synchronous and Asynchronous Deployment Pattern for addressing the integration needs.

The Synchronous Deployment Pattern is classified into the following:

- Oracle Banking Payments EJB Based Synchronous Inbound Application Integration Deployment Pattern
- Oracle Banking Payments Web Services Based Synchronous Inbound Application Integration Deployment Pattern
- Oracle Banking Payments MDB Based Asynchronous Inbound Application Integration Deployment Patten

3.2 EJB Based Synchronous Deployment Pattern

The Enterprise Java Beans (EJB) deployment pattern will be used in integration scenarios where the external system connecting to Oracle Banking Payments is 'EJB literate', i.e., the external system is capable of interacting with Oracle Banking Payments based upon the EJB interface. In this deployment pattern, the external system will use the RMI/IIOP protocol to communicate with the Oracle Banking Payments EJB.

In this deployment pattern the EJB displayed by Oracle Banking Payments will be a stateless session bean. The actual request will be in the form of an XML message. After the necessary processing is done in Oracle Banking Payments based on the request, the response is returned to the external system as an XML message. The transaction control for the processing will stay with the Oracle Banking Payments EJB.

3.3 Web Services Based Synchronous Deployment Pattern

The web services deployment pattern will be used in integration scenarios where the external system connecting to Oracle Banking Payments wants to connect using standards-based, inter-operable web services.

This deployment pattern is especially applicable to systems which meet the following broad guidelines:

- Systems that are not 'EJB literate', i.e., such systems are not capable of establishing connections with Oracle Banking Payments based upon the EJB interface; and/or
- Systems that prefer to use a standards-based approach

In this deployment pattern, the external system will use the SOAP (Simple Object Access Protocol) messages to communicate to the Oracle Banking Payments web services.

The services displayed by Oracle Banking Payments are of a 'message based' style, i.e., the actual request will be in the form of an XML message, but the request will be a 'payload' within the SOAP message. After the necessary processing is done in Oracle Banking Payments based on the request, the response is returned to the external system as an XML message which will be a 'payload' within the response SOAP message. The transaction control for the processing will stay with the Oracle Banking Payments.

3.4 HTTP Servlet Based Synchronous Deployment Pattern

The HTTP servlet deployment pattern will be used in integration scenarios where the external system connecting to Oracle Banking Payments wants to connect to Oracle Banking Payments using simple HTTP messages.
This is especially applicable to systems such as the following:

- Systems that are not 'EJB literate', i.e., are not capable establishing a connections with Oracle Banking Payments based upon the EJB interface; and/or
- Systems that prefer to use a simple http message based approach without wanting to use SOAP as the standard
-

In this deployment pattern, the external system will make an HTTP request to the Oracle Banking Payments servlet.

For this deployment pattern, Oracle Banking Payments will display a single servlet. The actual request will be in the form of an XML message. This XML message is embedded into the body of the HTTP request sent to the Oracle Banking Payments servlet. After the necessary processing is done in Oracle Banking based on the request, the response is returned to the external system as an XML message which is once again embedded within the body of the response HTTP message. The transaction control for the processing will stay with the Oracle Banking.

3.5 MDB Based Asynchronous Deployment Pattern

The MDB deployment pattern is used in integration scenarios where the external system connecting to Oracle Banking Payments wants to connect to Oracle Banking Payments using JMS queues.

This is especially applicable to systems such as the following:

- Systems that prefer to use JMS queues based approach without wanting to wait for the reply
-

Here external system sends messages in XML format to request queue on which an MDB is listening. When a message arrives on the queue, it is picked up for processing. After the necessary processing is done in Oracle Banking Payments, based on the request, the response is sent to the response queue as an XML message.

3.6 Outbound Application Integration

The Outbound Application Integration is also called the Oracle Banking Payments Notify Application Integration layer. This application layer sends out notification messages to the external system whenever events occur in Oracle Banking Payments.

The notification messages generated by FCUBS on the occurrence of these events will be XML messages. These XML messages are defined in FCUBS in the form of XML Schema Documents (XSD) and are referred to as 'FCUBS formats'

3.7 Securing Web Services

Web services can be secured by applying security policies available in web logic sever. We can attach two types of policies to Web Logic Web services and clients at design and deployment time.
Oracle WSM policy : We can attach Oracle Web Services Manager(WSM) policies to Web Logic JAX-WS Web services and clients

Web Logic Web service policy: This policies are provided by Oracle Web Logic Server and can be attached to any web service deployed in Web Logic.
We can use Oracle Enterprise Manager Fusion Middleware Control to attach Oracle WSM security policies to Web Logic Java EE Web services and clients.
We can attach policies to Web Logic Web services at both design time and after the Web service has been deployed.
At design time, use the weblogic.jws.Policy and weblogic.jws.Policies JWS annotations in JWS file to associate policy files with Web service. We can associate any number of policy files with a Web service, although it is up to us to ensure that the assertions do not contradict each other. We can specify a policy file at the class level of our JWS file.

After the Web service has been deployed, use the Oracle Web Logic Server Administration Console to attach Web Logic Web service policies to Web Logic Web services.

3.8 Accessing Service and Operation

In a message it is mandatory to maintain a list of Service Names and Operation Codes. This information is called Gateway Operations.
A combination of every such Service Name and Operation Code is mapped to a combination of Function ID and Action. Every screen in Oracle Banking Payments is linked with a function ID. This information is called Gateway Functions.
User can gain access to an external system using the Gateway Functions. The Function IDs mapped in Gateway Functions should be valid Function IDs maintained in Oracle Banking Payments. Hence, for every new Service or Operation being introduced, it is important that you provide data in Gateway Operations and Gateway Functions.

3.9 Gateway Password Generation Logic for External System Authentication

As a secure configuration password authentication should be enabled for the external system maintained. The same can be verifying in External system detail screen level.
Once these features enable, system will validate for Encrypted password as part of every request sent by the External System.
The Message ID which is present as part of the header in Request XML, is considered as hash. External System generates an unique Message ID, which is functional mandatory field in the header. Create a Message Digest with SHA-512 algorithm.
The hash created from the previous step and the password in clear text together is encrypted in AES encryption method. Apply Base64 encoding to encrypted value and send to the Oracle Banking Payments gateway.

3.10 XSD Validation and Input Validation

Oracle Banking Payments supports the XSD validation for all types Gateway. Each node in request xml is getting validated with the corresponding webservice XSD's.

Restriction on Script/Html tags

Banking Gateway has blacklist validation for unwanted tag in xml like scripting tag or html tag inside xml content particularly in the header

3.11 List of Services

COCategoryTypeService
COCurrencyPairService
COCurrencyRateTypeService
COCurrencyService
FCUBSPXService

3.12 List of Interfaces

Generic Interface
Document Management System Interface
SWIFTNet Services Integrator Messaging Hub Interface
Oracle Identity Manager Interface

Development Security Guide
[May] [2018]
Version 14.1.0.0.0
Oracle Financial Services Software Limited
Oracle Park
Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India

Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax:+91 22 6718 3001
www.oracle.com/financialservices/