

Ozone Configuration Guide

DOD GOSS

Version 8.0.0.0, 2019-07-01

Table of Contents

1. Introduction	1
1.1. Objectives	1
1.2. Document Scope	1
1.3. Related Documents	1
2. Overview	2
2.1. Purpose	2
2.2. Dependencies	2
2.3. Components	2
2.3.1. Ozone Widget Framework (OWF) Web Application	2
2.3.2. Pluggable Security	2
2.3.2.1. Example Security Configuration	2
2.3.2.1.1. Username and Password Security (Default)	2
2.3.2.1.2. Certificate-only (X509) Security	3
2.3.2.1.3. Central Authentication Service (CAS) Security	3
2.3.2.2. Additional Resources (OWF Security Project)	3
2.3.3. Sample Widgets	3
3. Installation	4
3.1. Dependencies	4
3.2. Supported Browsers	4
3.3. Bundle Overview	4
3.3.1. Bundle Contents	4
3.3.2. Running the Application	5
3.4. Default Installation	5
3.5. Custom Installation	6
4. Database Configuration	7
4.1. Using Oracle	9
4.2. Using MySQL	10
4.2.1. Configuring MySQL JDBC to use SSL	11
4.2.1.1. Creating MySQL Certificates	11
4.2.1.2. Configure MySQL	12
4.2.1.3. Configure the Franchise Store Bundle	13
4.3. Using SQL Server	14
5. Security	16
5.1. Overview	16
5.1.1. Basic Security Concepts and OWF	16
5.1.2. Production Deployments	16
5.2. Default Security Configuration	17
5.2.1. Adding Users, Roles, & Groups	17
5.3. Sample Security Modules	18
5.3.1. Installing Security Modules	19
5.3.2. X.509 Certificate (PKI) Only	19
5.3.2.1. Installing User Certificates (PKI)	20
6. Configuration	21
6.1. Environments settings	21

6.2. Grails	21
6.3. Security settings	22
6.4. Hibernate settings	22
6.5. Quartz settings	23
6.6. Data Sources settings	23
6.7. Server settings	24
6.8. OWF settings	25
6.8.1. Help settings	25
6.8.1.1. Changing the location of help files	25
6.8.1.2. Custom Help file types	26
6.8.2. Notification settings	26
6.8.3. Custom Data Guard and Widget Security	27
6.8.4. Store Synchronization settings	28
6.8.4.1. Automatically add the Store listings and updates	29
6.8.4.2. Enabling Auto-Updates from the Store	30
6.8.4.3. Widget Load Times	30
6.8.5. Custom Access Alert settings	30
6.8.6. Logout settings	32
6.8.7. Auto Save settings	33
7. Logging	34
7.1. Logging Configuration	34
7.2. Audit Logging	36
7.2.1. Sign-in Events	36
7.2.2. Logout Events	37
7.3. Common Event Format (CEF) Auditing	38
8. Upgrading	40
8.1. Upgrading from version 7.17.1.0	40
8.2. Connecting to OMP (Marketplace)	40
9. Deployment	42
9.1. Operating OWF From Different/Multiple Ports	42
9.2. Adding the Store To OWF	43
9.3. Server Certificate Creation and Installation	43
9.3.1. Generating a New Self-Signed Server Certificate	43
9.3.2. Configuring Ozone For a Different Truststore/Keystore	44
Appendix A: Custom Security Modules	46
A.1. Requirements for Customizing Security	46
A.2. Custom Security Classes	47
A.2.1. OWFUserDetails	47
A.2.2. OWFUserDetailsImpl	48
A.2.3. OWFGroup	48
A.2.4. OwfGroupImpl	48
A.2.5. GrantedAuthorityImpl	48
A.2.6. MyDetailsService	49
Glossary	50

1. Introduction

1.1. Objectives

This guide covers topics relevant to installing and configuring the Ozone Widget Framework (OWF).

1.2. Document Scope

This guide is intended for OWF developers who wish to configure or customize an OWF instance.

For the purpose of this document, a developer is understood to be someone who is comfortable unpacking and packing WAR files and editing configuration files.

In this document, the term Store and Marketplace are used interchangeably.

1.3. Related Documents

Table 1. Related Documents

Document	Purpose
Quick Start Guide	Walkthrough of basic OWF functions such as using widgets; unpacking the OWF bundle; setting up a local instance of OWF; installing security certificates; truststore and keystore configuration.
User's Guide	Understanding the OWF user interface; adding, deleting, modifying widgets and using intents; accessing and using the Store; using dashboards; creating, deleting, adding, switching, modifying dashboard pages; defining accessibility features such as high-contrast themes.
Administrator's Guide	Understanding administrative tools: adding, deleting, and editing users, groups, widgets, and dashboards; creating default content for users, groups and group dashboards.
Configuration Guide	Overview of basic architecture and security; OWF installation instructions; instructions for modifying default settings; database set up and logging guidance; framework and theme customization instructions; OWF upgrade instructions; directions for adding and deleting help content.

2. Overview

2.1. Purpose

The Ozone Widget Framework (OWF) is a set of tools, generally delivered in the OWF Bundle. When deployed, OWF is used for organizing and displaying Web applications (widgets) in a single browser window known as an OZONE dashboard.

2.2. Dependencies

The OWF Bundle is shipped with Tomcat v8.5.23 and requires Java 8 or higher. If running OWF with a Web server other than Tomcat, see that Web server's documentation for any additional requirements.

2.3. Components

2.3.1. Ozone Widget Framework (OWF) Web Application

owf.war

This file, located in the `/tomcat/webapps/` directory, contains the components which make up the OWF application. Whether a user logs in and accesses the framework, or an administrator logs in to modify preferences, the `owf.war` is the application that launches those pages to the browser.

2.3.2. Pluggable Security

OWF allows an administrator to customize the type of security that will be implemented for user authentication and authorization.

2.3.2.1. Example Security Configuration

Included within the OWF bundle and framework `/tomcat/lib/ozone/framework/` is the default `security.xml` Spring configuration file, also within `/tomcat/lib/ozone/framework/security/` are additional example configurations files.

These default and example security files are intended ONLY as examples and should NOT be used in a production environment.

Please refer to the [Chapter 5, Security](#) section of this guide for further details about OWF Security.

2.3.2.1.1. Username and Password Security (Default)

security.xml

Contains the default sample security implementation that uses a login form requiring a username and password.

2.3.2.1.2. Certificate-only (X509) Security

security_cert-only.xml

Contains the sample certificate-only (X509) security implementation that uses PKI certificates for client authentication. If no authentication is provided, the user is denied access to the system.

2.3.2.1.3. Central Authentication Service (CAS) Security

security_cas-only.xml

Contains the sample CAS security implementation that uses an external authentication service for client authentication.

2.3.2.2. Additional Resources (OWF Security Project)

For building custom security implementations, the **owf-security** project may be referenced or used as a starting point. The source code for this project is available at the Ozone Platform organization GitHub repositories located at <https://github.com/ozoneplatform/>.

In previous releases, these files were distributed in the bundle in the **ozone-security-project.zip** file. As described above, the **owf-security** project repository is the new location for the latest versions of these files.

2.3.3. Sample Widgets

A set of example widgets are provided with the system in order to demonstrate the functionality of the widget APIs.

The example widgets are enabled by default, and run embedded within the main application inside the **owf.war** file.

In previous OWF releases, an additional set of sample widgets utilizing various web technologies and frameworks was provided in the **owf-sample-widgets.zip** file. Due to the declining prevalence of these technologies, this file is no longer being distributed with the bundle. However, the sources for these sample widgets are still available within the **owf-framework** repository, located under the **/sample-widgets/** directory.

The OWF Developer's Guide includes specific examples and guides regarding the developing widgets and utilizing the widget APIs.

3. Installation

3.1. Dependencies

Listed below are the dependencies for OWF:

- Java 8 or higher.
- A Relational Database Management System (RDBMS). OWF currently ships with an in-memory HyperSQL (HSQLDB) database for testing and development purposes, but it is expected that a live deployment will use a more robust RDBMS such as Oracle or MySQL.

3.2. Supported Browsers

OWF is tested against the following browsers:

Table 2. Supported Browsers

Browser	Version(s)
Internet Explorer	> 11
Edge	> 44
Chrome	> 74
Firefox	> 60

3.3. Bundle Overview

3.3.1. Bundle Contents

The distribution of OWF consists of a ZIP file containing the necessary components to set up and run OWF in a development environment. The bundle contains the following:

- **dbscripts/** – SQL scripts to create the database schema and the initial data.
- **docs/** – Copies of the application documentation and guides.
- **drivers/** – Example database drivers.
- **tomcat/** – The pre-configured Tomcat container. Includes the custom **start.bat** and **start.sh** scripts.
- **tomcat/certs/** – Sample server and user certificates for SSL and certificate-based authentication
- **tomcat/conf/** – Tomcat configuration files.
- **tomcat/lib/** – Files to include in the Tomcat Java classpath.
- **tomcat/lib/config/** – Example OWF configuration files for the supported databases.

- `tomcat/lib/ozone/framework/` – Externalized OWF configuration files, including custom security configuration.
- `tomcat/webapps/owf.war` – The OWF web application WAR file.
- `owf-jboss-modules.zip` – Sample JBoss modules for deployment onto a JBoss container.

3.3.2. Running the Application

The following example shows how an administrator might copy, unzip and start Ozone from the bundle deployment on *nix operating systems, assuming the bundle is named `ozone-framework-8.0.0.0.zip`:

```
cp ozone-framework-8.0.0.0.zip /opt
cd /opt
unzip ozone-framework-8.0.0.0.zip
cd tomcat
./start.sh
```

The following example shows how an administrator might copy, unzip, and start OWF from the bundle on Windows operating systems, assuming the bundle is named `ozone-framework-8.0.0.0.zip`:

1. Create a new directory (for example, `C:\owf\`) from where OWF will be run. This can be done via the Windows UI or a command prompt.
2. Copy `ozone-framework-8.0.0.0.zip` to the new directory created in step 1.
3. Right-click on `ozone-framework-8.0.0.0.zip` and select Open, Explore, or the command for the system's default zip/unzip program.
4. Unzip/unpack the bundle into the new directory created in step 1.
5. From a command-line, run `start.bat` from within the `C:\owf\tomcat\` directory.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 8+ enabled system.

3.4. Default Installation

Running the OWF Bundle via the included Tomcat web server with the default values requires minimal installation and configuration. When using this standard configuration, OWF uses the default security module which provides a simple username and password login form for authentication.

The application uses a keystore and a truststore which are local to the installation. There is no need to install any certificates into the server's Java installation. However, the default certificates contained in the OWF Bundle only function for localhost communications. When accessed from a remote machine with a name that differs from localhost while using the included certificates, OWF will not function correctly. Accordingly, see the Server Certificate Creation and Installation section below for information about creating additional certificates.

3.5. Custom Installation

OWF can be customized to run in a variety of environments.

To configure an external database, see [\[database-setup\]](#).

To configure security settings or install a security module, see [Chapter 5, Security](#).

4. Database Configuration

While the full extent of administering databases is outside the scope of this guide, this section provides information on how to work with databases for OWF.


`application.yml` is a configuration file that allows an administrator to modify database connectivity information. It is located in the `/tomcat/lib/ozone/framework` directory. Once changes are made, restart OWF to apply them.


Listed below are the variable database elements that need to be modified to customize the OWF database. A detailed explanation of each field follows in [Table 3, “OWF Externalized Database Properties”](#).

application.yml

```
dataSource:
  dbCreate: "none"
  username: "sa"
  password: ""
  driverClassName: "org.hsqldb.jdbcDriver"
  url: "jdbc:hsqldb:file:prodDb;shutdown=true"
  pooled: true
  properties:
    minEvictableIdleTimeMillis: 180000
    timeBetweenEvictionRunsMillis: 180000
    numTestsPerEvictionRun: 3
    testOnBorrow: true
    testWhileIdle: true
    testOnReturn: true
    validationQuery: "SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS"
```

Table 3. OWF Externalized Database Properties

Property	Purpose	Example
<code>dbCreate</code>	<div>Whether the database schema is created and/or updated when the server is started.</div> <div> Use the appropriate database creation script in the <code>dbscript</code> directory before running OWF if set this option is set to <code>none</code>.</div>	<code>none / create / update</code>
<code>username</code>	The username for the database connection	<code>admin</code>

Property	Purpose	Example
<code>password</code>	The password for the database connection	<code>password</code>
<code>driverClassName</code>	The fully-qualified class name of the JDBC driver	<code>org.hsqldb.jdbcDriver</code>
<code>url</code>	JDBC Connection String	<code>jdbc:hsqldb:file:prodDb; shutdown=true</code>
<code>pooled</code>	Whether database connection pooling is enabled.	<code>true</code>
<code>minEvictableIdleTimeMillis</code>	The minimum amount of time (in milliseconds) an object can be idle in the pool before becoming eligible for eviction.	<code>18000</code>
<code>timeBetweenEvictionRunsMillis</code>	The time (in milliseconds) to sleep between runs of the idle object evictor thread.	<code>18000</code>
<code>numTestsPerEvictionRun</code>	The number of objects to be examined on each run of the idle evictor thread.	<code>3</code>
<code>testOnBorrow</code>	Whether objects are validated before borrowed from the pool.	<code>true</code>
<code>testWhileIdle</code>	Whether objects are validated by the idle object evictor thread.	<code>true</code>
<code>testOnReturn</code>	Whether objects are validated before returned to the pool.	<code>true</code>
<code>validationQuery</code>	Validation query, used to test connections before use. <div>  Syntax varies by database, see the examples included in this document. </div>	<code>SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS</code>



When setting up databases for OWF, be mindful of the database's lexical sorting mechanism. For some instances of OWF, with a small handful of users, this may not be much of an issue, but as the database becomes more populated, sorting may become increasingly difficult to manage.

4.1. Using Oracle

1. Create an Oracle database user for OWF. It is recommended that there be a dedicated user for OWF to avoid database object name collisions. The OWF team recommends using UTF-8 encoding.
2. Due to licensing issues, OWF does not provide a JDBC driver for Oracle. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
3. Open the `/tomcat/lib/ozone/framework/application.yml` file and modify the `environments.production.dataSource` section using the values that are appropriate for the OWF environment.

Example: application.yml

```
dataSource:
  pooled: true
  dbCreate: "none"
  username: "owf_user"
  password: "owf_password"
  dialect: "org.hibernate.dialect.Oracle10gDialect"
  driverClassName: "oracle.jdbc.driver.OracleDriver"
  url: "jdbc:oracle:thin:@myhost.somewhere.org:1521:DEVDB1"
  properties:
    minEvictableIdleTimeMillis: 180000
    timeBetweenEvictionRunsMillis: 180000
    numTestsPerEvictionRun: 3
    testOnBorrow: true
    testWhileIdle: true
    testOnReturn: true
    validationQuery: "SELECT 1 FROM DUAL"
```

In the example above, an Oracle database-user named `owf_user` with a password of `owf_password` is used for a database named `DEVDB1`.

There are several different types of Oracle drivers (thin, OCI, kprb) and connection options (service, SID, TNSName) available. Please consult the Oracle DBA and Oracle's JDBC documentation to create the connection most appropriate for the installed environment.

4. To create the schema, run the `/dbscripts/oracle/OracleCreate.sql` script, prior to starting OWF.
5. Ensure that the transaction is committed.

Notes:

- If running a production environment, no additional steps are necessary. However, if sample widgets are to be installed, `OraclePrefsUpdate_v7.14.0.sql` must be run prior to logging in. Logging in between the execution of these scripts can cause system failure.

- The OWF team is aware of a known issue with the Oracle Web-based Admin Console returning truncated characters when dealing with large data sets. Accordingly, using SQLPlus to run the script mentioned above is recommended.
- Old versions of the SQL scripts are included under `/database/archive`. These are included for the convenience of those upgrading from older versions of OWF.

4.2. Using MySQL

1. Create a schema within MySQL for use with OWF. It is recommended that there be a dedicated schema for OWF to avoid database object name collisions. The OWF team recommends using UTF-8 encoding.
2. Create a MySQL user with full access to the OWF schema created above.
3. OWF does not provide a JDBC driver for MySQL. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
4. Open the `/tomcat/lib/ozone/framework/application.yml` file and modify the `environments.production.dataSource` section using the values that are appropriate for the OWF environment.

Example: application.yml

```
dataSource:
  pooled: true
  dbCreate: "none"
  driverClassName: "com.mysql.jdbc.Driver"
  url: "jdbc:mysql://myhost.somewhere.org/owf"
  username: "owf_user"
  password: "owf_password"
  dialect: "org.hibernate.dialect.MySQL5InnoDBDialect"
  properties:
    minEvictableIdleTimeMillis: 180000
    timeBetweenEvictionRunsMillis: 180000
    numTestsPerEvictionRun: 3
    testOnBorrow: true
    testWhileIdle: true
    testOnReturn: true
    validationQuery: "SELECT 1"
```

In the example above, a MySQL database user named `owf_user` with a password of `owf_password` is used, for a database named `owf`. The dialect `org.hibernate.dialect.MySQL5InnoDBDialect` will use the InnoDB engine which is recommended for interactive web apps and explicitly used as the engine on OWF create and upgrade scripts.

5. To create the schema, run the `/dbscripts/mysql/MySqlCreate.sql` script, prior to starting OWF.

Notes:

- If manually creating the database objects, be sure to modify the SQL script (mentioned above) with the appropriate schema name.

Example

```
use owf;
```

- If running a production environment, no additional steps are necessary. However, if sample widgets are to be installed, `MySQLPrefsUpdate_v7.14.0.sql` must be run prior to logging in. Logging in between the execution of these scripts can cause system failure.
- Old versions of the SQL scripts are included under `/database/archive`. These are included for convenience for people upgrading from older versions of OWF.

4.2.1. Configuring MySQL JDBC to use SSL

By default MySQL communicates over an unencrypted connection - however, in most cases, it can be configured to use SSL. This is somewhat implementation specific.



This capability completely depends on how the MySQL binaries were compiled, packaged, etc.

The following procedure covers how to configure an SSL capable MySQL server to work with an OWF bundle. The starting point for the server implementation used in this example is:

- Operating System: CentOS 6.4, 64 bit minimal installation (no updates were applied)
- MySQL Server v5.1.69 (achieved by installing the "mysql-server" package with yum)

This procedure was developed from the MySQL 5.1 documentation, specifically the following sections:

- Using SSL for Secure Connections – <http://dev.mysql.com/doc/refman/5.1/en/ssl-connections.html>
- Connector/J (JDBC) Reference – <http://dev.mysql.com/doc/refman/5.1/en/connector-j-reference.html>

This procedure relies on self-signed certificates. It is for testing and demonstration purposes only. The following three sections explain the configuration steps:

4.2.1.1. Creating MySQL Certificates

The following steps guide a developer through creating the CA and Server certificates for a MySQL server:

Step 1: Create the CA key and certificate

From a shell prompt on a MySQL server, type the following commands:

```
openssl genrsa 2048 > ca-key.pem
```

```
openssl req -new -x509 -nodes -days 365 -key ca-key.pem -out cacert.pem
```



After the second command, the system prompts the developer to provide basic identity information. For the purpose of this demonstration, it is not important what information the developer provides here. However, it will be necessary to provide the same information in the next step.

Step 2: Create the server certificate

From a shell prompt on your MySQL server, type the following commands. After the first command, the developer will again be prompted to provide identity information. The developer must provide the same information that they provided in Step 1. However, when prompted for the Common Name, provide the MySQL server's hostname (e.g. mysql.mydomain.com).

```
openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-key.pem -out  
server-req.pem
```

```
openssl rsa -in server-key.pem -out server-key.pem
```

```
openssl x509 -req -in server-req.pem -days 365 -CA cacert.pem -CAkey ca-  
key.pem -set_serial 01 -out server-cert.pem
```

Step 3: Consolidate the output from the previous steps

Copy the following files (produced in the preceding two steps) to a location where at a minimum the MySQL user has read access. In this example, it is `/etc/ssl/certs/`:

- `cacert.pem`
- `server-key.pem`
- `server-cert.pem`

4.2.1.2. Configure MySQL

Step 1: Edit my.cnf

Edit the MySQL configuration file. For this example, the file is located at `/etc/my.cnf`. Add the lines shown in bold to the `[mysqld]` section of the file.

```
ssl-ca=/etc/ssl/certs/cacert.pem
ssl-cert=/etc/ssl/certs/server-cert.pem
ssl-key=/etc/ssl/certs/server-key.pem
```

Step 2: Restart MySQL

Run the following command (or its equivalent) to restart the MySQL server:

```
sudo service mysqld restart
```

Step 3: Create the MySQL franchise store schema

At a minimum, create the schema and assign permissions as you would normally, following the steps in section Adding Users/Roles/Groups. This will leave the choice of using an encrypted connection up to the connecting client. To enforce the use of SSL from the database server side, add **REQUIRE SSL** to the end of the grant statement where user permissions to the Franchise Store schema are assigned.

Example

```
GRANT ALL ON franchise.* TO 'franchise'@'storehost.mydomain.com' IDENTIFIED BY
'franchise' REQUIRE SSL;
```

4.2.1.3. Configure the Franchise Store Bundle

Step 1: Modify application.yml

Add **useSSL=true** to the **environments.production.dataSource.url** configuration. This can also be enforced from the client side with the **requireSSL** option.

Example

```
dataSource:
  url:
    "jdbc:mysql://192.168.56.11/franchise?useSSL=true&requireSSL=true&trustCertificateKeyStoreUrl=file://$/{System.properties['javax.net.ssl.trustStore']}&trustCertificateKeyStorePassword=changeit"""
```

Step 2: Modify the application trust store

Add the CA certificate, created above (**cacert.pem**) to the application's trust store. In the case of the franchise bundle, the trust store is a file called **keystore.jks** found in the **/tomcat/certs/** directory. Do this with the following command (assuming you have the JDK installed and **JAVA_HOME/bin** on your PATH, if they aren't there, add them first).


```
keytool -import -alias mysqlCAcert -file cacert.pem -keystore keystore.jks
```

Step 3: Start the Application

```
./start.sh
```

4.3. Using SQL Server

1. Create a new SQL Server database for use with OWF.
2. Create a SQL Server user with full access to the OWF database created above.
3. OWF does not provide a JDBC driver for SQL Server. Obtain the appropriate JDBC driver and place it on the Web server's classpath. For example, if running Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
4. Open the `/tomcat/lib/ozone/framework/application.yml` file and modify the `environments.production.dataSource` section using the values that are appropriate for the OWF environment.

Example: application.yml

```
dataSource:
  pooled: true
  dbCreate: "none"
  username: "owf_user"
  password: "owf"
  driverClassName: "net.sourceforge.jtds.jdbc.Driver"
  url: "jdbc:jtds:sqlserver://localhost:1443/OWF"
  dialect: "ozone.owf.hibernate.OWFSQLServerDialect"
  properties:
    minEvictableIdleTimeMillis: 180000
    timeBetweenEvictionRunsMillis: 180000
    numTestsPerEvictionRun: 3
    testOnBorrow: true
    testWhileIdle: true
    testOnReturn: true
    validationQuery: "SELECT 1"
```

In the example above the SQL Server database user named `owf_user` with password of `owf` is used, to access a database named `OWF`.

5. Before starting OWF, run the following scripts to create the database schema:
`/dbscripts/sqlserver/SQLServerCreate.sql`

6. To load the initial sample data into the database:

- a. Open Microsoft SQL Server Management Studio (or another database editing tool) and select File → Open File
- b. Navigate to `/dbscripts/archive/SQLServerPrefsUpdate_v7.0.0.sql` in the bundle.



This script should ONLY be run against an empty database as it will delete pre-existing data.

- c. Select the OWF database, and execute the script.

Notes:

- Old versions of the SQL scripts are included under `/database/archive/`. These are included for convenience for people upgrading from older versions of OWF.

5. Security

5.1. Overview

OWF allows an administrator to customize the type of security that is implemented for authentication and authorization.

OWF uses a pluggable security solution based on Spring Security 4.x and ships with sample security plugins that can be used as a basis for building a custom security plugin.

Familiarity with Spring Security (<https://spring.io/projects/spring-security>) will help administrators customize OWF.

5.1.1. Basic Security Concepts and OWF

While this guide is not intended as a comprehensive guide to basic security concepts, Web security, or Spring Security, there are a few key concepts that must be understood in order to use the sample OWF security plugins and the OWF security plugin architecture.

First are the concepts of authentication and authorization, known colloquially as auth & auth. Authentication essentially means providing proof that the user is exactly who they are presenting themselves to be. Some authentication techniques include a username/password combination, an X509 certificate, a CAC card and card reader, or various biometric solutions. Authorization, on the other hand, is determining the specific access rights that an individual user should have. Consider the following:

- "Bill is allowed to log into the system – prove that you are Bill," is a matter of authentication.
- "Bill has access to resources," is a question of authorization.

By necessity, authentication occurs before authorization. Once authentication is satisfied, OWF moves to authorize. OWF has two authorization concepts at this time. First, OWF needs to know whether or not a user has OWF administrative access via `ROLE_ADMIN` or is only a regular user, via `ROLE_USER`. Administrative access provides a user access to the administrative widgets and the administrative console. Regular users have access only to the framework and their assigned dashboards.

Second, OWF needs to know what external OWF user groups (if any) the user has been assigned. There are two kinds of user groups; automatic user groups, which are pulled in from an external authorization source, such as LDAP or a configuration file, and manual user groups, which are set up from within OWF. If an automatic user group is new to OWF, all of the automatic user groups' details such as description, active/inactive status, contact email address, and name come from the external source. But after the initial creation of the group in OWF, no further updates to the description, status, etc. are made.

5.1.2. Production Deployments

The samples included with OWF are **NOT** production-quality samples. They are intended to provide examples

on how to easily integrate various security solutions with OWF, not to provide a comprehensive security solution out of the box or a comprehensive tutorial on Spring Security.

It is expected that each organization using OWF will examine its security guidelines and enterprise-wide authentication/authorization solutions and produce an OWF security plugin that is both secure and meets its standards. That solution can then be shared among OWF deployments within the organization.

5.2. Default Security Configuration

The OWF Bundle is configured to run by default on `localhost` with a predefined set of users.

When using the default Tomcat bundle, externalized configuration files should be placed in the `/tomcat/lib/ozone/framework/` directory. If using an application server other than Tomcat, copy the override files into the directory that will include them in the classpath for that specific application server.



The sample security modules are included as examples and should NOT be used in a production environment. For more information, please refer to the OWF Security section.

To add users to any security module using certificate authentication, generate a user PKI certificate that can be recognized by OWF.

5.2.1. Adding Users, Roles, & Groups

The addition of users, groups, and roles into OWF depends on the choice of security implementation. The following example outlines the procedures for adding users, groups, and roles when using the sample security configurations.

By default, authorization credentials are validated by the sample `MyDetailsService`. This example service is pre-configured to read the user information from the `/tomcat/lib/ozone/framework/users.properties` file.



The default `MyDetailsService` will not automatically create database entries for users. If a new user is added to the `users.properties` file, a corresponding user record must be manually added to the database.

1. Edit `/tomcat/lib/ozone/framework/users.properties`

```
testUser1=password,ROLE_USER,Test User 1,[group1;I am a sample Group 1 from
users.properties;test@gmail.com;active]
testUser2=password,ROLE_USER,Test User 2
testUser3=password,ROLE_USER,Test User 3
testAdmin1=password,ROLE_ADMIN,Test Admin 1,[group1;I am a sample Group 1
from users.properties;test@email.com;active],[group2;I am a sample Group 2
from users.properties;test2@email.com;active],[group3;I am a sample Group 3
from users.properties;test3@email.com;inactive]
```



To include space characters in the user names, escape the space characters using the `\` character.

2. Add users to the file in accordance with the following rules:

- Data Format:

```
username=password,role,display    name,[group    name;group    description;group
contact email;active/inactive]
```

- All of the information for a single user, including group information, should be on a single line.
- Multiple groups may be delimited by commas.
- Group information is optional and may be left out for any single user.
- Once a group has been created for the first time, the description, contact email, and active/inactive status will not effect those values within OWF – that information must be managed through the OWF Group Manager administrative widget.

3. Save the file and restart the OWF server.

Any changes (additions, deletions, or modifications) to the `users.properties` will only take effect once the OWF server has been restarted.



If a custom web server or application container is being used along with the provided example security, the `users.properties` file can be copied to any directory that is on the classpath of the Web server in use. For example, if using Jetty, the file can be copied to the `/jetty/resources/` directory.

5.3. Sample Security Modules



Most of the examples provided contain various obvious security hazards -- for example, the `users.properties` file contains a list of usernames, roles, and user groups on the hard drive in plain text in a properties file. **These are undeniable security hazards.** Keep this in mind when using the samples.

5.3.1. Installing Security Modules

The OZONE-security files offer multiple examples of security options. **These are intended as examples and should in no way be used in a production environment.**

For each available security option, there is a specific XML file which must be installed. Installing a new security module is accomplished in just a few simple steps:

1. Stop the application server.
An administrator can accomplish this by running the `ozone/tomcat/bin/shutdown.bat` or `/shutdown.sh` file, depending on the operating system in use.
2. Edit the `application.xml` file located in `ozone/tomcat/lib/ozone/framework/`
Modify the `<import resources="security/security.xml"/>` line to point to the desired security configuration file.
3. Restart the application server by running either the `ozone/tomcat/start.bat` or `/start.sh` file, depending on the operating system in use.

5.3.2. X.509 Certificate (PKI) Only

The `ozone/tomcat/lib/ozone/framework/examples/security_cert-only.xml` file eliminates the login form authentication. If the user does not present a valid X.509 certificate, they will be denied access to the system.

To use this security plugin:

1. Stop the application server.
2. Edit the `application.xml` file located in `ozone/tomcat/lib/ozone/framework/`
 - Modify the `<import resources="security/security.xml"/>` line to point to the certificate-only configuration.
Example: `<import resources="security/security_cert-only.xml"/>`
3. Edit the `server.xml` file located in `ozone/tomcat/conf/`
 - Set the HTTPS-enabled `Connector` element `clientAuth` property to `"true"`
4. Restart the application server by running either the `ozone/tomcat/start.bat` or `/start.sh` file, depending on the operating system in use.

Example: application.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans-4.3.xsd">

    <import resource="session-control.xml"/>
    <import resource="security/security-cert_only.xml"/>

</beans>
```

5.3.2.1. Installing User Certificates (PKI)

In order to use certificate-based authentication, both the server and clients must be configured with the appropriate certificates. Sample certificates are included in the bundle under the `tomcat/certs/` directory.

These default client certificates can be used by importing the included `testUser1.p12` or `testAdmin1.p12` certificate into the user's browser.

The `testUser1` certificate grants regular user permissions to use the application, while the `testAdmin1` certificate grants both regular user administrator permissions. The private key password for both certificates is `password`.

Please refer to the **Ozone Quick Start Guide** for detailed instructions on importing the user certificates into a web browser.

6. Configuration

OWF offers a variety of custom configuration options in the externalized `application.yml` configuration file, located in the bundle in the `/tomcat/lib/ozone/framework/` directory. Once changes are made, restart the system to apply the changes.

The default settings provided in the `application.yml` file are intended for use in a local, non-production environment. For production deployment to a non-local environment or to use an external database, this file must be configured with the appropriate settings, which are explained the following sections.



In previous versions of OWF, the `OwfConfig.xml` file housed many of the application's customizable values. Beginning with OWF 7.17.2, these values have been moved to the `application.yml` file described below.

6.1. Environments settings

The `environments` section of the `application.yml` file contains environment-specific settings which override the top-level configuration sections.

This section is optional; any sub-sections here may be copied to the top-level if no alternate environment configurations are desired. However, it may be used to enable quickly switching settings such as server hosting locations or databases for multiple environments such as testing, development, staging, or production.

The provided example `application.yml` uses these environment-specific blocks for some configuration settings. In the following guide, if the referenced section can not be found as a top-level section, check the `environments` section for the corresponding sub-section.

Example

```
environments:
  development:
    server:
      # Development `server` settings

  production:
    server:
      # Production `server` settings

  dataSource:
    # Production `dataSource` settings
```

6.2. Grails

The `grails` section of the `application.yml` file contains configuration specific to the Grails web framework.

Example: *application.yml*

```
grails:
  cache:
    ehcache:
      ehcacheXmlLocation: 'classpath:ozone/framework/ehcache.xml'
      lockTimeout: 200
  resources:
    cachePeriod: 600
  cors:
    enabled: true
```

grails.cache.ehcache.ehcacheXmlLocation

string — Location of the Ehcache XML configuration file.

grails.resources.cachePeriod

number — Enables caching of the static assets by setting the HTTP Cache-Control header's max-age property.

grails.cors.enabled

boolean — Enables Cross-Origin Resource Sharing (CORS) support to allow cross-domain AJAX requests to the application.

6.3. Security settings

The **security** section of the **application.yml** file contains additional configuration for Spring Security. Please see the Spring Security documentation for details.

Example

```
security:
  basic:
    enabled: false
```

security.basic.enabled

boolean — Enables basic HTTP username and password authentication. Disabled by default in the form-based login security example.

6.4. Hibernate settings

The **hibernate** section of the **application.yml** file contains the configuration for the Hibernate data access layer.

Example

```
hibernate:
  cache:
    queries: false
    use_second_level_cache: false
    use_query_cache: false
```

6.5. Quartz settings

The `quartz` section of the `application.yml` file contains the configuration for the Quartz job scheduler.

Example

```
quartz:
  autoStartup: false
  jdbcStore: false
```

6.6. Data Sources settings

The `dataSource` section of the `application.yml` file contains the connection configuration for the application database.

Please see [\[database-setup\]](#) for additional details.

Example

```
dataSource:
  dbCreate: none
  url: jdbc:postgresql://localhost:5432/owf
  driverClassName: org.postgresql.Driver
  username: owf
  password: owf

  properties:
    jmxEnabled: true
    initialSize: 5
    maxActive: 50
    minIdle: 5
    maxIdle: 25
    maxWait: 10000
    maxAge: 600000
    timeBetweenEvictionRunsMillis: 5000
    minEvictableIdleTimeMillis: 60000
    validationQuery: SELECT 1
    validationQueryTimeout: 3
    validationInterval: 15000
    testOnBorrow: true
    testWhileIdle: true
    testOnReturn: false
    jdbcInterceptors: ConnectionState
    defaultTransactionIsolation: 2 # TRANSACTION_READ_COMMITTED
```

6.7. Server settings

The **server** section of the **application.yml** file contains the configuration for the web server, such as port, context path, and SSL settings.

In the provided example **application.yml** file, the **server** section may be found under the top-level **environments** section.



Some of these settings may be handled by the host web server or application container. Please see the web server documentation for details.

Example

```
server:
  port: 8443
  contextPath: "/owf"
  ssl:
    enabled: true
    protocol: "TLS"
    client-auth: "want"
    key-store: "./certs/keystore.jks"
    key-store-password: "changeit"
    trust-store: "./certs/keystore.jks"
    trust-store-password: "changeit"
```

6.8. OWF settings

The **owf** section of the **application.yml** file contains the settings specific to the OWF application.

6.8.1. Help settings

When a user clicks the question mark button in the toolbar, OWF offers online help:

Out of the bundle, the Help window contains:

- Instructions for Configuring Help
- Keyboard Navigation Shortcuts

6.8.1.1. Changing the location of help files

The help directory location is defined by the **owf.external.helpPath** property in **application.yml**.

By default, help files are located on the classpath in the **/ozone/framework/help/** directory. In the default Tomcat bundle, this may be found at **/tomcat/lib/ozone/framework/help/**.

To change the directory location, replace **classpath:ozone/framework/help** with a location in one of the following supported formats, then restart the server:

Formats:

- **classpath:ozone/framework/help** — An absolute path to a directory on the file system.
- **file:/ozone/framework/help** — A relative path to a directory in the classpath root (for example, **/tomcat/lib/**)
- **ozone/framework/help** — A path of a resource directory within the OWF WAR file

Example

```
owf:  
  external:  
    helpPath: "classpath:ozon/framework/help"
```

6.8.1.2. Custom Help file types

Only files in the **helpPath** directory with specific file extensions will appear in the user interface. By default, files with the following file extensions will appear: **HTM**, **HTML**, **GSP**, **JSP**, **PDF**, **DOC**, **DOCX**, **MOV**, **MP4**, and **WMV**.

To modify the file types that will appear in the Help window, change the **owf.helpFileRegex** property (in regular expression format) in the **application.yml** file. The administrator must restart the server after the value is updated.

Example

```
owf:  
  helpFileRegex: '^.*\.(htm|html|gsp|jsp|pdf|doc|docx|mov|mp4|wmv)$'
```

6.8.2. Notification settings



The Notification feature is no longer maintained. The steps below are for reference purposes only.

The OWF Notifications system connects an OWF and Store to an XMPP server through which it can receive information about Widgets. This feature was originally designed to be used by administrators to monitor the change logs of Store listings, but may be used to allow any Widgets to communicate with OWF users. Widgets that connect to this XMPP server can use it to publish notification messages for consumption by OWF users. Messages published to the chat room must follow a specific format that designates both the message content and the OWF usernames of the intended recipient users. The OWF server will then receive the messages from the chat room and distribute them to individual users.

To configure OWF to use notifications, follow these steps:

- Create a chat room on an XMPP server that serves as a repository for alerts to/from OWF and the Store, respectively.
- Create user accounts on the XMPP server for the Store and OWF servers.
- Use the XMPP server information to populate the XMPP Settings in the **notifications** section of **application.yml** defined below:



Use the user accounts and their passwords from step two as the Username and Password fields on the Notifications configurations in OWF and the Store.

Example: Notification settings

```
notifications:
  enabled: false
  query.interval: 30

xmpp:
  username: ''
  password: ''
  host: ''
  room: ''
  port: 5222
```

notifications.enabled

boolean — Turn on or turn off notifications.

notifications.query.interval

number — How often OWF fetches new notifications.

notifications.xmpp.username

string — The domain name for the XMPP server that administers notifications.

notifications.xmpp.password

string — The domain password for the XMPP server that administers notifications.

notifications.xmpp.host

string — The hostname of the XMPP server.

notifications.xmpp.room

string — The XMPP chat room that receives notifications.

notifications.xmpp.port

number — The TCP port the XMPP server uses.

6.8.3. Custom Data Guard and Widget Security



The Data Guard feature is no longer maintained. The steps below are for reference purposes only.

OWF 7.1 enhanced widget security with a data guard system that addresses widget security in the following ways:

- Restricts messages between widgets based on their access levels. Disabled by default; to activate the feature, set **restrictMessages** to **true**.

- Audits all messages between widgets. To enable this feature, ensure that `restrictMessages` is set to `true`, then set the `auditAllMessages` to `true`.
- Restricts widgets ability to send messages until their access level is specified. To enable this feature, ensure that `restrictMessages` is set to `true`, then change `allowMessagesWithoutAccessLevel` to `false`.
- Specifies the amount of time the system allows to cache a widget's access level. To modify the duration, ensure that `restrictMessages` is set to `true` and quantify the `accessLevelCacheTimeout` in milliseconds.

To change this format, update the respective properties in `application.yml`:

Example: Data Guard settings

```
dataguard:
  # Option to restrict messages between widgets based on access levels.
  # If this option is set to false, all other dataguard options are ignored.
  restrictMessages: false

  # Option to audit all messages between widgets, not just failed messages.
  # restrictMessages must be set to true
  auditAllMessages: false

  # Option to allow widgets to send messages without specifying their access
  level.
  # restrictMessages must be set to true
  allowMessagesWithoutAccessLevel: true

  # The amount of time (in milliseconds) to cache a widget's access level.
  # restrictMessages must be set to true
  accessLevelCacheTimeout: 3600000
```

6.8.4. Store Synchronization settings

The synchronization feature allows the Store to automatically send Widgets and their subsequent updates to OWF. Synchronization is also necessary to push OWF applications to the Store. To use this feature, developers must do the following:

- Configure `application.yml` to accept synchronization messages from the Store.
- Configure `security.xml` for both OWF and the Store. (see below)
- Synchronize the OWF server with the Store through the Store's Administration Configuration pages.

To implement the synchronization feature with the OWF sample security plugin, configure the following `security.xml` files for OWF and the Store:

OWF – `/tomcat/lib/ozone/framework/security/security.xml`:

1. Add the following to the top-level <beans> element:

```
<sec:http pattern="/marketplace/sync/" security="none" />
```

Store – /tomcat/lib/ozone/marketplace/security/security.xml:

1. Add the following to the top-level <beans> element:

```
<sec:http pattern="/public/descriptor/" security="none" />
```

The Store will only synchronize with OWF servers linked to that Store. The Store will not synchronize with unspecified OWF systems or offline OWF systems. Synchronize OWF servers with the Store through the Store Administration Configuration pages. For more information, see the **OZONE Store Administrator's Guide**.

6.8.4.1. Automatically add the Store listings and updates

The Marketplace Synchronization feature allows OWF administrators to automatically receive Widgets and their updates from the Store.

The synchronization feature is disabled by default. When enabled, OWF compatible listings are directly added from the Store to the Widget Manager in OWF.

To enable this feature, edit the following property values in application.yml as indicated in the table.

mpSync.autoCreateWidget

boolean — Whether new Store listings are automatically added to OWF.

mpSync.enabled

boolean — Whether OWF will process listing requests from the Store. Set this to **true** to receive listings and updates from the Store.



Updates from the Store will overwrite any changes made to the corresponding widgets in OWF.

When **mpSync.autoCreateWidget** is **true**, OWF automatically adds Widgets to the Widget Manager that have the following listing criteria in the Store:

- Type is **App Component**
- Listing status is **Approved**
- Listing is **Enabled**
- A State that has "Is Published" set to **true**

Initially, the added Widgets are not associated with users or groups. However, users or administrators can see and add them from the Store. Administrators will not automatically receive all OWF compatible Widgets when they sign into OWF. Only new and updated Store listings that meet the criteria described above will automatically appear in the Widget Manager after enabling the this feature.

6.8.4.2. Enabling Auto-Updates from the Store

Using synchronization, users and administrators can automatically receive updates for Widgets previously added to OWF through the Store. Store updates will overwrite changes made to the corresponding Widget in OWF. For example, if an administrator adds an Intent to the Widget, the update from the Store will remove this data. To allow Store synchronization in OWF, set the `mpSync.enabled` property to `true` in the `application.yml` file.

To disable the synchronization between the Store listings and their OWF counterpart Widgets, set the `mpSync.enabled` property to `false`; by default, this property is set to `false`. Alternatively, disable the listing in the Store or remove the OWF connection from the Store.

If a developer chooses not to enable the automatic Widget synchronization, updating a Widget will require the user to delete their existing version of the widget and add the updated Widget from the Store Widget in their OWF.

6.8.4.3. Widget Load Times

Because the speed with which Widgets launch and operate is related to an individual system's resources, OWF has implemented ways to track the time between a Widget's rendering and the time it is actually ready to be used. The following values can be active at the same time:

Example: Widget load times settings

```
owf:  
  sendWidgetLoadTimeToServer: true  
  publishWidgetLoadTimes: true
```

owf.sendWidgetLoadTimeToServer

boolean — Enable or disable sending Widget load time data to a system log file where it is written and stored.

owf.publishWidgetLoadTimes

boolean — Enable or disable sending the data to the Widget Log Widget, which may be opened after it is assigned to a user's instance of OWF.

6.8.5. Custom Access Alert settings

Depending on the individual security requirements where OWF is being deployed, users may be required to agree to the specific terms of a security warning. Deploying a security warning is accomplished via a custom access alert.

The default OWF bundle configuration contains example consent notice and user agreements with placeholder text, which may be customized in the `owf` section of the `application.yml` file.

```
owf:
  consent:
    enabled: true
    title: "DoD Privacy and Consent Notice"
    nextUrl: "/login"
    details:
      enabled: true
      linkText: "See User Agreement for details."
    message: |-
      You are accessing a U.S. Government (USG) Information System (IS)
that is provided for USG-authorized use
      only. By using this IS (which includes any device attached to this
IS), you consent to the following conditions:

      * The USG routinely intercepts and monitors communications on this
IS for purposes including, but not limited to,
      penetration testing, COMSEC monitoring, network operations and
defense, personnel misconduct (PM), law enforcement
      (LE), and counterintelligence (CI) investigations.
      ...
```

owf.consent.enabled

boolean — Enable or disable the display of a custom consent or warning dialog to the user before they are allowed to access OWF.

owf.consent.title

string — The title text to display on the custom consent or warning banner dialog.

owf.consent.nextUrl

string — The URL to redirect the user to when they click the Accept button on the dialog.

owf.consent.details.enabled

boolean — Whether a link should be displayed to another dialog for additional details (for example, a user agreement).

owf.consent.details.linkText

string — The text to display as the link to the details dialog.

owf.consent.message

string — The full message to display in the consent or warning dialog. May be formatted using standard Markdown syntax, and the indentation on the left will be removed.

Example: Access Alert details (User Agreement) settings

```
owf:
  agreement:
    title: "User Agreement"
    message: |-
      __STANDARD MANDATORY NOTICE AND CONSENT PROVISION__

      By signing this document, you acknowledge and consent that when
you access Department of Defense (DoD) information
      systems:

      You are accessing a U.S. Government (USG) information system (IS)
      (which includes any device attached to this
      information system) that is provided for U.S. Government
authorized use only. You consent to the following
      conditions:

      * The U.S. Government routinely intercepts and monitors
communications on this information system for purposes
      ...
```

owf.agreement.title

string — The title text to display on the additional details (e.g. user agreement) dialog.

owf.agreement.message

string — The full message to display in the additional details. May be formatted using standard Markdown syntax, and the indentation on the left will be removed.

6.8.6. Logout settings

OWF allows users to sign out of OWF in accordance with their instance's security plugins. Any modifications made to the URL will take effect when the system is restarted.

Example: Logout settings

```
owf:
  logout:
    enabled: true
    url: "/login?out=1"
```

owf.logout.enabled

boolean — Enable or disable logging out of OWF using the User menu. (Value is case sensitive; must be true or false)

owf.logout.url

string — The URL to redirect the user to when they log out.



The security plugin being used must be customized so that the value entered above will work as expected.

6.8.7. Auto Save settings

The OWF user interface will automatically save the user's dashboard at a configurable time interval. The default is to save every 15 minutes. Any modifications to the default save interval will take effect when the `application.yml` is saved and the system is restarted.

Example: Auto Save settings

```
owf:  
  autoSaveInterval: 900000 # 15 minutes
```

owf.autoSaveInterval

number — The interval for saving the user's dashboard, in milliseconds.



The Auto Save feature will keep the user session alive as long as a dashboard is visible in the browser.

7. Logging

7.1. Logging Configuration

Logging can be configured by editing the `logback.groovy` file which can be found in the `/tomcat/webapps/owf/WEB-INF/classes/` directory.

```
logback.groovy
import java.nio.charset.Charset

// Configuration variables
def LOG_PATH = "./logs"

// Status listener to display changes to the logging configuration
// statusListener(OnConsoleStatusListener)

// Enable periodically scanning the logging configuration for changes
scan("30 seconds")

// Console appender
appender("CONSOLE", ConsoleAppender) {
    encoder(PatternLayoutEncoder) {
        charset = Charset.forName("UTF-8")
        pattern = "%d{yyyy-MM-dd HH:mm:ss.SSS} %level [%logger] %msg%n"
    }
}

// File appender
appender("FILE", RollingFileAppender) {
    append = true
    encoder(PatternLayoutEncoder) {
        charset = Charset.forName("UTF-8")
        pattern = "%d{yyyy-MM-dd HH:mm:ss.SSS} %level [%logger] %msg%n"
    }
    rollingPolicy(TimeBasedRollingPolicy) {
        FileNamePattern = "${LOG_PATH}/ozone-framework_%d.log"
    }
}

// Default root logging level
root(ERROR, ["CONSOLE", "FILE"])

// Logging for the Ozone namespaces -- set to INFO or DEBUG for more verbose
logging
logger("ozone", WARN)
logger("org.ozoneplatform", WARN)

// Logging for all AJAX requests
logger("ozone.owf.grails.controllers.RequestLogInterceptor", INFO)
```

Logback supports configuration using a Groovy Domain Specific Language (DSL) as demonstrated above, as well as traditional XML configuration file.

To use an XML file, replace the logback.groovy file in the above path with a file named `logback.xml`.

The configuration options and syntax for the Logback configuration are documented in the Logback manual, listed below.

References:

- Logback: Manual – <https://logback.qos.ch/manual/index.html>
- Logback: Groovy Configuration (DSL) – <https://logback.qos.ch/manual/groovy.html>
- Grails v3.3.x: Logging Guide – <https://docs.grails.org/3.3.2/guide/conf.html#logging>

7.2. Audit Logging

OWF includes an option to audit all user entry and exit in the system. The OWF Bundle ships with this feature enabled by default. The Audit Log tracks the following types of changes:

- Both successful and unsuccessful sign-in attempts
- User Sign-out Events:
 - A user signing out on purpose
 - A session times out



References to CAS and OWF must match the settings of the current installation.

7.2.1. Sign-in Events

A sign-in failure will occur and be recorded in the log if a user has a valid PKI Certificate but the associated username is not registered as a valid user within OWF. A failed sign in produces the following log statement at the info level:

Failed sign-in event (INFO)

```
INFO [02/15/2011 15:24:04 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]:  
LOGIN FAILURE - ACCESS DENIED with FAILURE MSG [Login for 'testAdmin1'  
attempted with authenticated credentials [CERTIFICATE LOGIN]; However, the  
Provider was not found. Access is DENIED.]
```

A failed sign in produces the following log statement at the debug level:

Failed sign-in event (DEBUG)

```
DEBUG [02/15/2011 15:27:18 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]:LOGIN FAILURE - ACCESS DENIED with FAILURE MSG [Login for 'testAdmin1' attempted with authenticated credentials [CERTIFICATE LOGIN >> Signature Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5]; Subject: [EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; Validity: [From: Thu Feb 04 13:58:52 EST 2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer: [EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; ]; However, the Provider was not found. Access is DENIED. Login Exception Message: [No AuthenticationProvider found for org.springframework.security.web.authentication.preauth.PreAuthenticatedAuthenticationToken]]
```

A successful PKI Certificate sign in produces the following log statement at the info level:

Successful certificate sign-in event (INFO)

```
INFO [02/15/2011 15:39:13 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]: LOGIN SUCCESS - ACCESS GRANTED USER [testAdmin1], with DISPLAY NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with ORGANIZATION [Test Admin Organization], with EMAIL [testAdmin1@nowhere.com]with CREDENTIALS [CERTIFICATE LOGIN]
```

A successful PKI Certificate sign-in statement produces the following log statement at the debug level:

Successful certificate sign-in event (DEBUG)

```
DEBUG [02/15/2011 15:42:10 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]:LOGIN SUCCESS - ACCESS GRANTED USER [testAdmin1], with DISPLAY NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with ORGANIZATION [Test Admin Organization], with EMAIL [testAdmin1@nowhere.com] with CREDENTIALS [CERTIFICATE LOGIN >> Signature Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5]; Subject: [EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; Validity: [From: Thu Feb 04 13:58:52 EST 2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer: [EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; ]
```

7.2.2. Logout Events

Sign-out events are logged by the `ozone.securitysample.authentication.audit.SecurityAuditLogger` logger. This logger supports two levels of logging: **info** and **debug**, with the latter providing more detailed information about each sign-out event.

Below is a typical user-initiated sign-out event which has been saved as a log entry, with the log level set to info:

Sign-out event (INFO)

```
INFO [02/03/2011 16:13:35 -0500] IP: 127.0.0.1 SessionID: 8ki2ttimdx User:
testAdmin1 [USER LOGOUT]:
```

Below is a typical user-initiated sign-out event which has been saved as a log entry, with the log level set to debug:

Sign-out event (DEBUG)

```
DEBUG [02/03/2011 15:59:53 -0500] IP: 127.0.0.1 SessionID: 1tjefhsxz1x6t User:
testUser1 [USER SESSION TIMEOUT] with ID [2], with EMAIL
[testUser1@nowhere.com], with ACCOUNT CREATED DATE [02/03/2011 15:58:50
-0500], with LAST LOGIN DATE [02/03/2011 15:58:50 -0500]
```

A user can also be forced to sign-out when their session times out. Below are info and debug log statements:

Session time-out event (INFO)

```
INFO [02/07/2011 10:08:21 -0500] IP: 127.0.0.1 SessionID: 1b4nvaqnb0qx8 User:
testAdmin1 [USER SESSION TIMEOUT]
```

Session time-out event (DEBUG)

```
DEBUG [02/07/2011 10:24:21 -0500] IP: 127.0.0.1 SessionID: d0pq3g4xguv3 User:
testAdmin1 [USER SESSION TIMEOUT] with ID [1], with EMAIL
[testAdmin1@nowhere.com], with ACCOUNT CREATED DATE [02/07/2011 10:23:18
-0500], with LAST LOGIN DATE [02/07/2011 10:23:18 -0500]
```

7.3. Common Event Format (CEF) Auditing

Common Event Format (CEF) auditing capabilities are available in OWF and the Store. The variables used for OWF CEF logging are defined in the application.yml. To enable/disable them, sign into OWF and the Store as an administrator and navigate to the auditing configurations. CEF auditing is turned ON by default, the toggle controls for both CEF and Object Access auditing are found in OWF's Application Configurations which is located on the drop-down User Menu in the user interface. For more information, see the OWF Administrator's Guide.

When enabled, CEF auditing records common user events:

- Sign in and out (Sign out - Marketplace only)
- Create, Read¹, Edit and Delete
- Search
- Import and Export

¹. Object Access auditing is a separate CEF auditing feature that records users' Read events. Read events are logged when both the CEF auditing global flag and the Object Access flag are ON in OWF's Application Configurations. If the Object Access auditing is ON and the CEF auditing is OFF, no Read events are logged with CEF auditing.

CEF auditing and Audit Logging (see [Section 7.2, "Audit Logging"](#)) record overlapping database events (ex. edit, delete and log out events). When CEF auditing is ON, CEF auditing is the recorder of these database events; these events are not recorded with Audit Logging. When CEF is turned OFF, Audit Logging records these database events.

The following are two log examples using CEF auditing:

CEF auditing from an object modification event

```
26 Jun 2013 12:31:37,217 EDT CEF:0|ORG|MARKETPLACE|500-  
27_L2::IC::1.3|FILEOBJ_MODIFY|Object was updated|7|cat=FILEOBJ_MODIFY  
suid=MikePAdmin shost=10.10.16.12 requestMethod=USER_INITIATED outcome=SUCCESS  
deviceFacility=0CCB5827C819E1A4AC9BE5BD4C6F9FE9.mp02 reason=UNKNOWN  
cs5=UNKNOWN act=7.2 deviceExternalId=UNKNOWN dhost=amlqa02.goss.owfgoss.org  
cs4=UNKNOWN start=06:26:2013 12:31:37 [.037]cs3=UNKNOWN  
fname=[CLASS:marketplace.OwfProperties, stackContext:, stackDescriptor:]  
filePermission=UNKNOWN fileId=16 fsize=2 fileType=OBJECT  
oldFilename=[CLASS:marketplace.OwfProperties, stackContext:null,  
stackDescriptor:null]oldFilePermission=UNKNOWN oldFileId=16 oldFileSizesize=2  
oldFileType=OBJECT
```

CEF auditing from a log-on event

```
26 Jun 2013 08:57:51,974 EDT CEF:0|ORG| MARKETPLACE L|500-  
27_L2::IC::1.3|LOGON|A logon event occurred.|7|cat=LOGON suid=MikePAdmin  
shost=10.10.16.12 requestMethod=USER_INITIATED outcome=SUCCESS  
deviceFacility=8C24A08B7E9848C80F929791DA40F734.mp02 reason=UNKNOWN  
cs5=UNKNOWN act=7.2 deviceExternalId=UNKNOWN dhost=amlqa02.goss.owfgoss.org  
cs4=UNKNOWN start=06:26:2013 08:57:51 [.051]cs3=UNKNOWN
```

8. Upgrading

8.1. Upgrading from version 7.17.1.0

1. Before starting the upgrade, backup the corresponding database previously used with OWF 7.17.1.0 as well as all custom override configuration files (In Tomcat, they are normally located in the `/tomcat/lib/` directory)

For this upgrade, you can use your database used in OWF 7.17.1.0 with no alterations necessary.

Please see [Chapter 6, Configuration](#) for any questions regarding configuration files.

2. Install OWF

```
cp ozone-framework-8.0.0.0.zip /opt
cd /opt
unzip ozone-framework-8.0.0.0.zip
cd tomcat
```

3. Make any configuration changes needed in `application.yml` configuration file, located in the bundle in the `/tomcat/lib/ozone/framework/` directory.
4. Completing the upgrade
 - a. Reconfigure and re-apply customizations in the `/tomcat/lib/` directory
 - b. If a custom keystore was deployed in a previous build, the keystore for OWF will need to be manually configured in the same manner.
 - c. Copy any custom certificates from previous build to OWF v8.0.0.0. Usually this is found in `/tomcat/certs/`.
 - d. Upgrade is now complete

8.2. Connecting to OMP (Marketplace)

In order to connect your OMP Marketplace (v7.17.X.X) with OWF v8.0.0.0, you must modify the following files:

- `application.properties`
 - Make sure the following versions match the version numbers of the respective items in owf-framework-server.

```
app.base.version=8.0.0.0
tomcat.custom.rev=1.2.3-0
```

- **build.gradle**

- Make sure the following versions match the version numbers of the respective items in owf-framework-server.

```
mavenBom 'org.ozoneplatform:ozone-classic-bom:7.17.2-0'
customTomcat(group: 'org.ozoneplatform', name: 'owf-custom-tomcat',
version: '1.2.3-0')
```

- **grails-app/conf/application.yml**

- Add URLs of all Ozone instances that will be using this store to the **allowedOrigins**, e.g.:

```
cors:
  allowedOrigins:
    - http://localhost:3000
    - https://192.168.1.2:8000

# Set the port for OMP to run on.
production:
  server:
    port: 8444
```

- **src/main/resources/ozone/marketplace/security.xml**

- Add URLs of all Ozone instances that will be using this store to the **allowedOrigins**, e.g.:

```
<sec:header name="Content-Security-Policy" value="frame-ancestors
    http://localhost:3000/
    https://192.168.1.2:3000/
"/>
```

- Restart your application
- See Administration Guide on how to add stores to application

9. Deployment

9.1. Operating OWF From Different/Multiple Ports

Initial OWF configuration is set up so that Tomcat can be run from a local installation. Throughout this document, `servername:port` implies a `localhost:8080` or `localhost:8443` location. The example below shows how to set up OWF so that it can be used on 5050/5443.

To enable ports other than 8080/8443, the desired ports need to be explicitly edited in the Web server configuration file: `/tomcat/conf/server.xml`.



In the event that OWF is running on a server where a port number is already in use, OWF must run from a different port number. Two applications cannot bind to the same port.

1. For example, in Tomcat, change the port numbers in `/tomcat/conf/server.xml`, as shown below to the following port number:

Example: server.xml

```
<Connector port="5050"
           protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="5443" />

<Connector port="5443"
           protocol="HTTP/1.1"
           SSLEnabled="true"
           maxThreads="150"
           scheme="https"
           secure="true"
           keystoreFile="certs/keystore.jks"
           keystorePass="changeit"
           clientAuth="false"
           sslProtocol="TLS" />
```

- a. Ports 5050 and 5443 are just examples and can be changed to whatever is needed. If OWF was running on a server where a port number was already in use, the shutdown port must also be changed. To do this, change the port number in the Tomcat Web server configuration file `/tomcat/conf/server.xml` to another port, in the following example the default shutdown port was changed from 8005 to 8006:

Example: server.xml

```
<Server port="8006" shutdown="SHUTDOWN">
```

2. Restart the OWF server.

9.2. Adding the Store To OWF

The flexible and scalable nature of OWF allow for applications used in concert (such as the Store) to be included in Ozone's deployment. This allows a user to develop with the products working together, without having to activate multiple ports via configuration.

To include the Store in the Ozone bundle, do the following:

1. Unpack the zipped bundles containing the applications to be included.
2. Navigate to `/tomcat/webapps/` in each unpacked bundle.
3. Copy the appropriate WAR files into the `/tomcat/webapps/` directory where Ozone was deployed.
4. Copy the additional applications' configuration files into `/tomcat/lib/` where Ozone was deployed. For the Store, the configuration file is `application.yml`.
5. Restart the Ozone server.

9.3. Server Certificate Creation and Installation

Valid server certificates are needed for configuring the server to allow HTTPS authentication.



Self-signed certificates will produce warnings in a user's browser. This is because a self-signed certificate, not signed by a recognized certificate authority, has no one authorizing its validity. In a production environment, certificates should be signed by a recognized certificate authority, such as an organization's internal certificate authority.

9.3.1. Generating a New Self-Signed Server Certificate

A new self-signed certificate can be generated by navigating to the tools directory and executing `create-certificates.bat` or `.sh`, depending on the operating system in use.

Follow the on-screen prompts and create the necessary certificates for the installation.

Make sure to enter the FULLY QUALIFIED server name. This needs to match the domain name of the machine exactly or the certificate will not work correctly.

If using an IP address as the Common Name (CN), an entry must be added to the Subject Alternative Name entry in the certificate. The better alternative to using an IP address is to add a name/IP pair to the hosts file and register the name as the CN.

9.3.2. Configuring Ozone For a Different Truststore/Keystore

1. For server-to-server calls (Ozone-to-CAS communications, for example) the newly created self-signed certificate should be imported into the truststore. If the truststore is a separate file from the keystore, the certificate can be copied from the keystore to the truststore as follows:

- a. Export the certificate from the keystore into a file:

```
keytool -export -file servername.crt -keystore servername.jks -alias servername
```

- b. Import the file into the Truststore:

```
keytool -import -alias servername -keystore mytruststore.jks -file servername.crt
```

2. Modify the JVM Parameters that are used to start the web application server in order to use the new truststore shown above. If a Tomcat server is being used, the parameters can be found in the setenv.bat (or .sh, depending on the operating system in use) script found within the /tomcat/bin/ directory inside of the unpacked owf-bundle-8.0.0.0.zip. If an application server other than Tomcat is being used, the parameters will need to be added to the JVM parameters which are loaded when the application server is started.

Table 4. Custom JVM Parameters

Parameter	Default Value	Note
<code>javax.net.ssl.keyStore</code>	<code>"%CATALINA_HOME%/certs/keystore.jks"</code>	Replace <code>certs/keystore.jks</code> with the path and filename to the keystore.
<code>javax.net.ssl.trustStore</code>	<code>"%CATALINA_HOME%/certs/keystore.jks"</code>	Replace <code>certs/keystore.jks</code> with the path of the truststore (may be the same as the keystore).
<code>javax.net.ssl.keyStorePassword</code>	<code>changeit</code>	Replace <code>changeit</code> with the keystore's password (if applicable)
<code>javax.net.ssl.trustStorePassword</code>	<code>changeit</code>	Replace <code>changeit</code> with the truststore's password.

3. Finally, the server configuration must be modified to use the new KeyStore/Truststore in SSL. Below is the relevant section from the Tomcat configuration script found in `/tomcat/conf/server.xml`:

Example: server.xml

```
<Connector port="8443"
           protocol="HTTP/1.1"
           SSLEnabled="true"
           maxThreads="150"
           scheme="https"
           secure="true"
           keystoreFile="certs/keystore.jks"
           keystorePass="changeit"
           truststoreFile="certs/truststore.jks"
           truststorePass="changeit"
           clientAuth="want"
           sslProtocol="TLS" />
```


Appendix A: Custom Security Modules

A.1. Requirements for Customizing Security

The Spring Security Framework allows individual deployments to customize the OWF authentication and authorization mechanism. Developers can use the security plugin to integrate with any available enterprise security solutions. When customizing the security plugin, it is important to remember OWF/Store requirements for the plugin. These five requirements are described in this section.



The requirements for OWF and the Store are in addition to any general Web application requirements relating to Spring Security.

1. User principal implements the `UserDetails` interface and (optionally) the `OWFUserDetails` interface.

Like all Spring Security Web applications, OWF expects its security plugin to provide a `UserDetails` object which represents the logged-in user. A custom plugin should set this object as the principal on the `Authentication` object stored within the active `SecurityContext`. Optionally, the provided object may implement the `OWFUserDetails` interface. In addition to the fields supported by the `UserDetails` interface, the `OWFUserDetails` interface supports access to the user's OWF display name, organization and email. The source code for `OWFUserDetails` can be found in the `owf-security` project repository.

2. `ROLE_USER` granted to all users.

The user principal object's `getAuthorities()` method must return a collection that includes the `ROLE_USER GrantedAuthority`.

3. `ROLE_ADMIN` granted to OWF administrators.

The user principal object's `getAuthorities()` method must return a collection that includes the `ROLE_ADMIN GrantedAuthority` if the user is to have administrative access.

4. `OZONELOGIN` cookie set when the user signs in and deleted on sign out.

The user interface performs a check for the existence of a cookie named `OZONELOGIN` during the page load. If the cookie does not exist, the interface will not load, but will instead present a message indicating that the user is not signed in. It is up to the security plugin to create this cookie when the user signs in, and to delete it when they sign out. This mechanism prevents users from signing out, and then pressing the browser's Back button to get back into an OWF instance that cannot communicate with the server due to failed authentication. The sample security plug-in configurations contain filters that manage this process. It is recommended that custom configurations include this default implementation of the cookie behavior by using the same `ozoneCookieFilter` beans that are included in the sample configuration.

5. Session management configurations must be present.

These configurations include the `concurrencyFilter` bean, the `concurrentSessionControlStrategy` bean, the `sessionRegistry` bean as well as a `<session-management>` element and a `<custom-filter>` element which references the `concurrencyFilter`. It is important not to change the id of the `concurrentSessionControlStrategy`, as it is referenced by id from within the application.



The `maximumSessions` setting contained in the XML configuration will be overwritten at runtime, since the maximum number of sessions is configured in the Application Configuration UI.

A.2. Custom Security Classes

A.2.1. OWFUserDetails

Package: `ozone.security.authentication`

This interface defines interactions for a data model that OWF requires in order to handle OWF user groups. Using an implementation of this interface (and implementations may vary) will ensure that OWF user groups work.

This interface extends the `UserDetails` interface as defined by Spring Security.

Please refer to the Spring Security API documentation for more details regarding the `UserDetails` interface: <https://docs.spring.io/autorepo/docs/spring-security/4.0.0.RELEASE/apidocs/org/springframework/security/core/userdetails/UserDetails.html>.

OWFUserDetails.java

```
package ozone.security.authentication;

public interface OWFUserDetails extends UserDetails {

    public Collection<OwfGroup> getOwfGroups();

    public String getDisplayName();

    public String getOrganization();

    public String getEmail();

}
```

A.2.2. OWFUserDetailsImpl

Package: *ozone.security.authentication*

This class is a sample implementation of the **OWFUserDetails** interface. It is not mandatory to use this implementation.



If a custom implementation is written, authorities must be write-accessible; create a **setAuthorities** method.

A.2.3. OWFGroup

Package: *ozone.security.authorization.target*

This interface describes a single OWF user group. A group is a way of collecting OWF users and being able to assign widgets and other behaviors to them collectively:

OwfGroup.java

```
package ozone.security.authorization.target;

public interface OwfGroup {

    public String getOwfGroupName();

    public String getOwfGroupDescription();

    public String getOwfGroupEmail();

    public boolean isActive();

}
```

A.2.4. OwfGroupImpl

Package: *ozone.security.authorization.model*

This class implements the **OwfGroup** interface. It can be used as-is in a security implementation, or one can be created as needed.

A.2.5. GrantedAuthorityImpl

Package: *ozone.security.authorization.model*

This class implements the Spring Security interface **GrantedAuthority**. It can be used as-is in a security

implementation, or one can be created as needed.

A.2.6. MyDetailsService

Package: *ozone.securitysample.authentication.basic*

This class is a sample implementation of the Spring Security **UserDetailsService**. It reads user authentication and authorization details (username, password, roles, and groups) from the **users.properties** file.



This is intended only to be used in testing and development environments. The usernames and passwords are stored in plain-text, creating a severe security risk for any production environment.

For a production deployment, a custom implementation of the UserDetailsService should be used. For example, the user's details may be retrieved from an external service such as LDAP.

Glossary

Accordion (layout)

Display widgets in equal, horizontal panes that do not scroll (each individual widget may scroll using its own scroll bar).

Affiliated Store

A store that another organization uses for their system. When a local store is connected to an affiliated store, users in the local store can search for and add listings from the affiliated store (assuming the user has proper authentication for the affiliated store).

App

Deprecated term for a Stack.

App Component

Deprecated term for a widget.

Dashboard

An organized collection of widgets with a customizable layout.

Filters

A feature used to reduce the number of search results by type or category.

Fit (layout)

Allows a user to place a single widget on the screen.

Help

Repository of instructional guides and video tutorials.

Intent

Instructions for carrying out a widget's intentions.

Listing

Any software dashboard or widget that a user enters into the Store is called a "Listing." Listings can be a various types of Web content.

Marketplace

A searchable catalog of shared listings of widgets and dashboards (also referred to as the Store).

OWF

Abbreviation for Ozone Widget Framework.

Pages

Deprecated term for a dashboard.

Portal (layout)

A column-oriented layout that organizes widgets of varying heights. Each new widget loads above the first one on the screen. The user drags a dividing bar to specify widget's height. The widgets and the Ozone window scroll.

Required Listings

An association between Listings. *Example: if Listing A needs Listing B to function, Listing B is a Required Listing.*

Stack

A collection of Dashboards (pages). Allows administrators and users to group Dashboards into folder-like collections that allow for easy transition from one to another.

Store

Commonly used term for the Ozone Marketplace.

Tabbed (layout)

Display one widget per screen, with tabs the top of the screen to switch from one widget to another.

Toolbar

The navigation bar at the top of the application. It links to a user's stacks, widgets, the Store, online Help and options from the drop-down User Menu.

User

A person signed into the Ozone application, usually referring to a person without administrative privileges.

Widget

A light-weight, single-purpose Web application that offers a summary or limited view of a larger Web application and may be configured by the user and displayed within a Dashboard.

Widget Menu

The Widgets Menu displays all available widgets. Use this feature to start or add widgets to a dashboard.