Development Security Guide
Oracle Banking Liquidity Management
Release 14.1.0.0.0
[May] [2018]

Table of Contents

1. About this Manual

1.1 Introduction

Purpose:

This document provides security-related usage and configuration recommendations for Oracle Banking Liquidity Management 14.1.0.0.0. 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 OBLM 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 OBLM 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 Liquidity Management

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.

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

1. **Use of prepared statements (parameterized queries)** - Oracle Banking Liquidity Management 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. Oracle Banking Liquidity Management uses safe Java stored procedures calls.

   In addition to the above, wherever dynamic queries exist, Oracle Banking Liquidity Management uses adequate defence to sanitize the untrusted input.

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.

2.2 Broken Authentication and Session Management

In Oracle Banking Liquidity Management application session interval will be validated against the session interval stored in the system setup maintenance screen. 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.

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 Liquidity Management Solution 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 Liquidity Management Solution 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. Oracle Banking Liquidity Management is coded keeping in view the XSS prevention rules as below:-

2.4 Insecure Direct Object References

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

2. **Input Validation**
   Oracle Liquidity Management Solution 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.

3. **Field validation**
   Field level validations exist for all mandatory fields. Database too had limits on the type and the length of data.

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 whenever 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 Liquidity Management does not provide any default user/password. User and password needs to be created at the time of installation.

2.6 Sensitive Data Exposure

1. Secure Transformation of Data (SSL)

The Oracle Banking Liquidity Management allows HTTP connections 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 Liquidity Management, 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

Oracle Banking Liquidity Management 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.
Oracle Banking Liquidity Management 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.

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. OBLM uses response.sendRedirect(newURL);

2.11 Securing REST Services

You can secure your RESTful Web services using one of the following methods to support authentication, authorization, or encryption:

- Updating the web.xml deployment descriptor to define security configuration.
- Using the javax.ws.rs.core.SecurityContext interface to implement security programmatically.
- Using Jersey OAuth libraries to sign and verify requests.
- 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.

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/