Ozone Configuration Guide

DOD GOSS

Version 8.0.0.1-RC1, 2019-12-31

Table of Contents

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 editing configuration files and has some understanding of Python and Django.

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; .env file glossary and related information; 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 Waitress and requires Python 3.7.4 or higher. The bundle does not include a reverse proxy and will require one if a secure connection is desired. More information of setting up a reverse proxy to work with OWF will be provided in the Chapter 5, *Security* section of this guide.

2.3. Components

2.3.1. Ozone Widget Framework (OWF) Web Application

2.3.1.1. Frontend

The frontend is built using ReactJs and TypeScript. The frontend client is transpiled into Javascript and minimized into smaller files to optimize loading of the application in the browser. This bundled client application is then served up as static files by the backend.

2.3.1.2. Backend

OWF is built using Python, an interpreted programming language, meaning the application will not be compiled into a single executable and all of the files necessary to start and run the application is located in the `OWF-8.0.0.0ga/` directory. The backend bundle will include a pre-built frontend that will be served up to the clients. OWF uses a Django package, WhiteNoise (https://github.com/evansd/whitenoise/), to serve up static files. OWF's default configuration will serve up static files from `config/staticfiles`.

2.3.1.3. Sample Widgets

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

The example widgets are also included in the bundle by default and are located under the static files directory (`/config/staticfiles`).

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 running OWF:

- Python 3.7.4 or higher.
- A Relational Database Management System (RDBMS). Oracle, MySQL, MSSQL, or Postgres.
- Docker 17.04.0 or higher (optional)

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 the bundled frontend and the Python/Django backend necessary to setup and run OWF in a development environment.

- `OWF-8.0.0.0ga/` - bundled application that includes the client and the server.
- `docs/` – Copies of the application documentation and guides.

3.3.2. Running the Application

The following instructions explain how an administrator might start the Ozone application from the bundle.

3.3.2.1. Prerequisites

- A supported database is running.
- The migration of the database is complete.

    - More details can be found later in the Chapter 4, *Database Configuration* section of this guide.

- All of the backend dependencies are installed.
    - This can be accomplished by running `pip install -r requirements_prod.txt` command from the `OWF-8.0.0.0ga/` directory in the bundle

Once the prerequisites are met, run `waitress-serve --port=8000 --url-scheme=https config.wsgi:application` in `OWF-8.0.0.0ga/`

3.4. Default Installation

The default configuration will attempt to connect to a postgres database with the following options:

```
host: localhost
port: 5432
database name: postgres
database user: postgres
database password: postgres
```

When using this standard configuration, OWF uses the default security module which provides a simple username and password login form for authentication.

The default installation will not be behind a secure connection. In order to connect to the application through SSL/TLS, the application should be behind a reverse proxy server. An example configuration using Nginx as the reverse proxy server is provided later in the Chapter 5, *Security* section of this guide

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, 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 configure an external database for OWF.

Below is an example setting for defining a connection to the database. This setting is defined in `config/production.py`

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'localhost',
        'PORT': 5432,
        # Wraps each web request in a transaction. So if anything fails, it
will rollback automatically.
        'ATOMIC_REQUESTS': True,
    }
}
```

Supported engine string for OWF are listed below
`'django.db.backends.postgresql'`
`'django.db.backends.mysql'`
`'django.db.backends.oracle'`
`'sql_server.pyodbc'`

Depending on the database that you choose to use, you will need to install the appropriate packages for Django to support that database. You can simply add any of the following packages to the `requirements_prod.txt` and execute `pip install -r requirements_prod.txt`

You can configure OWF's database connection settings using environment variables. If the environment variable does not exist, it will default to the values listed below. If further customization is needed, you can do so by modifying the `config/production.py` file.

Property	Default Value
`OWF_DB_ENGINE`	`django.db.backends.postgresql`
`OWF_DB_NAME`	`postgres`
`OWF_DB_USER`	`postgres`
`OWF_DB_PASSWORD`	`postgres`
`OWF_DB_HOST`	`localhost`
`OWF_DB_PORT`	`5432`

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.

5. Security

5.1. Overview

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

OWF uses a pluggable security solution using Django middlewares. This documentation will include instructions on how to configure them.

Familiarity with Django settings (https://docs.djangoproject.com/en/2.2/ref/settings/) 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 django's authentication mechanism.

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.

5.2.1. Adding Users, Roles, & Groups

The addition of users and groups 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.

5.2.2. Installing Security Modules

A security module can be enabled by setting the environment variable associated with enabling the security module or to set the settings value manually in `config/settings/base.py` to enable the the desired authentication mechanism.

5.2.3. Basic Authentication

By default, OWF uses basic authentication as its security mechanism. Basic auth is a username and password based authentication.

When using basic authentication users can be added to OWF via python shell.

```
python manage.py shell
>>> from people.models import Person
>>> Person.objects.create_user(email='', username='', password='')
```

5.2.4. CAC Authentication - X.509 Certificate (PKI)

CAC authentication is a certificate based authentication. Because OWF does not support communication over secure channels out-of-the-box, the proper way to enable certficate based authentication is to standup OWF behind a reverse proxy to handle the validation and decryption of client certificates.

When CAC authentication is enabled, OWF assumes the client's certficiate has been validated and decrypted for any request it receives.

The following options are used for configuring certificate authentication in the OWF application config.

Setting	Env Var	Default Value	Description
ENABLE_SSL_AUTH	OWF_ENABLE_SSL_AUTH	False	enable or disable certificate based authentication.
USER_AUTH_STATUS_HEADER	OWF_USER_AUTH_STATUS_HEADER	HTTP_X_SSL_AUTHENTICATED	header that contains the authentication status of the request.
USER_DN_SSL_HEADER	OWF_USER_DN_SSL_HEADER	HTTP_X_SSL_USER_DN	header that contains the certificate's DN.
EXTRACT_USERDATA_FN	OWF_EXTRACT_USERDATA_FN	config.ssl_auth.example.get_cac_id	location of the function to parse data from the certificate's DN.

The following is an example config file for nginx, a reverse proxy.

```
worker_processes  5;

events {
    worker_connections  1024;
}

http {
    server {
        listen       443 ssl;
        server_name  _;
        add_header Strict-Transport-Security max-age=31536000;

        # Certificate locations
        ssl_certificate      /etc/nginx/certs/localhost.crt;
        ssl_certificate_key  /etc/nginx/certs/localhost.key;
        ssl_client_certificate      /etc/nginx/certs/alldodcerts.pem;
        ssl_verify_client   optional;
        # Increase verify_depth if there are intermediate CAs,
        ssl_verify_depth   3;

        ssl_session_timeout  5m;

        # Disallow poor algorithms for SSL
        ssl_protocols  SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers
ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5;
        ssl_prefer_server_ciphers   on;

        location / {
            if ($ssl_client_verify != SUCCESS) {
```

```
                return 403;
            }
            # Change this based on where your application is running
            proxy_pass http://localhost:8000/;
            proxy_pass_header Server;
            proxy_set_header Host $http_host;
            proxy_redirect off;
            proxy_connect_timeout 60;
            proxy_read_timeout 90;

            # SSL settings for django to handle ssl auth
            proxy_set_header X-SSL-User-DN $ssl_client_s_dn;
            proxy_set_header X-SSL-Authenticated $ssl_client_verify;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Scheme $scheme;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

    }
}
```

More details on the nginx http ssl module can be found http://nginx.org/en/docs/http/ngx_http_ssl_module.html

For more info on how to test with the provided samples, refer to the README in the `/ozone-framework-python-server/config/ssl_auth` directory.

5.2.4.1. 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 `config/settings/ssl_auth/samples` 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.

5.2.5. CAS Authentication

OWF comes with support for CAS based authentication. OWF leverages the `django-ng-cas` package to implement the CAS client used for this security mechanism.

The following options are used for configuring CAS authentication in the OWF application config.

Setting	Env Var	Default Value	Description
ENABLE_CAS	OWF_ENABLE_CAS	False	enables or disabled CAS based authentication
CAS_EXTRA_LOGIN_PARAMS	OWF_CAS_EXTRA_LOGIN_PARAMETERS	{}	extra URL parameters to add to the login URL when redirecting the user
CAS_RENAME_ATTRIBUTES		{ uid: username }	a dict used to rename the (key of the) attributes that the CAS server may retrun
CAS_SERVER_URL	OWF_CAS_SERVER_URL		base URL of your CAS source
CAS_VERSION	OWF_CAS_VERSION	2	The CAS protocol version to use. '1' '2' '3' and 'CAS_2_SAML_1_0' are supported

More details on the django-cas-ng package can be found https://github.com/mingchen/django-cas-ng

6. Configuration

OWF offers custom configuration options via the settings files used by Django, located in `config/settings`. Once changes are made, restart OWF to have the changes take effect.

The default settings provided in the `config/settings/production.py` 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 throughout this guide.

6.1. 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

6.1.1. Changing the location of help files

The help directory location is defined by the `HELP_FILES` property in `OWF-8.0.0.0ga/config/`.

By default, help files are located in the `/config/staticfiles/` directory. In the default OWF bundle, this may be found at `OWF-8.0.0.0ga/`.

To change the directory location, replace `HELP_FILES` and then run `python manage.py collectstatic --settings=config.settings.production --no-input`

6.1.2. Environmental Variables (.env file)

The Environmental variables are configurations for the OWF server and client that will effect general functionality of the application in many ways. Depending on the modifications to the configurations at are made the application can be drastically altered to fit the needs of the end user. Please see the [env_configuration] documentation for more information regarding the specifics to each of the configuration settings that the environmental variables provide.

If the user changes these environmental variables it is recommended that the OWF server is restarted, otherwise these changes wont take effect.

6.1.3. 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 custom security warning requires modifications to one of the files in the client application and will

require a re-build of the client application. Modifications will need to be made in the `messages.ts` file located in the `ozone-framework-client/packages/application/src/environment/` directory. Once the changes have been made, follow the instructions in the **Build Guide** to re-build and deploy the client application.

6.2. Connecting to AML (Marketplace)

In order to the connect AML Marketplace with OWF-8.0.0.0ga, you must modify the following files in your AML installation:

```
aml-backend\Makefile
aml-frontend\webpack.dev.js
aml-frontend\webpack.prod.js
aml-frontend\src\components\shared\LaunchModal.js
```

Patching and possibly modifying with the files provided in the 'aml_ozone_patch.zip' compressed directory that is included with OWF-8.0.0.0ga.

6.2.1. Modifying AML Patch files

Within the `aml-frontend\webpack.prod.js` file, set the PERMITTED_OZONE_PARENT_DOMAIN variable equal to the URL for your OWF instance. If you wish to run the system in dev mode, make the same change in `aml-frontend\webpack.dev.js` as well.

7. Logging

7.1. Logging Configuration

Logging can be configured by editing the `base.py` file which can be found in the `config/settings/` directory.

Example logging configuration for the logger — base.py

```
#  LOG
if not os.path.exists('./logs'):
    os.mkdir('./logs')

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'ignore_markdown_logs': {
            '()': 'config.owf_utils.owf_logging_backends.MarkDownFilter',
        },
        'ignore_reload_logs': {
            '()': 'config.owf_utils.owf_logging_backends.ReloadFilter',
        },
        'ignore_favicon_logs': {
            '()': 'config.owf_utils.owf_logging_backends.FaviconFilter',
        },
    },
    'formatters': {
        'console': {
            'format': '%(levelname)-8s SHOST: [%(hostname)s] TIME [
%(asctime)s ] %(name)-12s %(message)s ',
            'class':
'config.owf_utils.owf_logging_backends.HostnameAddingFormatter',
        },
        'cef-format': {
            'format': '%(asctime)s CEF shost=%(hostname)s %(message)s ',
            'datefmt': "%d/%b/%Y %H:%M:%S",
            'class':
'config.owf_utils.owf_logging_backends.HostnameAddingFormatter',
        },
        'event-format': {
            'format': '%(levelname)-8s SHOST: [%(hostname)s] TIME [
%(asctime)s ] %(name)-12s %(message)s ',
            'datefmt': "%d/%b/%Y %H:%M:%S",
            'class':
'config.owf_utils.owf_logging_backends.HostnameAddingFormatter',
        }
    },
```

```
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'filters': ['ignore_markdown_logs', 'ignore_reload_logs',
'ignore_favicon_logs'],
            'formatter': 'console'
        },
        'cef-file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'filters': ['ignore_markdown_logs', 'ignore_reload_logs',
'ignore_favicon_logs'],
            'formatter': 'cef-format',
            'filename': os.getenv('CEF_LOCATION', 'logs') + '/owf-cef.log',
            'maxBytes': 50000,
            'backupCount': 2,
        },
        'event-file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'filters': ['ignore_markdown_logs', 'ignore_reload_logs',
'ignore_favicon_logs'],
            'formatter': 'event-format',
            'filename': os.getenv('CEF_LOCATION', 'logs') + '/owf-events.log',
            'maxBytes': 50000,
            'backupCount': 2,
        }
    },
    'loggers': {
        'owf.enable.cef.object.access.logging': {
            'level': 'DEBUG',
            'handlers': ['console', 'event-file']
        },
        'owf.enable.cef.logging': {
            'level': 'DEBUG',
            'handlers': ['console', 'cef-file']
        },
        'django.security.DisallowedHost': {
            'handlers': ['console', ],
            'propagate': False,
            'level': 'ERROR',
        },
        # 'django.db.backends': {
        #     'handlers': ['console'],
        #     'level': 'DEBUG',
        # },
        # Captures All SQL Expressions that are run in the server when
uncommented
        # WARNING IF RUN IN PRODUCTION THIS WILL SLOW DOWN THE APPLICATION
    }
}
```

Logging supports configuration using a Django (python wrapper) Domain Specific Language (DSL) as demonstrated above.

The configuration options and syntax for the Django logger configuration are documented in the Django documentation as well as the python documentation, listed below.

References:

- Django: Logger Configuration (DSL) – https://docs.djangoproject.com/en/2.2/topics/logging/
- Python >= v3.7.x: Logging Guide – https://docs.python.org/3.7/howto/logging-cookbook.html

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

Sign-in events are logged by the `owf.enable.cef.object.access.logging` logger. This logger supports two levels of logging: `info` and `debug`, with the latter providing more detailed information about each sign-in event. This is located in the handler file withing the appconf folder `appconf/handlers.py`.

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

Failed sign-in event (INFO)

```
INFO     SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:52:55 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 USER: admin[USER LOGIN]:
ACCESS DENIED with FAILURE MSG: [Login for admin] attempted with authenticated
credentials
```

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

Failed sign-in event (DEBUG)

```
DEBUG    SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:32:20 ]
owf.enable.cef.object.access.logging  IP: 127.0.0.1 USER: admin [USER LOGIN]:
ACCESS DENIED with FAILURE MSG: [Login for admin attempted with authenticated
credentials]
```

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

Successful certificate sign-in event (INFO)

```
INFO     SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:54:17 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 User: admin [USER LOGIN]:
LOGIN SUCCESS - ACCESS GRANTED USER [admin] with EMAIL [admin@goss.com]
```

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

Successful certificate sign-in event (DEBUG)

```
DEBUG    SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:36:36 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 User: admin [USER LOGIN]:
LOGIN SUCCESS - ACCESS GRANTED USER [admin] with EMAIL [admin@goss.com]
```

7.2.2. Logout Events

Sign-out events are logged by the `owf.enable.cef.object.access.logging` logger. This logger supports two levels of logging: `info` and `debug`, with the latter providing more detailed information about each sign-out event. This is located in the handler file withing the appconf folder `appconf/handlers.py`.

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     SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:54:54 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 SessionID:
k8ng2mgu1d9ycm7ofppdbhorfcbftqp4 USER: admin [USER LOGOUT]
```

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

Sign-out event (DEBUG)

```
DEBUG    SHOST: [xxxxxxxx] TIME [ 28/Nov/2019 00:48:33 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 SessionID:
jfe6idrvl7vpeacaebb8a5iw20v7f2rp USER: admin [USER LOGOUT] with EMAIL
admin@goss.com with LAST LOGIN DATE [ 2019-11-28 00:36:36.873949+00:00 ]
```

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

Session time-out event (INFO)

```
INFO    SHOST: [xxxx.xxxx.xxxx] TIME [ 10/Dec/2019 20:11:03 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 SessionID:
oanq3tguneb6i6oayt5fgm8c8v3lgdpe USER: admin [USER SESSION TIMEOUT]
```

Session time-out event (DEBUG)

```
DEBUG    SHOST: [xxxx.xxxx.xxxx] TIME [ 10/Dec/2019 20:11:03 ]
owf.enable.cef.object.access.logging IP: 127.0.0.1 SessionID:
oanq3tguneb6i6oayt5fgm8c8v3lgdpe USER: admin [USER SESSION TIMEOUT], with ID
[1], with EMAIL [admin@goss.com], with LAST LOGIN DATE [2019-12-10
20:10:34.940871+00:00]
```

7.3. Common Event Format (CEF) Auditing

Common Event Format (CEF) auditing capabilities are available in OWF. To enable/disable them, sign into OWF 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. This code is middleware and located in `config/owf_utils/log_middleware.py`.

When enabled, CEF auditing records common user events:

- Sign in and out requests
- Create, Read, Edit and Delete requests
- Import and Export requests

The following are two log examples using CEF auditing:

CEF auditing from an object modification event

```
28/Nov/2019 01:04:36 CEF shost=xxxx suid=admin
requestMethod=USER_INITIATED|PUT outcome=200 data=<QueryDict: {'version':
['1571151178'], 'created_date': ['2019-10-15'], 'edited_date': ['2019-10-15'],
'code': ['owf.job.disable.accounts.start.time'], 'value': ['23:59:59'],
'title': ['Disable Accounts Job Start Time'], 'description': [''], 'type':
['String'], 'group_name': ['HIDDEN'], 'sub_group_name': [''], 'mutable':
['true'], 'sub_group_order': ['1'], 'help': ['']}> urlName=admin_application-
configuration-detail requestType=<WSGIRequest: PUT '/api/v2/admin/application-
configuration/11/'>
```

CEF auditing from a log-in event

```
28/Nov/2019 01:02:24 CEF shost=xxxx suid=admin
requestMethod=USER_INITIATED|POST outcome=302 data=<QueryDict:
{'csrfmiddlewaretoken':
['IpAMhvpgjUkSNa8WOOQwJjfeRz5SA73TD0YJfN2YGy51tdidvoqqC5MRx0wR8snH'],
'username': ['admin'], 'next': ['/admin/']}> urlName=login
requestType=<WSGIRequest: POST '/admin/login/?next=/admin/'>
```

8. Upgrading

OWF v8.0.0.0 is a complete rewrite of the OWF backend. Because of the new technology that OWF leverages, the legacy config files will no longer be used and the only thing that needs to migrate is the data in the database.

8.1. Data Migration

There are 2 modules in OWF that handle the exporting and importing of legacy data, into the new application. These modules are located with the rest of the OWF application modules. `ozone-framework-python-server/` in the repo and `OWF-8.x.x.x/` in the bundle.

/migration_tool/

a standalone library which helps connect different databases (mysql, postgres, oracle, mssql) and provides a core functionality to import / export data (SQL to JSON or JSON to SQL) where SQL can be any of the mentioned databases.

/migration_owf/

a wrapper built on top of migration_tool which utilizes the library to import, export and transform data **this is the directory that you will be accessing to run the migrations to the new database** This directory also contains a `README.md` that has more fine-grained details for using the tool.

/migration_owf/test_data

contains docker configurations for MYSQL, MSSQL, and POSTGRES databases as well as example dumps/migrations

8.1.1. SETUP

Install requirements from pip.

```
pip install -r migration_owf/requirements.txt
```

8.1.1.1. Install drivers

You may need to install additional drivers to connect to a MSSQL database

- Download Microsoft® ODBC Driver 17 for SQL Server
  - https://www.microsoft.com/en-us/download/details.aspx?id=56567

Configure Django to set up initial tables on your desired database type

```
# MYSQL
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysql_db',
        'USER': 'admin',
        'PASSWORD': 'password',
        'HOST': 'localhost', # if localhost doesn't work, try '127.0.0.1'
        'PORT': '3306',
        # Wraps each web request in a transaction. So if anything fails, it
will rollback automatically.
        'ATOMIC_REQUESTS': True,
    }
}
```

```
# Postgres
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'owf',
        'PASSWORD': 'password',
        'HOST': 'localhost', # if localhost doesn't work, try '127.0.0.1'
        'PORT': '5432',
        # Wraps each web request in a transaction. So if anything fails, it
will rollback automatically.
        'ATOMIC_REQUESTS': True,
    }
}
```

```
# MSSQL
DATABASES = {
    'default': {
        'ENGINE': 'sql_server.pyodbc',
        'NAME': 'owf_new',
        'USER': 'sa',
        'PASSWORD': 'superstrong_password123',
        'HOST': 'localhost',
        'PORT': '1433',
        # Wraps each web request in a transaction. So if anything fails, it
will rollback automatically.
        'ATOMIC_REQUESTS': True,
        'OPTIONS': {
            'driver': 'ODBC Driver 17 for SQL Server', # You may need to
select a different driver
        },
    }
}
```

```
# Oracle
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.oracle',
        'NAME': 'localhost:1521/ORCLDB',
        'USER': 'system',
        'PASSWORD': 'Welcome1',
    }
}
```

8.1.2. USAGE

1. Change the database connections in `import.py` and `export.py` files.
2. Export the databases by running `` `cd migration_owf `` & `python export.py`

`cd migration_owf` & `python import.py`

Make sure to change `import_db` and `import_db_var` in import.py to the desired database.

On a fresh django db, make sure to run `./manage.py migrate` for initial table setup.

Start the backend server by running `python manage.py runserver`

If you wish to create a new administrator account you can do so by running `python manage.py createsuperuser`

Environmental Variable Configuration (.env file)

Environmental Variable	Purpose
`DEBUG=True`	This boolean value represents the applications state when its running. When DEBUG is set to `True` the application will give detailed error messages and be running in a development state.
`SECRET_KEY=xxxxxxxxx`	This value is the key to securing signed data – it is vital you keep this secure, or attackers could use it to generate their own signed values.
`ALLOWED_HOSTS=['*']`	This is a list of strings representing the host/domain names that this Django site can serve. This is a security measure to prevent HTTP Host header attacks, which are possible even under many seemingly-safe web server configurations.
`OWF_DB_ENGINE=django.db.backends.postgresql`	This value is the the pointer to the Django provided backend engine that will manage the interactions between the different databases. Values for the other backends can be located here at the Django official documentation.
`OWF_DB_NAME=postgres`	This value is your database **name** that you will need to connect to your database.
`OWF_DB_USER=postgres`	This value is your database **username** that you will need to connect to your database.
`OWF_DB_PASSWORD=postgres`	This value is your database **password** that you will need to connect to your database.
`OWF_DB_HOST=localhost`	This value is your database **host** location that you will need to connect to your database.
`OWF_DB_PORT=5432`	This value is the database **port number** needed to identify the location of the database related to the host.
`CEF_LOCATION=logs`	This is the logging files directory location. When changing this location value the directory needs to exist before the application can run & the server needs to be restarted if the location has changed. NOTE: *The recommended way of interacting with this configuration is to go through the API to keep the DB settings and the* `.env` *file in sync.*
`OWF_ENABLE_CAS=False`	When this value is enabled it allows the CAS (Central Authentication Service) to serve as a alternate form of authorization for the application. Please refer to this section on the implementation [cas-config]
`OWF_CAS_EXTRA_LOGIN_PARAMETERS={}`	When CAS is enabled this is the extra URL parameters to add to the login URL when redirecting the user. Please refer to this section on the implementation [cas-config]. For more information read about the CAS integration here.
`OWF_CAS_USERNAME_ATTRIBUTE=uid`	Please refer to this section on the implementation [cas-config]. For more information read about the CAS integration here.
`OWF_CAS_SERVER_URL=`	Please refer to this section on the implementation [cas-config].For more information read about the CAS integration here.

Environmental Variable	Purpose
`OWF_CAS_VERSION=2`	Please refer to this section on the implementation [cas-config]. For more information read about the CAS integration here.
`OWF_CAS_CREATE_USER=False`	Please refer to this section on the implementation [cas-config]. For more information read about the CAS integration here.
`OWF_CAS_STORE_NEXT=True`	Please refer to this section on the implementation [cas-config]. For more information read about the CAS integration here.
`OWF_ENABLE_SSL_AUTH=False`	This Boolean value is represents the applications ability to handle the CAC (Common Access Card) based login through SSL verification. Please refer to this Section 5.2.4, "CAC Authentication - X.509 Certificate (PKI)" documentation for detailed information on how this security feature works.
`OWF_EXTRACT_USERDATA_FN=config.ssl_auth.example.get_cac_id`	This value is used to parse data from the certificate's DN. Please refer to this Section 5.2.4, "CAC Authentication - X.509 Certificate (PKI)" documentation for detailed information on how this security feature works.
`OWF_USER_DN_SSL_HEADER=HTTP_X_SSL_USER_DN`	Please refer to this Section 5.2.4, "CAC Authentication - X.509 Certificate (PKI)" documentation for detailed information on how this security feature works.
`OWF_USER_AUTH_STATUS_HEADER=HTTP_X_SSL_AUTHENTICATED`	Please refer to this Section 5.2.4, "CAC Authentication - X.509 Certificate (PKI)" documentation for detailed information on how this security feature works.
`ENABLE_METRICS=False`	This boolean value is the control for metrics application metrics to be enabled. Please refer to the [admin-managers] for more information.
`METRICS_SERVER_URL=http://localhost:3000/metric`	Please refer to the [admin-managers] for more information.
`SESSION_EXPIRE_AT_BROWSER_CLOSE=True`	This boolean value is the control for sessions for browser interactions. If the value is `True` then the session of the logged in user will expire at the moment of the browser being closed. If it is false that users session will still be valid until the `SESSION_COOKIE_AGE` time has lapsed. SESSION_COOKIE_AGE can be found withing the `base.py` located in `/config/settings/`.
`ENABLE_USER_AGREEMENT=True`	This boolean value is the control for the user agreement popup that is displayed. When set to `True` the popup will be displayed. When set to `False` the popup will be disabled.
`ENABLE_CONSENT=True`	This boolean value is the control for the user consent link that is displayed. When set to `True` the link will be displayed. When set to `False` the link will be disabled.
`SERVER_URL=http://localhost:8000`	This value is the server location and port number of the application. This is used within aspects of the application as a reference point. NOTE: *This should be changed anytime that the application is running on anything other than the default values of the Django server.*

Environmental Variable	Purpose
`ENABLE_LOGIN=True`	This boolean value when `True` is allowing the user to login through the django based authentication system. When `False` it is assumed that you will be providing your own login authentication.
`ENABLE_LOGOUT=True`	This boolean value when `True` allows the user to logout through the user interface. When `False` the logout is disabled through the user interface.

9. Known Issues

9.1. Backend

- Upgrading (migration_owf scripts)
  - The migration script does not work for users upgrading from an Oracle database
    - This issue is related to a 4000 character limit that Oracle has in place. Presently the dashboards and preferences tables cause errors.
    - Note: There are no issues connecting to oracle database in the backend if no migration was required

9.2. Frontend

- N/A

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 stacks (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.