Oracle Banking Digital Experience

Security Guide
Release 16.2.0.0.0

Part No. E79009-01

October 2016

ORACLE®

Security Guide

October 2016

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/

Table of Contents

Preface

This document provides a comprehensive overview of security for Oracle Banking Digital Experience. It includes conceptual information about security principles, descriptions of the product's security features, and procedural information that explains how to use those features to secure Oracle Banking Digital Experience.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Access to Oracle Support
- Related Documents

Audience

This Security Guide is intended for Bank IT Staff responsible for application installation and security configuration as well as Product Implementation teams.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if your hearing is impaired.

Related Documents

For more information, see the following documentation:

- Hardening Tips for Default Installation of Oracle Enterprise Linux 6 at https://docs.oracle.com/cd/E37670_01/E36387/E36387.pdf
- Oracle® Fusion Middleware Installation Guide for Oracle WebLogic Server at http://docs.oracle.com/cd/E17904_01/doc.1111/e14142/toc.htm
- Oracle® Fusion Middleware Application Security Guide - Configuring and Managing Auditing at http://docs.oracle.com/cd/E23943_01/core.1111/e10043/audpolicy.htm
- For installation and configuration information, see the Oracle Banking Digital Experience Installation Guide
- For the complete list of Oracle Banking licensed products and the Third Party licenses included with the license, see the Oracle Banking Licensing Guide.

1

General Security Principles

The following principles are fundamental for using any application securely.

1.1 Restrict Network Access to Critical Services

Keep both the Oracle Banking Digital Experience middle-tier and the database behind a firewall. In addition, place a firewall between the middle-tier and the database. The firewalls provide assurance that access to these systems is restricted to a known network route, which can be monitored and restricted, if necessary. As an alternative, a firewall router substitutes for multiple, independent firewalls.

If firewalls cannot be used, be certain to configure the TNS Listener Valid Node Checking feature which restricts access based upon IP address. Restricting database access by IP address often causes application client or server programs to fail for DHCP clients. To resolve this, consider using static IP addresses, a software or a hardware VPN or Windows Terminal Services or its equivalent.

1.2 Follow the Principle of Least Privilege

The principle of least privilege states that users should be given the least amount of privilege to perform their jobs. User privileges should be reviewed periodically to determine relevance to current job responsibilities.

1.3 Monitor System Activity

System security largely depends on the following practices:

- Good security protocols
- Proper system configuration
- System monitoring

The system needs to be constantly monitored from a monitoring tool.

1.4 Keep Up To Date on Latest Security Information

Oracle continually improves its software and documentation. It is recommended to keep your software updated.

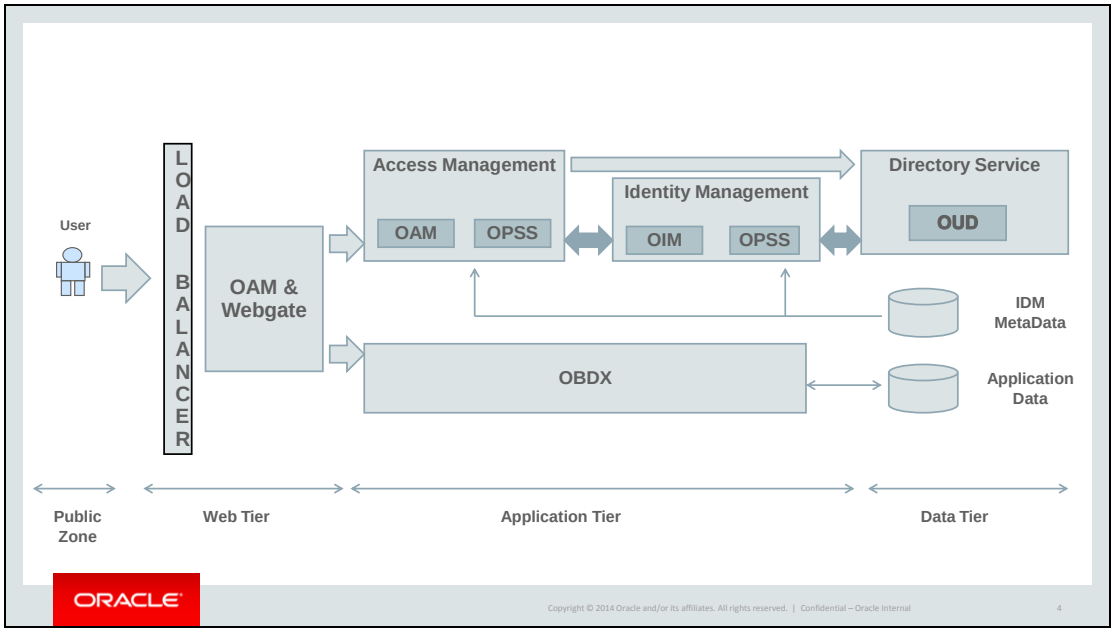
2

Secure Installation and Configuration

This chapter provides an overview of the architecture of the deployment and describes the installation and configuration procedure for Oracle Banking Digital Experience.

Please note that this is only a guide to securing the Oracle Banking Digital Experience application and does not replace periodic reviews of the security architecture of the entire ecosystem of multiple applications maintained by the customer. The guidance provided in this document must always be augmented by specific understanding of the security considerations of the specific deployment architecture.

2.1 Architecture Diagram



2.2 Installing WebLogic

Installation of Weblogic Server can be done by referring to the documentation published at https://docs.oracle.com/cd/E24329_01/doc.1211/e24492/toc.htm.

2.3 Configuring SSL

One way SSL between the presentation tier and the application on WebLogic server is supported. The detailed configuration is explained below:

Note: Procure an external CA signed certificate before proceeding further. Follow the instructions below to install the certificate once the certificate is available

1. Import the Certificate into a Java Trust Keystore

   Execute the following command:

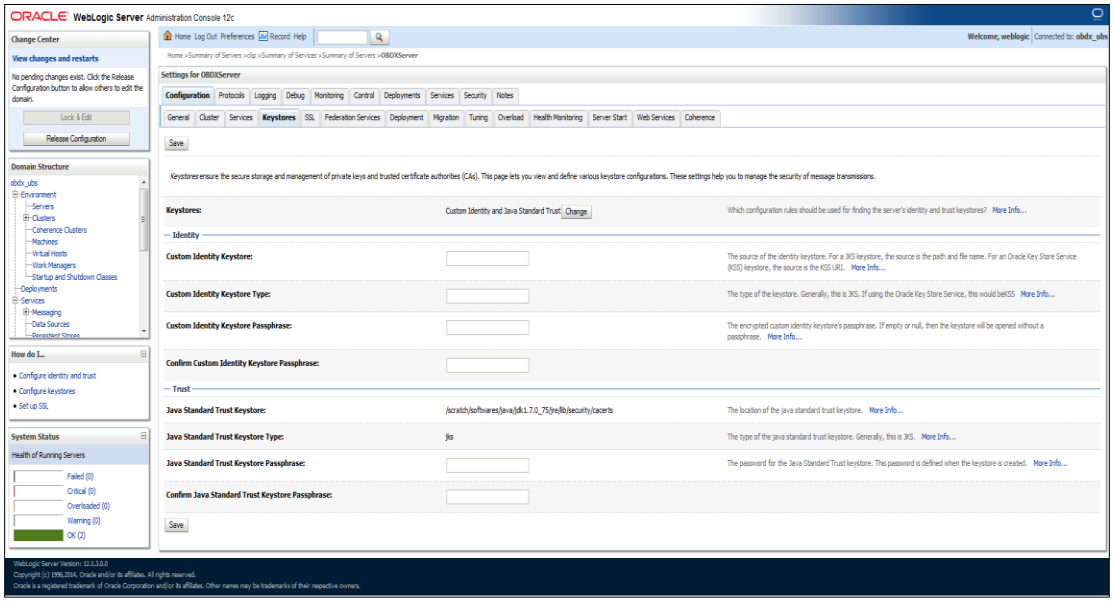
   ```
   keytool -import -trustcacerts -alias sampletrustself -keystore
   SampleTrust.jks -file SampleSelfCA.cer.der -keyalg RSA
   ```

   ```
   keytool -import -alias `hostname -f` -file `hostname -f`.cer -keystore
   <JAVA_HOME>/jre/lib/security/cacerts -storepass changeit -noprompt
   ```

2. Configure Application Domain's Weblogic with Custom Identity and Trust Keystores

   a. Open the WebLogic admin console and navigate to *Home --> Summary of Servers --> AdminServer*.

   b. Click the Keystores tab.



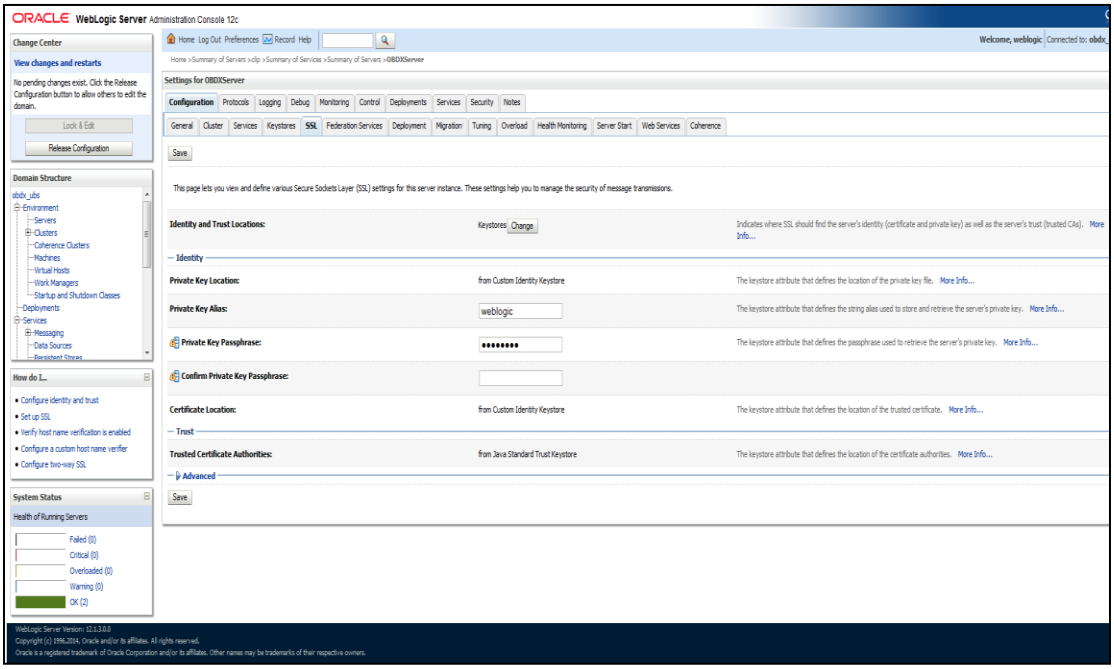
- Click the Change button.
- Select Custom Identity and Java Standard Trust option from the list.
- Click the Save button.
- Enter the following details in the Identity and Trust sections:

Field	Value
Custom Identity Keystore	Absolute path of the custom keystore
Custom Identity KeyStore Type	JCEKS
Custom Identity KeyStore Passphrase	<Passphrase>
Confirm Custom Identity KeyStore Passphrase	<Re-enter the same Passphrase>

Enter the passphrases that were used while creating the custom Identity Keystore and certificate.

c. Click the Save button.

d. Click the SSL Tab.



Enter the following details in the Identity section:

Field	Value
Private Key Alias	<Alias>
Private Key Passphrase	<Passphrase>
Confirm Private Key Passphrase	<Re-enter passphrase>

i. Enter the passphrases that were used while creating the certificate.

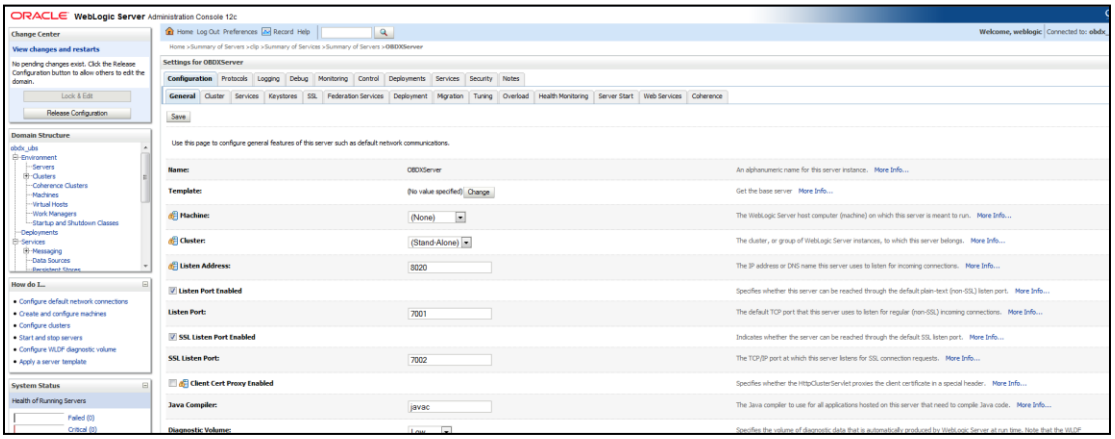
ii. Click the Save button.

iii. Click the Advanced link.

iv. Ensure that Two Way Client Cert Behavior is set to Client Certs Not Requested.

e. Click the General tab.

f. Select the SSL Listen Port Enabled check box.



g. Click the Save button.

2.4 Disable SSLv3

By default, SSLv3 should be disabled.

Specifying the `weblogic.security.SSL.protocolVersion` system property in a command-line argument that starts the WebLogic Server lets you specify the protocol that is used for SSL connections.

The following command-line arguments can be specified so that WebLogic Server supports only TLS connections:

```
-Dweblogic.security.SSL.protocolVersion=TLS1
```

Note: If you don't specify the above property, weblogic assumes SSLv3 by default.

2.5 HTTP Response Header Configurations

The following are some HTTP Response Headers that mitigate certain vulnerabilities.

Vulnerability	HTTP Response Header
Clickjacking	`X-Frame-Options`
XSS	`Content-Security-Policy`
	`X-XSS-Protection`
Cookie hijacking Protocol Downgrade attacks	`Strict-Transport-Security`
Retrieving Sensitive data from browser cache	`Cache-Control`

The sections below specify how to configure these response headers in the `httpd.conf` file of the web server.

2.5.1 X-Frame-Options

```
Header always append X-Frame-Options SAMEORIGIN
```

2.5.2 Content-Security-Policy

```
Header set Content-Security-Policy "default-src 'none'; img-src 'self';
script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self'
https://fonts.googleapis.com 'unsafe-inline'; object-src 'none'; frame-
src 'none'; font-src 'self' https://fonts.gstatic.com; connect-src 'self'
http://<OAM Server>:<OAM Port>/; child-src 'self'"
```

Please note that the policy mentioned here is for the base product. If the product gets customized and content from different URLs needs to be allowed to be executed by the browser, then this policy will have to be modified accordingly.

2.5.3 X-XSS-Protection

```
Header set X-XSS-Protection "1; mode=block"
```

2.5.4 Strict-Transport-Security

Set this for your top level domain. The header directive needs to be included inside the VirtualHost directive

```
<VirtualHost *:443>
  Header always set Strict-Transport-Security "max-
age=31540000; includeSubDomains"
</VirtualHost>
```

Consider submitting your website to be included in the HSTS preload list of websites maintained by Google Chrome at https://hstspreload.appspot.com/ .
Other browsers like MS IE 11, MS Edge, Firefox and Opera also refer to this list maintained by Google and therefore the security offered by this mechanism will extend to other browsers too.

2.5.5 Cache-Control

```
Header set Cache-Control "max-age=0, no-cache, no-store, must-revalidate"
Header set Pragma "no-cache"
Header set Expires 0
```

2.6 Password Policy Guidelines

Our recommendations for setting a password policy are in line with the latest recommendations from NIST as of October 2016.

1) The minimum length of a password must be at least 8 characters. You can choose to increase this number to 10 or 12.
2) The maximum length of a password must be at least 64 characters. You can choose to increase this number to 80 or 100.
3) Do not cause passwords to expire without reason. A password must be expired only when the user has forgotten it and has requested a reset.
4) Allow all printable ASCII characters, including spaces, and accept all UNICODE characters too.
5) Do not force the user to use a combination of upper case characters, lower case characters, numbers and special characters.
   Instead recommend to him that he uses "passphrases" instead of passwords, and that's the reason why the recommended minimum length must be at least 8 and the maximum length must be at least 64. Passphrases are sentences like "*Wow, I like the freedom to choose this password!!*" (yes, with spaces, a comma and exclamation marks in it)

2.6.1 Password Policy Configuration

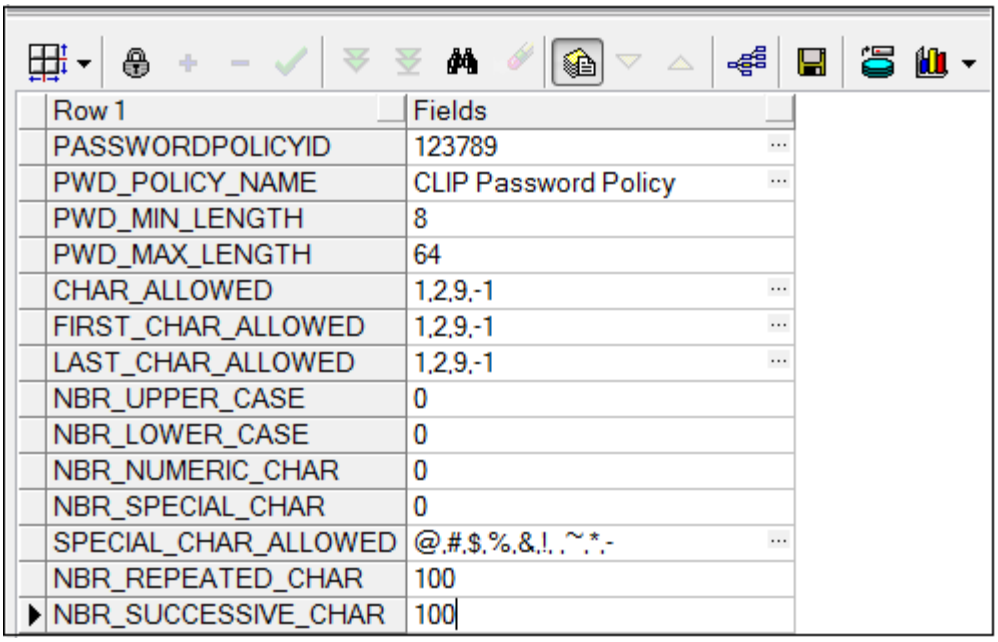
The password policy needs to be maintained in the Oracle Banking Digital Experience database. We understand that you might already have the password policy set in an existing setup of an LDAP user repository. The same needs to be in synch with the password policy set in the Oracle Banking Digital Experience database.

Oracle Banking Digital Experience itself generates passwords during operations like "Password Reset" and it needs access to the password policy during these operations. And that's why the password policies in the 2 systems need to be in synch with each other.

The password policy is set to be set in the database table `DIGX_UM_PWD_POLICY`. The following table lists the columns of the table and significance of each column:

Column Name	Significance	
PASSWORDPOLICYID	Unique Identifier for the password policy.	
PWD_POLICY_NAME	Human friendly name given to the policy.	
PWD_MIN_LENGTH	Minimum length of the password.	
PWD_MAX_LENGTH	Maximum length of the password.	
CHAR_ALLOWED	Characters allowed in the password.	
	1	Upper Case Letters
	2	Lower Case Letters
	9	Numbers
	-1	Special Characters
FIRST_CHAR_ALLOWED	Characters allowed as the first character of a password.	
LAST_CHAR_ALLOWED	Characters allowed as the last character of a password.	
NBR_UPPER_CASE	Minimum number of upper case characters required in the password.	
NBR_LOWER_CASE	Minimum number of lower case characters required in the password.	
NBR_NUMERIC_CHAR	Minimum number of numeric characters required in the password.	
NBR_SPECIAL_CHAR	Minimum number of special characters required in the password.	
SPECIAL_CHAR_ALLOWED	List of special characters allowed in the password.	
NBR_REPEATED_CHAR	Maximum number of times a character is repeated in the password.	
NBR_SUCCESSIVE_CHAR	Maximum number of times a character can be repeated successively in the password.	

The following image shows the recommended values for the key columns of the database table `DIGX_UM_PWD_POLICY`



Row 1	Fields
PASSWORDPOLICYID	123789
PWD_POLICY_NAME	CLIP Password Policy
PWD_MIN_LENGTH	8
PWD_MAX_LENGTH	64
CHAR_ALLOWED	1,2,9,-1
FIRST_CHAR_ALLOWED	1,2,9,-1
LAST_CHAR_ALLOWED	1,2,9,-1
NBR_UPPER_CASE	0
NBR_LOWER_CASE	0
NBR_NUMERIC_CHAR	0
NBR_SPECIAL_CHAR	0
SPECIAL_CHAR_ALLOWED	@,#,$,%,&,!, ,~,*,-
NBR_REPEATED_CHAR	100
NBR_SUCCESSIVE_CHAR	100

Note:

1) The password policy ID and name are at your discretion and no specific recommendations are being made for them.
2) A different password policy can be set for each user group that you define. The password policy applicable for a particular user group can be defined in the table `DIGX_UM_PWD_GROUP_MAP`. That's where the password policy ID will come in handy.
3) If you want the same password policy to be applicable for all groups then you can simply define just one policy in `DIGX_UM_PWD_POLICY`. You can leave the mapping table `DIGX_UM_PWD_GROUP_MAP` empty.

2.7 OTP for Login

Oracle Banking Digital Experience supports a 2nd factor of authentication during login in the form of a One Time Password (OTP). It is a configuration that can be turned on or turned off for a user role.

We recommend that you make use of this feature and keep it turned on for all user roles.

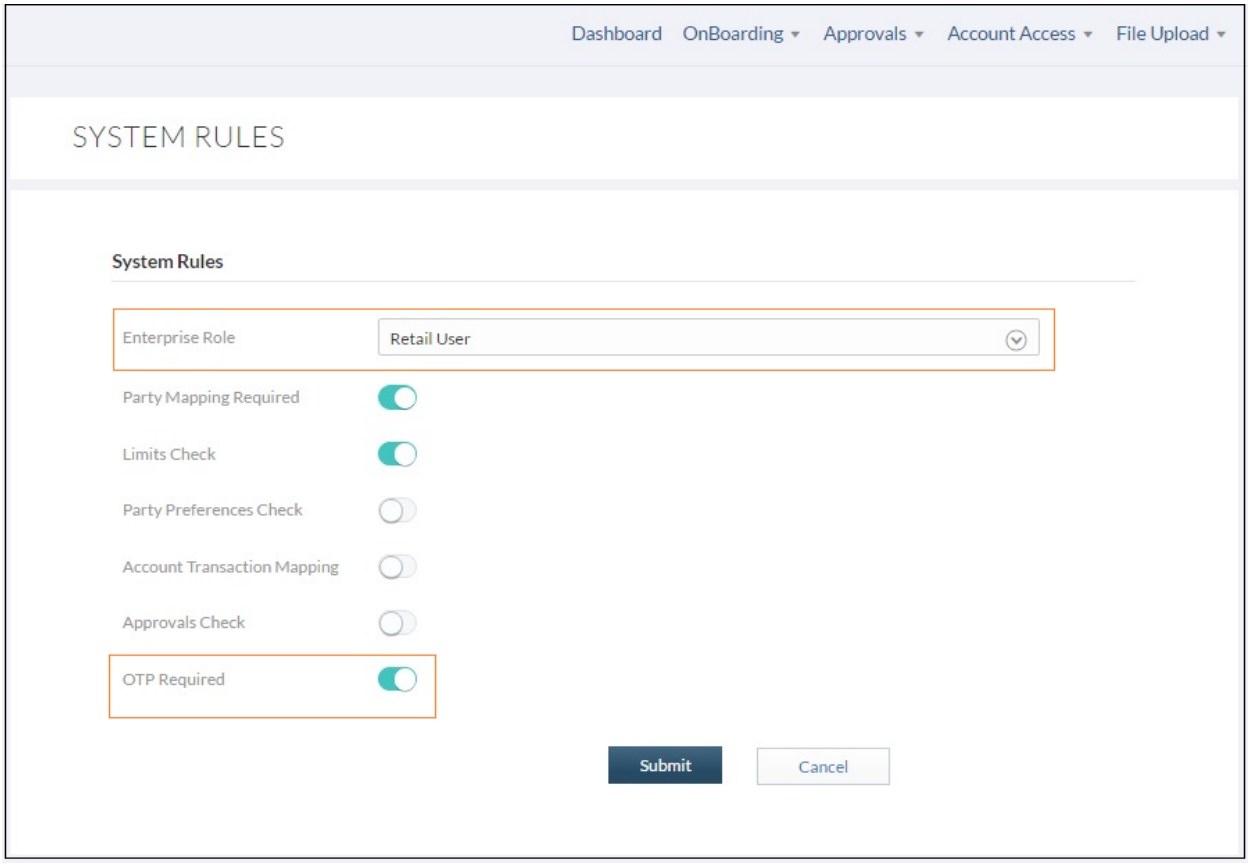
To turn on this feature follow the steps below:

1) Fire the following script on the database.

```
update digx_fw_config_all_b set prop_value='true'
where category_id = 'SecurityConstants' and
prop_id = 'IS_LOGIN_TFA_ENABLED';
```

2) Restart the application server.

3) Now you need to login as an application administrator and turn on the flag called “OTP Required” under System Rules. You can select the user role for which the flag needs to be turned on, from the dropdown.



After this, a user with the role for which Login OTP has been turned on, will need to enter an OTP as a 2nd factor authentication after he enters a valid user name and password in order to be successfully authenticated into the application. The OTP will be sent to the user via SMS/Email.

3

Guidance for Implementation Teams

3.1 CSRF Mitigation – Generating Nonces

A nonce is a pseudo random number that may be used only once. If a nonce is sent across in every request from the client to the server and the server validates the sent nonce every single time, then it mitigates the risk of Cross Site Request Forgery (CSRF).

The product provides a REST Service to generate nonces – each nonce can be used only once to identify each request uniquely, for each session. The product also has an inbuilt framework that will validate the nonce sent in the request.

Therefore post a successful login you need to make a call to `https://<Host>:<Port>/digx/v1/session/nonce` before you make a call to any other service. This service will return back an array of nonces in the response header. You can pick up any one nonce from the array and use it to send across the nonce required in a subsequent request. A nonce can be used only once. You need to discard it after usage.

Please note that unless you send across a nonce, the services that are accessed post login will not work.

3.2 Indirect Object Reference Implementation

3.2.1 What it means

It is a good security practice to hide sensitive data objects from the end user. Although the system needs to play around with sensitive data objects, it is recommended to refer to these sensitive data objects via pointers – tokens that temporarily point to the sensitive data objects but themselves do not contain any sensitive data.

For example consider a credit card application on the web which offers the following 2 transactions:

- Credit Cards Summary – Displays a list of all credit cards the user owns.
- Credit Card Details – Displays the details of one specific Credit Card that the user selects

The Credit Cards Summary page will typically list all credit card numbers in a masked format. Let's assume that the end user holds 2 Credit Cards C1 and C2. When the end user hits the Summary link, the server returns back the following in its response:

a. Masked Credit Card Number C1 (visible to the user)

b. Masked Credit Card Number C2 (visible to the user)

c. Token T1 (not visible to the user)

d. Token T2 (not visible to the user)

T1 and T2 are random tokens – difficult to guess – which the server has generated as proxies for C1 and C2 respectively. The server has internally stored this mapping of C1-T1 and C2-T2 somewhere. Please note that T1 and T2 are tied to the current session. The

moment the session expires, T1 and T2 get discarded. Next time the user logs in, the server generates different tokens T1x and T2x for C1 and C2 respectively.

Whenever the user clicks on say Credit Card Details for C1, the client sends T1 to the server instead of C1, as a request parameter. The server internally figures out that the request is actually for C1 and processes the request accordingly.

Thus we refer to sensitive data indirectly via tokens that are generated with different values for every session.

3.2.2 How OBDX supports it

To implement the above mechanisms the framework offers interception of both the request and the response. The following steps need to be taken:

1) The Service Request and Response DTOs need to implement the interface `com.ofss.digx.app.framework.IIndirection`

   This is how the Interface Definition looks like:

```
package com.ofss.digx.app.framework;

import java.io.Serializable;

/**
 * By inheriting this interface one can convert sensitive data(Parameters of response and request DTOs) to such data
 * that have respective MASKED and INDIRECTED value instead original value.
 *
 */
public interface IIndirection extends Serializable {
    /**
     * Convert original value into respective MASKED and INDIRECTED value for Objects that have sensitive
     * data(Parameters of response DTOs) .
     *
     * @param sessionId
     *            - session Id of the request.
     */
    public void indirectResponse(String sessionId);

    /**
     * Convert sensitive data(Parameters of request DTOs) like party id to such data that have respective MASKED and
     * INDIRECTED value instead of original value.
     *
     * @param sessionId
     *            - Object of Session
     */
    public void indirectRequest(String sessionId);

}
```

2) Before processing the request the system needs to read the indirect value and convert it to the actual value. Before processing the response the system needs to read the actual value, pick up the corresponding indirect value and pass that back in the response.
   Therefore the Service Request DTO needs to implement the method `indirectRequest()`
   and the Service Response DTO needs to implement the method `indirectResponse()`

3) Import the following class in both the Request and the Response DTO as follows:

```
import com.ofss.digx.framework.security.indirection.handlers.IndirectionHandler;
```

4) Define the following member variable in both Request and Response DTOs

```
private IndirectionHandler indirectionHandler = new IndirectionHandler();
```

5) Let's assume that the sensitive data that we want to refer indirectly to, is a field defined as follows in the DTO:

```
private String accountNo;
```

6) The `indirectResponse()` method needs to look like the following sample piece of code

```
@Override
public void indirectResponse(String sessionId) {
    try {

        accountNo = indirectionHandler.create(sessionId, accountNo, IndirectionType.ACCOUNT_ID);

    } catch (Exception e) {
        logger.log(
                Level.SEVERE,
                formatter.formatMessage("Exception occurred in implementing Indirection."),
                e);
    }

}
```

7) The `indirectRequest()` method needs to look like the following sample piece of code

```
@Override
public void indirectRequest(String sessionId) {
    try {

        accountNo = indirectionHandler.retrieve(sessionId, accountNo);

    } catch (Exception e) {
        logger.log(
                Level.SEVERE,
                    formatter.formatMessage("Exception occurred in implementing Indirection."),
                e);
    }
}
```

3.3 Output Encoding

To mitigate inline Cross Site Scripting attacks, the product provides a framework to encode the data sent in the response.

To make use of this framework you need to define something called as an Encoder class – 1 Encoder for 1 Response DTO. Therefore if you have 5 Response DTOs, then you need to write 5 Encoder classes, one for each Response DTO. Also, if you have DTOs defined as class member variables within a Response DTO, then you need to define Encoders for those DTOs too, 1 Encoder class for 1 DTO.

1) Every Response DTO class must extend `com.ofss.digx.service.response.BaseResponseObject`
2) Every Response DTO class must define a corresponding Encoder class.
3) The Encoder class must extend `com.ofss.digx.infra.esapi.AbstractDataEncoder<T>` and must override the method `public T encode(T t)`
   `T` is your Response DTO Class and `t` is an object of that class.

This is how `AbstractDataEncoder` looks like:

```
package com.ofss.digx.infra.esapi;

/**
 * Provides functionality to encode a BaseResponseObject using ESAPI. This class is extended by all the DTOEncoders.
 *
 * @param <T>
 *            Generic to specify encoder of which class to be instantiated.
 */
public abstract class AbstractDataEncoder<T> {
    /**
     * {@link com.ofss.digx.esapi.security.encoderdecoder.ESAPIHelper} instance to provide OWASP ESAPI encoding and
     * decoding. The DTOencoders extending the {@link AbstractDataEncoder} uses this instance to access the
     * functionality of OWASP ESAPI encode method.
     */
    protected ESAPIHelper esapiHelper = new ESAPIHelper();

    /**
     * Provides mechanism to encode the given Data Transfer object. It encodes all the {@code java.lang.String}
     * parameter inside the object passed using OWASP ESAPI encoding. Apart from {@code java.lang.String}, it also
     * provides encoding of inner- objects (if there are any) by calling the DTOEncoder for the respective inner-
     * objects.
     *
     * @param responseDTO
     *            The generic Response DTO object to encode.
     * @return A generic encoded Response DTO object.
     *
     */
    public abstract T encode(T responseDTO);

}
```

The method `encode()` that you implement must contain code to encode every attribute of the DTO explicitly.

The class `ESAPIHelper` defines and implements the following method
`public String encode(String param)`

You need to invoke the method `esapiHelper.encode()` for each attribute of the DTO and set the output of this method back in the attribute on which it was invoked.

3.3.1 An Example

As an example, consider the Response DTO named `MyListResponse`. This is how the class looks like:

```
import java.util.List;
import com.ofss.digx.service.response.BaseResponseObject;


public class MyListResponse extends BaseResponseObject {


    private List<MyDTO> list;

    public List<MyDTO> getList() {
        return list;
    }

    public void setList(List<MyDTO> list) {
        this.list = list;
    }

}
```

This response DTO contains 1 member variable which is a list that contains objects of the class `MyDTO`.

Let's take a look at how the DTO `MyDTO` looks like:

```
import com.ofss.fc.framework.domain.common.dto.DataTransferObject;

public class MyDTO extends DataTransferObject {


    private String id;

    private String name;

    private String nickName;

    private String partyId;


    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getNickName() {
        return nickName;
    }

    public void setNickName(String nickName) {
        this.nickName = nickName;
    }

    public String getPartyId() {
        return partyId;
    }

    public void setPartyId(String partyId) {
        this.partyId = partyId;
    }

}
```

As explained earlier, member variables that themselves are DTOs must also have Encoder classes written for them. Therefore there is an Encoder class written for `MyDTO` . This is how the class `MyDTOEncoder` looks like:

```
import java.util.logging.Level;


public class MyDTOEncoder extends AbstractDataEncoder<MyDTO> {
    /**
     * Name of the entity(class) represented by this {@code Class} object as a {@code String}
     */
    private static final String THIS_COMPONENT_NAME = MyDTOEncoder.class.getName();

     * Create instance of multi-entity logger
    private MultiEntityLogger formatter = MultiEntityLogger.getUniqueInstance();

     * This is an instance variable which is required to support multi-entity wide logging.
    private static final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);

     * Public Constructor.
    public MyDTOEncoder() {
    }


    @Override
    public MyDTO encode(MyDTO response) {

        try {
            response.setId(esapiHelper.encode(response.getId()));
            response.setName(esapiHelper.encode(response.getName()));
            response.setNickName(esapiHelper.encode(response.getNickName()));
            response.setPartyId(esapiHelper.encode(response.getPartyId()));


        } catch (Exception e) {
            logger.log(Level.SEVERE,
                    formatter.formatMessage("Fatal error while encoding object."), e);
        }

        return response;
    }

}
```

And finally this is how the Encoder Class `MyListResponseEncoder` for the Response DTO Class `MyListResponse` looks like:

```
import java.util.logging.Level;


public class MyListResponseEncoder extends AbstractDataEncoder< MyListResponse >
{
     * Name of the entity(class) represented by this {@code Class} object as a {@code String}
    private static final String THIS_COMPONENT_NAME = MyListResponseEncoder.class.getName();

     * Create instance of multi-entity logger
    private MultiEntityLogger formatter = MultiEntityLogger.getUniqueInstance();

     * This is an instance variable which is required to support multi-entity wide logging.
    private static final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);
     * Public Constructor.
    public MyListResponseEncoder() {
    }


    @Override
    public MyListResponse encode(MyListResponse myListResponse) {
        try {
            if(myListResponse.getList() != null ) {
                MyDTO dto = null;
                for(int i=0; i< myListResponse.getList().size() ; i++) {
                    dto = myListResponse.getList().get(i);
                    dto = esapiHelper.encodeObject(dto);
                }
                myListResponse.setList(myListResponse.getList());
            }
        } catch(Exception e) {
            logger.log(Level.SEVERE, formatter.formatMessage("Fatal error while encoding MyListResponse object."), e);
        }
        return myListResponse;
  }
}
```

3.4 Configuring the Hashing Algorithm for OTPs

The hashing algorithm used to hash the One Time Passwords used for payments and other transactions is configurable.

This has been made configurable since Cryptography is an ever changing field. Algorithms that are considered as secure today might be rendered insecure tomorrow by attackers doing research in cryptography. Making the hashing algorithm configurable will help the customer upgrade to the latest algorithm prevailing at the time in the future, with reasonable ease.

Configuration Steps:

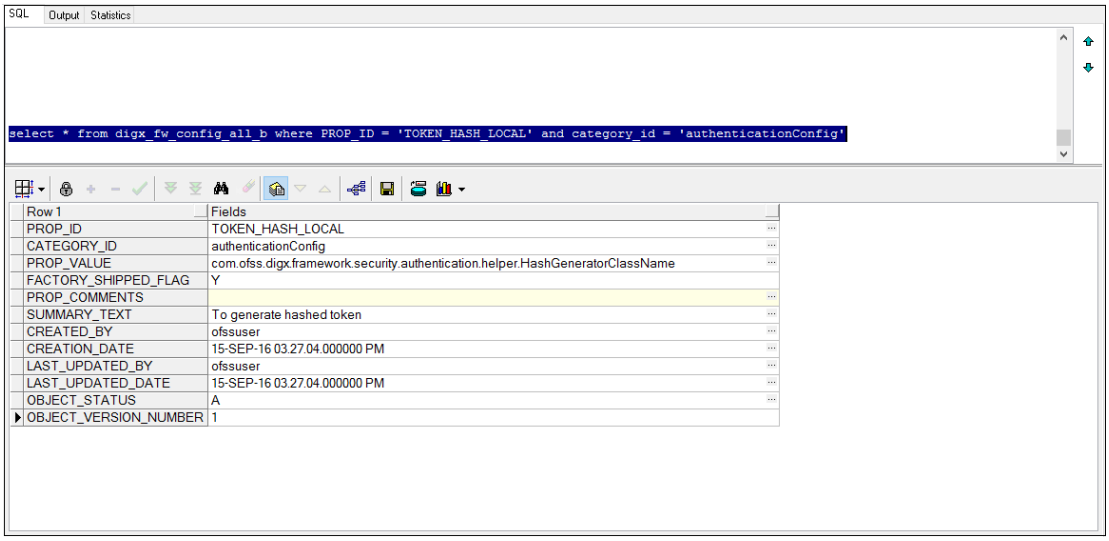
1) Implement your own java class that contains the code to generate the hash.
2) The class should implement the interface `IHashHelper.`

```
package com.ofss.digx.framework.security.authentication.helper;

 * Hashing is the transformation of a string of characters into a fixed-length value or key that represents the original
public interface IHashHelper {
    /**
     * Creates the hash value for the token. Hashing is one way encryption. Every string has a unique hash value. To add
     * extra security we add secret to original token value while creating hash.
     *
     * @param token
     *            {@link String} Value that needs to be hashed
     * @param secret
     *            {@link String} Salt added for hashing.
     * @return {@link String} type hashed value
     */
    public String createHash(String token, String secret);

}
```

3) Add/Edit the value of the column PROP_VALUE in the following entry in the database table DIGX_FW_CONFIG_ALL_B (PROP_ID = 'TOKEN_HASH_LOCAL' and CATEGORY_ID = 'authenticationConfig'). You need to provide the fully qualified class name of your java class.



Row 1	Fields
PROP_ID	TOKEN_HASH_LOCAL
CATEGORY_ID	authenticationConfig
PROP_VALUE	com.ofss.digx.framework.security.authentication.helper.HashGeneratorClassName
FACTORY_SHIPPED_FLAG	Y
PROP_COMMENTS	
SUMMARY_TEXT	To generate hashed token
CREATED_BY	ofssuser
CREATION_DATE	15-SEP-16 03.27.04.000000 PM
LAST_UPDATED_BY	ofssuser
LAST_UPDATED_DATE	15-SEP-16 03.27.04.000000 PM
OBJECT_STATUS	A
OBJECT_VERSION_NUMBER	1

4) Needless to say, an application server restart is needed.