Oracle® Banking Platform

Secure Development Guide

Release 2.6.2.0.0

E95189-01

May 2018

ORACLE®

Oracle Banking Platform Secure Development Guide, Release 2.6.2.0.0

E95189-01

Contents

List of Figures

List of Tables

Preface

The Secure Development Guide provides recommendations for secure usage of extensible components.

This preface contains the following topics:

Audience

This guide is intended for Oracle partners and any users who want to extend the product's components.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/us/corporate/accessibility/index.html.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/us/corporate/accessibility/support/index.html#info or visit http://www.oracle.com/us/corporate/accessibility/support/index.html#trs if you are hearing impaired.

Organization of the Guide

This document contains:

Chapter 1 About This Guide

This chapter provides details about the applicability of this guide.

Chapter 2 Overview

This chapter presents an overview of the secure usage of extensible components.

Chapter 3 Common Terms

This chapter provides a list of common terms used in this guide along with their descriptions.

Chapter 4 Security in API Development

This chapter explains the security features offered by the product in API development.

Related Documents

For more information, see the following documentation:

- For installation and configuration information, see the Oracle Banking Platform Installation Guide - Silent Installation.
- For the complete list of licensed products and the third-party licenses included with the license, see the Oracle Banking Licensing Guide.
- For information related to setting up a bank or a branch, and other operational and administrative functions, see the Oracle Banking Administrator's Guide.
- For information related to customization and extension, see the Oracle Banking Extensibility Guide.
- For a comprehensive overview of security, see the Oracle Banking Security Guide.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 About This Guide

This guide is applicable for the following products:

- Oracle Banking Platform
- Oracle Banking Enterprise Product Manufacturing
- Oracle Banking Enterprise Originations
- Oracle Banking Enterprise Collections

References to Oracle Banking Platform or OBP in this guide apply to all the above mentioned products.

2 Overview

Oracle Banking Platform (OBP) leverages on Oracle Platform Security Services to protect for authentication, authorization, auditing, and role and credential management.

Many features in Oracle Application Development Framework (ADF), such as bounded task flows, that are not primarily designed as a security feature, are used by OBP to protect against known security threats.

This guide provides recommendations for secure usage of OBP's extensible components.

3 Common Terms

The following table presents the common terms used in this guide along with their descriptions.

Table 3–1 Common terms

Term	Description
Transaction	Any activity done using SPI with intention of change in Oracle Banking Platform is treated as single transaction. The effect of a transaction leads to state change of the entity (like maintenance) or financial change. Being SOA driven, Oracle Banking Platform is stateless. Each transaction is unaware of previous transaction and is atomic in that respect. The client will have to ensure the continuity of information or state if there is a requirement to hold state between multiple user interactions.
Approval	It signifies the process of making an entity change effective in the system. Authorization is performed by the supervisor. It can be done for both maintenance as well as financial transactions. If the response of the API indicates that authorization is required due to the role assigned to the user, then the client must use Approval Workflow processes to authorize the transaction.
One Factor Authentication (1FA)	This is a special case when customers have to again provide their consent, even though the transaction is being done by customers themselves within the capacity of their assigned role. The client will have to capture the consent and call a specific service for completion of the transaction.
Two Factor Authentication (2FA)	This is a special case of authorization where the customer related to the entity has to approve the transaction, even though it is being done through role of authorized system user. The client will have to invoke additional APIs to procure the customer's consent to the transaction. This can be done by sending Short Messaging Service (SMS) to the registered cellular device in the system asking for confirmation. This approach is also known as One Time Password (OTP) feature. In this case, the end-user provides the OTP to the system to complete the transaction. The client should have mechanisms to send and accept the OTP. Once authenticity is confirmed, then the client must call specific API in the Oracle Banking Platform to complete the transaction.

4 Security in API Development

The basic steps to call an API are as follows:

1. Verify credentials
2. Invoke the desired API

These steps are explained in detail in the following sections.

4.1 Credential Verification

Oracle Banking Platform authenticates users by verifying credentials against an LDAP credential store. Clients must use SAML 2.0 for verification.

SAML 2.0 is an XML-based protocol that uses security tokens containing assertions to pass information about a principal (usually an end user) between a SAML authority, that is an identity provider, and a web service, that is a service provider. SAML 2.0 enables web-based authentication and authorization scenarios including single sign-on (SSO).

The client will have to pass the SAML compliant token by implementing SAML interface. This will add the token to the SOAP header.

Non-weblogic clients must use the Username Token Policy. Username Token is a signed supporting token that can be used to send the username or password to the other end. The recipient can check whether the request has come from a valid user. When using a Username Token based security policy, transport level security should be implemented. Using HTTPS, the client can communicate through a pre-established secure tunnel. So the confidentiality and the integrity of the messages are protected. Please note that the username must be registered in the LDAP server for the correct level of access or role.

Weblogic clients can implement SAML Token Policy. In this case, the weblogic server can be configured to communicate with identity authority server, and the intercommunication between the client and application can just share a token, instead transmitting username and password.

4.1.1 Sharing the token

In either case, the SOAP request has to be modified through SOAP Handler for adding the token to the header. This can be done by creating a custom SOAP request handler and adding it to the handler chain.

```
CustomSOAPHandler customHandler = new CustomSOAPHandler();
List<Handler> handlerChain = new ArrayList<Handler>();
handlerChain.add(customHandler);
((BindingProvider)clientProcess).getBinding().setHandlerChain
(handlerChain);
```

Refer javax.xml.ws.Binding.setHandlerChain(List<Handler> chain)

4.1.2 Preparing the token

In the CustomSOAPHandler, the method handleMessage(SOAPMessageContext context) is implemented as shown below for Username Token policy.

```
String AUTH_PREFIX = "wsse";
```

```
String AUTH_NS = "http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd";

SOAPEnvelope envelope = context.getMessage().getSOAPPart
().getEnvelope();
SOAPFactory soapFactory = SOAPFactory.newInstance();
SOAPElement wsSecHeaderElm = soapFactory.createElement("Security",
AUTH_PREFIX, AUTH_NS);
Name wsSecHdrMustUnderstandAttr = soapFactory.createName
("mustUnderstand",
"S",
AUTH_NS);
wsSecHeaderElm.addAttribute(wsSecHdrMustUnderstandAttr, "1");
SOAPElement userNameTokenElm = soapFactory.createElement
("UsernameToken", AUTH_PREFIX, AUTH_NS);
Name userNameTokenIdName = soapFactory.createName("id", "wsu",
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd");
userNameTokenElm.addAttribute(userNameTokenIdName, "UsernameToken-
ORbTEPzNsEMDfzrI9sscVA22");
SOAPElement userNameElm = soapFactory.createElement("Username",
AUTH_PREFIX, AUTH_NS);

String username = getSessionUserName();
userNameElm.addTextNode(username);

SOAPElement passwdElm = soapFactory.createElement("Password", AUTH_
PREFIX, AUTH_NS);
Name passwdTypeAttr = soapFactory.createName("Type");
passwdElm.addAttribute(passwdTypeAttr, "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText");

String password = getSessionUserPassword();
passwdElm.addTextNode(password);

userNameTokenElm.addChildElement(userNameElm);
userNameTokenElm.addChildElement(passwdElm);
wsSecHeaderElm.addChildElement(userNameTokenElm);
if (envelope.getHeader() == null) {
SOAPHeader sh = envelope.addHeader();
sh.addChildElement(wsSecHeaderElm);
} else {
SOAPHeader sh = envelope.getHeader();
sh.addChildElement(wsSecHeaderElm);
}
```

The SOAP Request looks like this:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dda="http://core.account.service.dda.app.fc.ofss.com/DDAInqu
iryApplicationService" xmlns:con="http://context.app.fc.ofss.com"
xmlns:exc="http://exception.infra.fc.ofss.com"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:soap="soap">
<soapenv:Header>
<wsse:Security soap:mustUnderstand="1">
<wsse:UsernameToken wsu:Id="UsernameToken-1">
<wsse:Username>devuser01</wsse:Username>
<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-
1.0#PasswordText">welcome1</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</soapenv:Header>
<soapenv:Body>
<dda:fetchAccountDetailsDDAInquiry>
<!--Optional:-->
<dda:sessionContext>
<con:bankCode>08</con:bankCode>
<con:targetUnit>UBank_1</con:targetUnit>
<con:transactionBranch>082991</con:transactionBranch>
<con:userId>devuser01</con:userId>
</dda:sessionContext>
<dda:accountId>0000000000008590</dda:accountId>
</dda:fetchAccountDetailsDDAInquiry>
</soapenv:Body>
</soapenv:Envelope>
```

4.2 Invocation of API

A typical API signature will follow the convention as shown below:

```
public SomeAPIResponse com.ofss.fc.app.service.method
(SessionContext sessionContext, Object... input) throws
FatalException
```

OR

```
public TransactionStatus com.ofss.fc.app.service.method
(SessionContext sessionContext, Object... input) throws
FatalException
```

The clients will call the APIs multiple times during one session. Since the nature of the transactions will be financial, it will be necessary to ensure security, integrity and reliability in the various inter system

interactions. In such cases, the following concepts are important and must be followed for proper interaction with OBP.

4.2.1 Establishing the context

com.ofss.fc.app.context.SessionContext:

This entity captures the basic information about the interaction being initiated with the system. This information is used by the system for various purposes:

- For establishing identity
- For establishing target unit
- For establishing the business dates to be used
- For capturing referential information with external system or caller
- For capturing the authorization related business information
- For establishing the behavior of the interaction

Once the processing of the request starts, the system fills in more contextual information which has not been supplied by the caller. This context is then shared between all the APIs which are invoked during the interaction. Specific relevance of the information has been discussed below:

- **Identity**

  The caller of the system must present the user ID which will be used for the interaction.This user ID will also be logged for internal audit purposes, and must be a valid. The system will validate the user ID with internal identity management systems. It is also important to note the medium or channel through which this interaction is happening. This allows the system to behave differently in terms of handling error conditions depending on the configuration. These attributes are of relevance here:

  - userId: Mandatory
  - channel: Optional, Defaults to BRN
  - enterpriseRole: Derived, For internal usage only
  - userLocale: Optional, Defaults to en_US

- **Authorization Information**

  Sometimes, an interaction requires supervisor authorization. In such cases, the transaction has to re-presented with authorization related information such as, supervisor's identity, the reasons due to which authorization is being given, and the reasons for which authorization has been requested. These attributes are of relevance here:

  - approvalContext: Conditional Mandatory (only for authorization)
  - authorizationReason: Conditional Mandatory (only for authorization)
  - overridenWarnings: Optional

- **Behavior**

  It is often required to interact with the system under controlled conditions. For example, for simulation, it is required to rollback the effect after completing the entire interaction. In some other cases, the system may behave differently for different API calls. These attributes are of relevance here:

- serviceCallContextType: Optional, defaults to ServiceCallContextType.NORMAL
- serviceCode: Optional
- channel: Optional

Sample code to create the SessionContext is given below:

```
/**
* Method to configure the SessionContext
*
* @return SessionContext
*/
protected SessionContext getSessionContext() {

SessionContext sessionContext = new SessionContext();

sessionContext.setBankCode(getStringValue
("sessionContext.bankCode")); sessionContext.setTransactionBranch
(getStringValue("sessionContext.transactionBranch"));
sessionContext.setUserId(getStringValue("sessionContext.userId"));
sessionContext.setChannel(getStringValue
("sessionContext.channel"));
sessionContext.setTargetUnit(getStringValue
("sessionContext.targetUnit")); sessionContext.setPostingDateText
(getStringValue("sessionContext.postingDate"));
return sessionContext;

}
```

com.ofss.fc.app.context.ApprovalContext:

This class represents the approval related information. When a transaction is approved by a supervisor, it is done against a set of reply codes that was presented by the system during initial execution. These are captured as approvedOverrides. When an approval is given, it is accompanied with reasons. These are captured in approvalReasons.

com.ofss.fc.enumeration.ServiceCallContextType:

This enumeration defines the mode or behavioral constraint under which the system has to action upon the service request. To rollback forcefully, the client must send VALIDATE.

Sample JUnit code for invocation of an API is given below:

```
public final void testDebitWithLedger() {

try {
SessionContext sessionContext = getSessionContext();
TransactionStatus transactionStatus =
applicationService.debitWthLedger(sessionContext,
getStringValue("accountId"),
new MoneyDTO(getBigDecimalValue("Amount"),
getStringValue("Currency")),
getStringValue("ledgerCode"),
null);
```

```
if (transactionStatus.getReplyCode() == 30) {
System.out.println("Transaction requires authorization,will be
logged in worklist for approval");
return;
} else if (transactionStatus.getReplyCode() != Long.parseLong
(ResponseCodeType.SUCCESS.getValue())) {
fail();
}
if (transactionStatus.getReplyCode() == 40) {
try {
SelfApprovalApplicationService applicationService = new
SelfApprovalApplicationService();
WorkItemApprovalRequestDTO requestDTO = new
WorkItemApprovalRequestDTO();
requestDTO.setWorkFlowId
(transactionStatus.getInternalReferenceNumber());
requestDTO.setComment(getUserApprovalComments());
WorkItemApprovalResponse response =
applicationService.inquireAndApproveWorkItem(sessionContext
,requestDTO);

if (response.getStatus().getReplyCode() == 30) {
System.out.println("Transaction requires authorization,will be
logged in worklist for approval");
return;
} else if (response.getStatus().getReplyCode() != Long.parseLong
(ResponseCodeType.SUCCESS.getValue())) {
fail();
}

} catch (FatalException fatalException) {
dumpFatalException("DDATransactionApplicationServiceJUnit",
"testDebitWithLedger", fatalException);
fail();
}
}
return;
} catch (FatalException fatalException) {
dumpFatalException("DDATransactionApplicationServiceJUnit",
"testDebitWithLedger", fatalException);
fail();
}
}
```

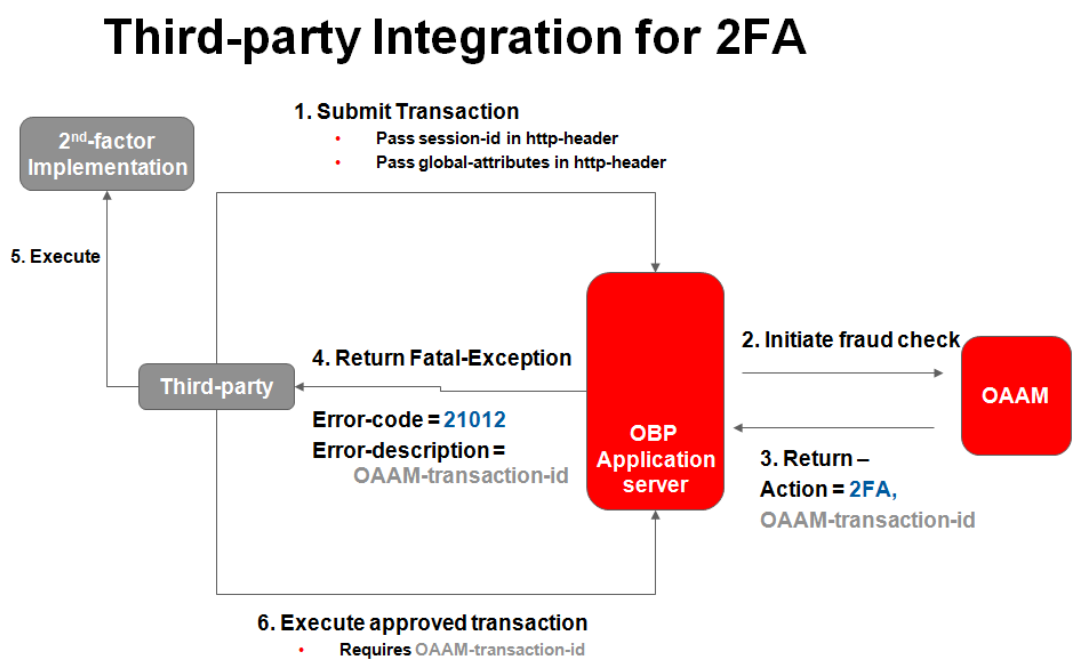
4.2.2 Establishing the user session continuity and detecting fraud

We need to establish the continuity of user interaction across multiple interactions between the client and system. This is done for various instances where fraud assertion is required before the transaction is

submitted for final completion.

The figure below depicts the second-factor authentication sequence.

Figure 4–1 Third-party Integration for 2FA



In the above figure, steps 2 and 3 are transparent to the third-party

Second-factor implementation can be of various types:

- Validation of a one-time password (OTP)
- Validation against an RSA token

The variables to be passed in the http header are detailed below:

Table 4–1 Variables

Header Variables	Description
IP_ADDRESS	IP address of the machine where the client browser is running
FRAUD_ASSERTION_ SESSION_ID	OAAM session ID for the logged-in session
CLIENT_HEADER	Client header details from http request
Remote_Host	Remote host IP from http request
User-Agent	USER AGENT details from http request

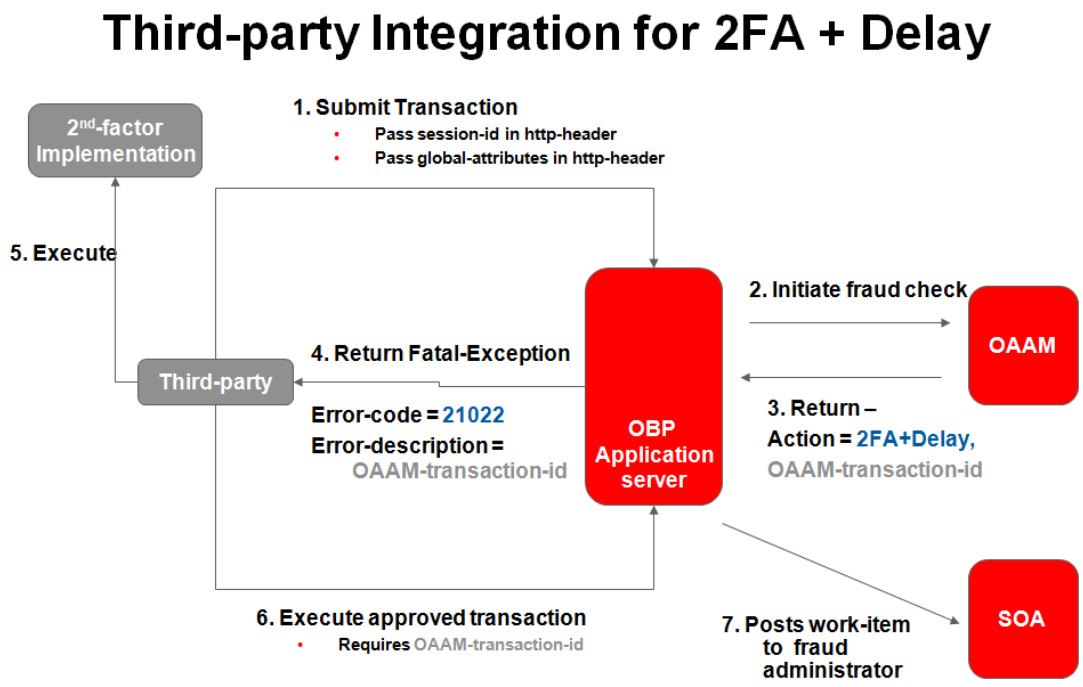
The Fatal-Exception error codes and their meanings are detailed below:

Table 4–2 Error Codes

Error Code	Expected behavior by client, if applicable
21022	Challenge 2FA to customer
11021	Challenge 1FA to customer

The figure below depicts the sequence for triggering a payment delay, in addition to triggering a second-factor authentication sequence.

Figure 4–2 Third-party Integration for 2FA + Delay



This additional step is configured only for a few sensitive payment transactions. Such transactions are delayed for a few hours and are sent to the fraud-administrator's queue for a final go-ahead. If the work-item is rejected at this stage, a reversal transaction is added and the payment is not sent out.