# OneLogin's SAML PHP Toolkit


Add SAML support to your PHP softwares using this library. Forget those complicated libraries and use that open source library provided and supported by OneLogin Inc.

## Why add SAML support to my software?

SAML is an XML-based standard for web browser single sign-on and is defined by the OASIS Security Services Technical Committee. The standard has been around since 2002, but lately it is becoming popular due its advantages:

- **Usability** - One-click access from portals or intranets, deep linking, password elimination and automatically renewing sessions make life easier for the user.
- **Security** - Based on strong digital signatures for authentication and integrity, SAML is a secure single sign-on protocol that the largest and most security conscious enterprises in the world rely on.
- **Speed** - SAML is fast. One browser redirect is all it takes to securely sign a user into an application.
- **Phishing Prevention** - If you don’t have a password for an app, you can’t be tricked into entering it on a fake login page.
- **IT Friendly** - SAML simplifies life for IT because it centralizes authentication, provides greater visibility and makes directory integration easier.
- **Opportunity** - B2B cloud vendor should support SAML to facilitate the integration of their product.

## General description

OneLogin's SAML PHP toolkit let you build a SP (Service Provider) over your PHP application and connect it to any IdP (Identity Provider).

Supports:

- SSO and SLO (SP-Initiated and IdP-Initiated).
- Assertion and nameId encryption.
- Assertion signature.
- Message signature: AuthNRequest, LogoutRequest, LogoutResponses.
- Enable an Assertion Consumer Service endpoint.
- Enable an Single Logour Service endpoint.
- Publish the SP metadata (which can be signed).

Key features:

- **saml2int** - Implements the SAML 2.0 Web Browser SSO Profile.
- **Session-less** - Forget those common conflicts between the SP and the final app, the toolkit delegate session in the final app.

- **Easy to use** - Programmer will be allowed to code high-level and low-level programming, 2 easy to use APIs are available.
- **Tested** - Thoroughly tested.
- **Popular** - OneLogin's customers use it. Many PHP SAML plugins uses it.

# Installation

## Dependences

- php >= 5.3.3
- openssl. Install the openssl library. It handles x509 certificates.
- mcrypt. Install that library and its php driver if you gonna handle encrypted data. (nameID, assertions)
- gettext. Install that library and its php driver. It handles translations.

## Code

### Option 1. Download from github

The toolkit is hosted on github. You can download it from:

- Lastest release: https://github.com/onelogin/php-saml/releases/tag/v2.0.0
- Master repo: https://github.com/onelogin/php-saml/tree/master

Copy the core of the library inside the php application. (each application has its structure so take your time to locate the PHP SAML toolkit in the best place). See the "Guide to add SAML support to my app" to know how.

### Option 2. Composer

The toolkit supports composer. You can find the onelogin/php-saml package at https://packagist.org/packages/onelogin/php-saml

In order to import the saml toolkit to your current php project, just add to the composer.json file this require:

"onelogin/php-saml": "master-dev"

Done that, execute:

```
composer install
```

you will find at the "vendor" folder a new folder named *onelogin* and inside the *php-saml*.

Composer will automatically handle the libraries (with require "vendor/autoload.php" you will load all the vendor libraries).

**Important** In this option, the x509 certs must be stored at *vendor/onelogin/php-saml/certs* and settings file should be stored at *vendor/onelogin/php-saml*. But could be easier to use the "array method" to provide settings info (explained later)

# Compatibility

This 2.0 version has a new library. The toolkit is still compatible.

The old code that you used in order to add SAML support gonna continue working with minor changes. You only need to load the files of the lib/Saml folder. (notice that the compatibility.php file do that).

The old-demo folder contains code from an old app that uses the old version of the toolkit (v.1). Take a look.

Sometimes the names of the classes of the old code could be a bit different and if is your case you must change them for OneLogin_Saml_Settings, OneLogin_Saml_Response, OneLogin_Saml_AuthRequest or OneLogin_Saml_Metadata.

We recommend that you migrate the old code to the new one to be able to use the new features that the new library Saml2 carries.

# Namespaces

If you are using the library with a framework like Symphony2 that contains namespaces, remember that calls to the class must be done by adding a \ to the start, for example to use the static method getSelfURLNoQuery use: \OneLogin_Saml2_Utils::getSelfURLNoQuery()

# Getting started

## Knowing the toolkit

The new OneLogin SAML Toolkit contains different folders (certs, endpoints, extlib, lib, demo, etc) and some files.

Let's start describing the folders:

**certs**

SAML requires a x.509 cert to sign and encrypt elements like NameID, Message, Assertion, Metadata.

If our environment requires sign or encrypt support, this folder may contain the x509 cert and the private key that the SP will use:

- **sp.crt** The public cert of the SP
- **sp.key** The privake key of the SP

Or also we can provide those data in the setting file at the $settings['sp']['x509cert'] and the $settings['sp']['privateKey'].

Sometimes we could need a signature on the metadata published by the SP, in this case we could use the x.509 cert previously mentioned or use a new x.509 cert: **metadata.crt** and **metadata.key**.

#### extlib

This folder contains the 3rd party libraries that the toolkit uses. At the moment only uses the xmlseclibs (autor Robert Richards, BSD Licensed) which handle the sign and the encryption of xml elements.

#### lib

This folder contains the heart of the toolkit, the libraries:

- **Saml** folder contains a modified version of the toolkit v.1 and allows the old code to keep working. (This library is provided to maintain backward compatibility).
- **Saml2** folder contains the new version of the classes and methods that are described in a later section.

#### doc

This folder contains the API documentation of the toolkit.

#### endpoints

The toolkit has 3 endpoints:

- **metadata.php** - Where the metadata of the SP is published.
- **acs.php** - Assertion Consumer Service. Processes the SAML Responses.
- **sls.php** - Single Logout Service. Processes Logout Requests and Logout Responses.

You can use the files provided by the toolkit or create your own endpoints files when adding SAML support to your applications. Take in mind that those endpoints files uses the setting file of the toolkit's base folder.

#### locale

Locale folder contains some translations: en_US and es_ES as a proof of concept Currently there are no translations but we will eventually localize the messages and support multiple languages.

#### Other important files

- **settings_example_example.php** - A template to be used in order to create a settings.php file which contains the basic configuration info of the toolkit.
- **advanced_settings_example.php** - A template to be used in order to create a advanced_settings.php file which contains extra configuration info related to the security, the contact person, and the organization associated to the SP.

- **_toolkit_loader.php** - This file load the toolkit libraries (The SAML2 lib).
- **compatibility** - Import that file to make compatible your old code with the new toolkit. (loads the SAML library).

#### Miscellaneous

- **tests** - Contains the unit test of the toolkit.
- **demo1** - Contains an example of a simple PHP app with SAML support. Read the Readme.txt inside for more info.
- **demo2** - Contains another example.
- **demo-old** - Contains an example that uses the code of the older version of the the toolkit to demonstrate the backwards compatibility.

## How it works

#### Settings

First of all we need to configure the toolkit. The SP's info, the IdP's info, and in some cases, configure advanced security issues like signatures and encryption.

There are two ways to provide the settings information:

- Use a settings.php file that we should locate at the base folder of the toolkit.
- Use an array with the setting data and provide it directly to the constructor of the class.

There is a template file, settings_example.php, so you can make a copy of this file, rename and edit it.

```
<?php

$settings = array (
    // If 'strict' is True, then the PHP Toolkit will reject unsigned
    // or unencrypted messages if it expects them to be signed or encrypted.
    // Also it will reject the messages if the SAML standard is not strictly
    // followed: Destination, NameId, Conditions ... are validated too.
    'strict' => false,

    // Enable debug mode (to print errors).
    'debug' => false,

    // Service Provider Data that we are deploying.
    'sp' => array (
        // Identifier of the SP entity  (must be a URI)
        'entityId' => '',
        // Specifies info about where and how the <AuthnResponse> message MUST be
        // returned to the requester, in this case our SP.
        'assertionConsumerService' => array (
            // URL Location where the <Response> from the IdP will be returned
            'url' => '',
            // SAML protocol binding to be used when returning the <Response>
            // message. OneLogin Toolkit supports this endpoint for the
```

```
            // HTTP-POST binding only.
            'binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
        ),
        // Specifies info about where and how the <Logout Response> message MUST be
        // returned to the requester, in this case our SP.
        'singleLogoutService' => array (
            // URL Location where the <Response> from the IdP will be returned
            'url' => '',
            // SAML protocol binding to be used when returning the <Response>
            // message. OneLogin Toolkit supports the HTTP-Redirect binding
            // only for this endpoint.
            'binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
        ),
        // Specifies the constraints on the name identifier to be used to
        // represent the requested subject.
        // Take a look on lib/Saml2/Constants.php to see the NameIdFormat supported.
        'nameIdFormat' => 'urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress',
        // Usually x509cert and privateKey of the SP are provided by files placed
at
        // the certs folder. But we can also provide them with the following
parameters
        'x509cert' => '',
        'privateKey' > '',

    ),

    // Identity Provider Data that we want connected with our SP.
    'idp' => array (
        // Identifier of the IdP entity  (must be a URI)
        'entityId' => '',
        // SSO endpoint info of the IdP. (Authentication Request protocol)
        'singleSignOnService' => array (
            // URL Target of the IdP where the Authentication Request Message
            // will be sent.
            'url' => '',
            // SAML protocol binding to be used when returning the <Response>
            // message. OneLogin Toolkit supports the HTTP-Redirect binding
            // only for this endpoint.
            'binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
        ),
        // SLO endpoint info of the IdP.
        'singleLogoutService' => array (
            // URL Location of the IdP where SLO Request will be sent.
            'url' => '',
            // SAML protocol binding to be used when returning the <Response>
            // message. OneLogin Toolkit supports the HTTP-Redirect binding
            // only for this endpoint.
            'binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
        ),
        // Public x509 certificate of the IdP
        'x509cert' => '',
        /*
         *  Instead of use the whole x509cert you can use a fingerprint
         *  (openssl x509 -noout -fingerprint -in "idp.crt" to generate it)
         */
        // 'certFingerprint' => '',
    ),
```

```
);
```

In addition to the required settings data (idp, sp), there is extra information that could be defined. In the same way that a template exists for the basic info, there is a template for that advanced info located at the base folder of the toolkit and named advanced_settings_example.php that you can copy and rename it as advanced_settings.php

```
<?php

$advancedSettings = array (

    // Security settings
    'security' => array (

        /** signatures and encryptions offered */

        // Indicates that the nameID of the <samlp:logoutRequest> sent by this SP
        // will be encrypted.
        'nameIdEncrypted' => false,

        // Indicates whether the <samlp:AuthnRequest> messages sent by this SP
        // will be signed.  [Metadata of the SP will offer this info]
        'authnRequestsSigned' => false,

        // Indicates whether the <samlp:logoutRequest> messages sent by this SP
        // will be signed.
        'logoutRequestSigned' => false,

        // Indicates whether the <samlp:logoutResponse> messages sent by this SP
        // will be signed.
        'logoutResponseSigned' => false,

        /* Sign the Metadata
         False || True (use sp certs) || array (
                                                    keyFileName => 'metadata.key',
                                                    certFileName => 'metadata.crt'
                                                )
        */
        'signMetadata' => false,


        /** signatures and encryptions required **/

        // Indicates a requirement for the <samlp:Response>, <samlp:LogoutRequest>
        // and <samlp:LogoutResponse> elements received by this SP to be signed.
        'wantMessagesSigned' => false,

        // Indicates a requirement for the <saml:Assertion> elements received by
        // this SP to be signed. [Metadata of the SP will offer this info]
        'wantAssertionsSigned' => false,

        // Indicates a requirement for the NameID received by
        // this SP to be encrypted.
        'wantNameIdEncrypted' => false,
```

```
    ),

    // Contact information template, it is recommended to suply a
    // technical and support contacts.
    'contactPerson' => array (
        'technical' => array (
            'givenName' => '',
            'emailAddress' => ''
        ),
        'support' => array (
            'givenName' => '',
            'emailAddress' => ''
        ),
    ),

    // Organization information template, the info in en_US lang is
    // recomended, add more if required.
    'organization' => array (
        'en-US' => array(
            'name' => '',
            'displayname' => '',
            'url' => ''
        ),
    ),
);
```

In the security section, you can set the way that the SP will handle the messages and assertions. Contact the admin of the IdP and ask him what the IdP expects, and decide what validations will handle the SP and what requirements the SP will have and communicate them to the IdP's admin too.

Once we know what kind of data could be configured, let's talk about the way settings are handled within the toolkit.

The settings files described (settings.php and advanced_settings.php) are loaded by the toolkit if not other array with settings info is provided in the constructors of the toolkit. Let's see some examples.

```
// Initializes toolkit with settings.php & advanced_settings files.
$auth = new OneLogin_Saml2_Auth();
//or
$settings = new OneLogin_Saml2_Settings();

// Initializes toolkit with the array provided.
$auth = new OneLogin_Saml2_Auth($settingsInfo);
//or
$settings = new OneLogin_Saml2_Settings($settingsInfo);
```

You can declare the $settingsInfo in the file that constains the constructor execution or locate them in any file and load the file in order to get the array available as we see in the following example:

```
<?php

require_once 'custom_settings.php';  // The custom_settings.php contains a
                                     // $settingsInfo array.

$auth = new OneLogin_Saml2_Auth($settingsInfo);
```

**How load the library**

In order to use the toolkit library you need to import the _toolkit_loader.php file located on the base folder of the toolkit. You can load this file in this way:

```
<?php

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once(TOOLKIT_PATH . '_toolkit_loader.php');
```

After that line we will be able to use the classes (and their methods) of the toolkit (because the external and the Saml2 libraries files are loaded).

If you wrote the code of your SAML app for the version 1 of the PHP-SAML toolkit you will need to load the compatibility.php, file which loads the SAML library files, in addition to the the _toolkit_loader.php.

That SAML library uses the new classes and methods of the latest version of the toolkits but maintain the old classes, methods, and workflow of the old process to accomplish the same things.

We strongly recommend migrating your old code and use the new API of the new toolkit due there are a lot of new features that you can't handle with the old code.

**Initiate SSO**

In order to send an AuthNRequest to the IdP:

```
<?php

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once(TOOLKIT_PATH . '_toolkit_loader.php');   // We load the SAML2 lib

$auth = new OneLogin_Saml_Auth(); // Constructor of the SP, loads settings.php
                                  // and advanced_settings.php
$auth->login();   // Method that sent the AuthNRequest
```

The AuthNRequest will be sent signed or unsigned based on the security info of the advanced_settings.php ('authnRequestsSigned').

The IdP will return the response to the Attribute Consumer Service of the SP. If we do not set a 'url'

param in the login method and we are using the default ACS provided by the toolkit (endpoints/acs.php), then the ACS endpoint will redirect the user to the file that launched the SSO request.

We can set an 'returnTo' url to change the workflow and redirect the user to the other PHP file.

```
$newTargetUrl = 'http://example.com/consume2.php';
$auth = new OneLogin_Saml_Auth();
$auth->login($newTargetUrl);
```

### The SP Endpoints

Related to the SP there are 3 important views: The metadata view, the ACS view and the SLS view. The toolkit provides examples of those views in the endpoints directory.

#### SP Metadata endpoints/metadata.php

This code will provide the XML metadata file of our SP, based on the info that we provided in the settings files.

```
<?php

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once dirname(TOOLKIT_PATH.'/_toolkit_loader.php';

try {
    $auth = new OneLogin_Saml2_Auth();
    $settings = $auth->getSettings();
    $metadata = $settings->getSPMetadata();
    $errors = $settings->validateMetadata($metadata);
    if (empty($errors)) {
        header('Content-Type: text/xml');
        echo $metadata;
    } else {
        throw new OneLogin_Saml2_Error(
            'Invalid SP metadata: '.implode(', ', $errors),
            OneLogin_Saml2_Error::METADATA_SP_INVALID
        );
    }
} catch (Exception $e) {
    echo $e->getMessage();
}
```

The getSPMetadata will return the metadata signed or not based on the security info of the advanced_settings.php ('signMetadata').

Before the XML metadata is exposed, a validation takes place to ensure that the info to be provided is authorized.

**Attribute Consumer Service(ACS) endpoints/acs.php**

This code handles the SAML response that the IdP returns to the SP.

```
<?php

session_start();  // IMPORTANT: This is required in order to be able
                  // to store the user data in the session.

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once dirname(TOOLKIT_PATH.'/_toolkit_loader.php';

$auth = new OneLogin_Saml2_Auth();

$auth->processResponse();

$errors = $auth->getErrors();

if (!empty($errors)) {
    print_r('<p>'.implode(', ', $errors).'</p>');
    exit();
}

if (!$auth->isAuthenticated()) {
    echo "<p>Not authenticated</p>";
    exit();
}

$_SESSION['samlUserdata'] = $auth->getAttributes();
if (isset($_POST['RelayState']) && OneLogin_Saml2_Utils::getSelfURL() !=
$_POST['RelayState']) {
    $auth->redirectTo($_POST['RelayState']);
}

$attributes = $_SESSION['samlUserdata'];

if (!empty($attributes)) {
    echo '<h1>'._('User attributes:').'</h1>';
    echo
'<table><thead><th>'._('Name').'</th><th>'._('Values').'</th></thead><tbody>';
    foreach ($attributes as $attributeName => $attributeValues) {
        echo '<tr><td>' . htmlentities($attributeName) . '</td><td><ul>';
        foreach ($attributeValues as $attributeValue) {
            echo '<li>' . htmlentities($attributeValue) . '</li>';
        }
        echo '</ul></td></tr>';
    }
    echo '</tbody></table>';
} else {
    echo _('Attributes not found');
}
```

The SAML response is processed and then checked that there are no errors. It also verifies that the user is authenticated and stored the userdata in session.

At that point there are 2 possible alternatives:

1. If no RelayState is provided, we could show the user data in this view or however we wanted.
2. If RelayState is provided, a rediretion take place.

Notice that we saved the user data in the session before the redirection to have the user data available at the RelayState view.

#### The getAttributes method

In order to retrieve attributes we can use:

```
$attributes = $auth->getAttributes();
```

With this method we get all the user data provided by the IdP in the Assertion of the SAML Response.

If we execute `print_r($attributes)` we could get:

```
Array
(
    [cn] => Array
        (
            [0] => Jhon
        )
    [sn] => Array
        (
            [0] => Doe
        )
    [mail] => Array
        (
            [0] => jhon.doe@example.com
        )
    [groups] => Array
        (
            [0] => users
            [1] => members
        )
)
```

Each attribute name can be used as an index into $attributes to obtain the value. Every attribute value is an array - a single-valued attribute is an array of a single element.

The following code is equivalent:

```
$attributes = $auth->getAttributes();
print_r($attributes['cn']);
```

```
print_r($auth->getAttribute('cn'));
```

Before trying to get an attribute, check that the user is authenticated. If the user isn't authenticated, an empty array will be returned. For example, if we call to getAttributes before a $auth->processResponse, the getAttributes() will return an empty array.

#### Single Logout Service (SLS) endpoints/sls.php

This code handles the Logout Request and the Logout Responses.

```
<?php

session_start();  // IMPORTANT: This is required in order to be able
                  // to close the user session.

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once dirname(TOOLKIT_PATH.'/_toolkit_loader.php';

$auth = new OneLogin_Saml2_Auth();

$auth->processSLO();

$errors = $auth->getErrors();

if (empty($errors)) {
    print_r('Sucessfully logged out');
} else {
    print_r(implode(', ', $errors));
}
```

If the SLS endpoints receives an Logout Request, the request is validated, the session is closed and a Logout Response is sent to the SLS endpoint of the idP.

```
// part of the processSLO method

$logoutResponse = new OneLogin_Saml2_LogoutResponse($this->_settings,
$_GET['SAMLResponse']);
if (!$logoutResponse->isValid($requestId)) {
    $this->_errors[] = 'invalid_logout_response';
} else if ($logoutResponse->getStatus() !==
OneLogin_Saml2_Constants::STATUS_SUCCESS) {
    $this->_errors[] = 'logout_not_success';
} else {
    if (!$keepLocalSession) {
        OneLogin_Saml2_Utils::deleteLocalSession();
    }
}
```

If the SLS endpoints receives a Logout Response, the response is validated and the session is closed, but if a RelayState is provided a redirection take place.

```
// part of the processSLO method

$decoded = base64_decode($_GET['SAMLRequest']);
$request = gzinflate($decoded);
if (!OneLogin_Saml2_LogoutRequest::isValid($this->_settings, $request)) {
    $this->_errors[] = 'invalid_logout_request';
} else {
    if (!$keepLocalSession) {
        OneLogin_Saml2_Utils::deleteLocalSession();
    }

    $inResponseTo = OneLogin_Saml2_LogoutRequest::getID($request);
    $responseBuilder = new OneLogin_Saml2_LogoutResponse($this->_settings);
    $responseBuilder->build($inResponseTo);
    $logoutResponse = $responseBuilder->getResponse();

    $parameters = array('SAMLResponse' => $logoutResponse);
    if (isset($_GET['RelayState'])) {
        $parameters['RelayState'] = $_GET['RelayState'];
    }

    $security = $this->_settings->getSecurityData();
    if (isset($security['logoutResponseSigned']) &&
$security['logoutResponseSigned']) {
        $signature = $this->buildResponseSignature($logoutResponse,
$parameters['RelayState']);
        $parameters['SigAlg'] = XMLSecurityKey::RSA_SHA1;
        $parameters['Signature'] = $signature;
    }

    $this->redirectTo($this->getSLOurl(), $parameters);
}
```

If we don't want that processSLO to destroy the session, pass a true parameter to the processSLO method

```
$keepLocalSession = true,
$auth->processSLO($keepLocalSession);
```

#### Initiate SLO

In order to send a Logout Request to the IdP:

```
<?php

define("TOOLKIT_PATH", '/var/www/php-saml/');
require_once(TOOLKIT_PATH . '_toolkit_loader.php');

$auth = new OneLogin_Saml_Auth();

$auth->logout();   // Method that sent the Logout Request.
```

The Logout Request will be sent signed or unsigned based on the security info of the advanced_settings.php ('logoutRequestSigned').

The IdP will return the Logout Response to the Single Logout Service of the SP. If we do not set an 'url' param in the logout method and are using the default SLS provided by the toolkit (endpoints/sls.php), then the SLS endpoint will redirect the user to the file that launched the SLO request.

We can set an 'returnTo' url to change the workflow and redirect the user to other php file.

```
$newTargetUrl = 'http://example.com/loggedOut.php';
$auth = new OneLogin_Saml_Auth();
$auth->logout($newTargetUrl);
```

**Example of a view that initiates the SSO request and handles the response (is the acs target)**

We can code a unique file that initiates the SSO process, handle the response, get the attributes, initiate the slo and processes the logout response.

Note: Review the demo1 folder that contains that use case; in a later section we explain the demo1 use case further in detail.

```
<?php

session_start();    // Initialize the session, we do that because
                    // Note that processResponse and processSLO
                    // methods could manipulate/close that session

require_once dirname(dirname(__FILE__)).'/_toolkit_loader.php'; // Load Saml2 and
                                                               // external libs
require_once 'settings.php';    // Load the setting info as an Array

$auth = new OneLogin_Saml2_Auth($settingsInfo);  // Initialize the SP SAML instance

if (isset($_GET['sso'])) {    // SSO action.  Will send an AuthNRequest to the IdP
    $auth->login();
} else if (isset($_GET['sso2'])) {              // Another SSO action
    $returnTo = $spBaseUrl.'/demo1/attrs.php';  // but set a custom RelayState URL
    $auth->login($returnTo);
} else if (isset($_GET['slo'])) {  // SLO action. Will sent a Logout Request to IdP
    $auth->logout();
} else if (isset($_GET['acs'])) {  // Assertion Consumer Service
    $auth->processResponse();      // Process the Response of the IdP, get the
                                   // attributes and put then at
                                   // $_SESSION['samlUserdata']

    $errors = $auth->getErrors();  // This method receives an array with the errors
                                   // that could took place during the process
```

```
    if (!empty($errors)) {
        print_r('<p>'.implode(', ', $errors).'</p>');
    }
                                          // This check if the response was
    if (!$auth->isAuthenticated()) {      // sucessfully validated and the user
        echo "<p>Not authenticated</p>";  // data retrieved or not
        exit();
    }

    $_SESSION['samlUserdata'] = $auth->getAttributes(); // Retrieves user data
    if (isset($_POST['RelayState']) && OneLogin_Saml2_Utils::getSelfURL() !=
$_POST['RelayState']) {
        $auth->redirectTo($_POST['RelayState']);  // Redirect if there is a
    }                                             // relayState set
} else if (isset($_GET['sls'])) {   // Single Logout Service
    $auth->processSLO();            // Process the Logout Request & Logout Response
    $errors = $auth->getErrors(); // Retrieves possible validation errors
    if (empty($errors)) {
        print_r('<p>Sucessfully logged out</p>');
    } else {
        print_r('<p>'.implode(', ', $errors).'</p>');
    }
}

if (isset($_SESSION['samlUserdata'])) {   // If there is user data we print it.
    if (!empty($_SESSION['samlUserdata'])) {
        $attributes = $_SESSION['samlUserdata'];
        echo 'You have the following attributes:<br>';
        echo '<table><thead><th>Name</th><th>Values</th></thead><tbody>';
        foreach ($attributes as $attributeName => $attributeValues) {
            echo '<tr><td>' . htmlentities($attributeName) . '</td><td><ul>';
            foreach ($attributeValues as $attributeValue) {
                echo '<li>' . htmlentities($attributeValue) . '</li>';
            }
            echo '</ul></td></tr>';
        }
        echo '</tbody></table>';
    } else {                              // If there is not user data, we notify
        echo "<p>You don't have any attribute</p>";
    }

    echo '<p><a href="?slo" >Logout</a></p>'; // Print some links with possible
} else {                                      // actions
    echo '<p><a href="?sso" >Login</a></p>';
    echo '<p><a href="?sso2" >Login and access to attrs.php page</a></p>';
}
```

## Main classes and methods

Described below are the main classes and methods that can be invoked.

### The Old Saml library

Lets start describing the classes and methods of the SAML library, an evolution of the old v.1 toolkit that is provided to keep the backward compability. Most of them use classes and methods of the

new SAML2 library.

#### OneLogin_Saml_AuthRequest - AuthRequest.php

Has the protected attribute $auth, an OneLogin_Saml2_Auth object.

- **OneLogin_Saml_AuthRequest**. Constructs the OneLogin_Saml2_Auth, initializing the SP SAML instance.
- **getRedirectUrl($returnTo)**. Obtains the SSO URL containing the AuthRequest message deflated.

#### OneLogin_Saml_Response - Response.php

- **OneLogin_Saml_Response**. Constructor that process the SAML Response, Internally initializes an SP SAML instance and an OneLogin_Saml2_Response.
- **get_saml_attributes**. Retrieves an Array with the logged user data.

#### OneLogin_Saml_Settings - Settings.php

A simple class used to build the Setting object used in the v1.0 of the toolkit.

#### OneLogin_Saml_Metadata - Metadata.php

- **OneLogin_Saml_Metadata** Constructor that build the Metadata XML info based on the settings of the SP
- **getXml** An XML with the metadata info of the SP

#### OneLogin_Saml_XmlSec - XmlSec.php

Auxiliary class that contains methos to validate the SAML Response: **validateNumAssertions**, **validateTimestamps**, **isValid** (whith uses the other 2 previous methods and also validate the signature of SAML Response).

### Saml2 library

Lets describe now the classes and methods of the SAML2 library.

#### OneLogin_Saml2_Auth - Auth.php

Main class of OneLogin PHP Toolkit

- **OneLogin_Saml2_Auth** Initializes the SP SAML instance
- **login** Initiates the SSO process.
- **logout** Initiates the SLO process.
- **processResponse** Process the SAML Response sent by the IdP.
- **processSLO** Process the SAML Logout Response / Logout Request sent by the IdP.
- **redirectTo** Redirects the user to the url past by parameter or to the url that we defined in our SSO Request.
- **isAuthenticated** Checks if the user is authenticated or not.
- **getAttributes** Returns the set of SAML attributes.

- **getAttribute** Returns the requested SAML attribute
- **getNameId** Returns the nameID
- **getErrors** Returns if there were any error
- **getSSOurl** Gets the SSO url.
- **getSLOurl** Gets the SLO url.
- **buildRequestSignature** Generates the Signature for a SAML Request
- **buildResponseSignature** Generates the Signature for a SAML Response
- **getSettings** Returns the settings info
- **setStrict** Set the strict mode active/disable

#### OneLogin_Saml2_Auth - AuthnRequest.php

SAML 2 Authentication Request class

- **OneLogin_Saml2_Auth** Constructs the AuthnRequest object.
- **getRequest** Returns deflated, base64 encoded, unsigned AuthnRequest.

#### OneLogin_Saml2_Response - Response.php

SAML 2 Authentication Response class

- **OneLogin_Saml2_Response** Constructs the SAML Response object.
- **isValid** Determines if the SAML Response is valid using the certificate.
- **checkStatus** Checks if the Status is success.
- **getAudiences** Gets the audiences.
- **getIssuers** Gets the Issuers (from Response and Assertion)
- **getNameIdData** Gets the NameID Data provided by the SAML response from the IdP.
- **getNameId** Gets the NameID provided by the SAML response from the IdP.
- **getSessionNotOnOrAfter** Gets the SessionNotOnOrAfter from the AuthnStatement
- **getSessionIndex** Gets the SessionIndex from the AuthnStatement.
- **getAttributes** Gets the Attributes from the AttributeStatement element.
- **validateNumAssertions** Verifies that the document only contains a single Assertion (encrypted or not).
- **validateTimestamps** Verifies that the document is still valid according Conditions Element.

#### OneLogin_Saml2_LogoutRequest - LogoutRequest.php

SAML 2 Logout Request class

- **OneLogin_Saml2_LogoutRequest** Constructs the Logout Request object.
- **getRequest** Returns the Logout Request defated, base64encoded, unsigned
- **getID** Returns the ID of the Logout Request.
- **getNameIdData** Gets the NameID Data of the the Logout Request.
- **getNameId** Gets the NameID of the Logout Request.
- **getIssuer** Gets the Issuer of the Logout Request.
- **getSessionIndexes** Gets the SessionIndexes from the Logout Request.

- **isValid** Checks if the Logout Request recieved is valid.

#### OneLogin_Saml2_LogoutResponse - LogoutResponse.php

SAML 2 Logout Response class

- **OneLogin_Saml2_LogoutResponse** Constructs a Logout Response object (Initialize params from settings and if provided load the Logout Response)
- **getIssuer** Gets the Issuer of the Logout Response.
- **getStatus** Gets the Status of the Logout Response.
- **isValid** Determines if the SAML LogoutResponse is valid
- **build** Generates a Logout Response object.
- **getResponse** Returns a Logout Response object.

#### OneLogin_Saml2_Settings - Settings.php

Configuration of the OneLogin PHP Toolkit

- **OneLogin_Saml2_Settings** Initializes the settings: Sets the paths of the different folders and Loads settings info from settings file or array/object provided
- **checkSettings** Checks the settings info.
- **getBasePath** Returns base path.
- **getCertPath** Returns cert path.
- **getLibPath** Returns lib path.
- **getExtLibPath** Returns external lib path.
- **getSchemasPath** Returns schema path.
- **checkSPCerts** Checks if the x509 certs of the SP exists and are valid.
- **getSPkey** Returns the x509 private key of the SP.
- **getSPcert** Returns the x509 public cert of the SP.
- **getIdPData** Gets the IdP data.
- **getSPData**Gets the SP data.
- **getSecurityData** Gets security data.
- **getContacts** Gets contact data.
- **getOrganization** Gets organization data.
- **getSPMetadata** Gets the SP metadata. The XML representation.
- **validateMetadata** Validates an XML SP Metadata.
- **formatIdPCert** Formats the IdP cert.
- **formatSPCert** Formats the SP cert.
- **formatSPKey** Formats the SP private key.
- **getErrors** Returns an array with the errors, the array is empty when the settings is ok.
- **setStrict** Activates or deactivates the strict mode.
- **isStrict** Returns if the 'strict' mode is active.
- **isDebugActive** Returns if the debug is active.

#### OneLogin_Saml2_Metadata - Metadata.php

A class that contains functionality related to the metadata of the SP

- **builder** Generates the metadata of the SP based on the settings.
- **signmetadata** Signs the metadata with the key/cert provided
- **addX509KeyDescriptors** Adds the x509 descriptors (sign/encriptation) to the metadata

#### OneLogin_Saml2_Utils - Utils.php

Auxiliary class that contains several methods

- **validateXML** This function attempts to validate an XML string against the specified schema.
- **formatCert** Returns a x509 cert (adding header & footer if required).
- **formatPrivateKey** returns a RSA private key (adding header & footer if required).
- **redirect** Executes a redirection to the provided url (or return the target url).
- **isHTTPS** Checks if https or http.
- **getSelfHost** Returns the current host.
- **getSelfURLhost** Returns the protocol + the current host + the port (if different than common ports).
- **getSelfURLNoQuery** Returns the URL of the current host + current view.
- **getSelfURL** Returns the URL of the current host + current view + query.
- **generateUniqueID** Generates an unique string (used for example as ID for assertions).
- **parseTime2SAML** Converts a UNIX timestamp to SAML2 timestamp on the form yyyy-mm-ddThh:mm:ss(.s+)?Z.
- **parseSAML2Time** Converts a SAML2 timestamp on the form yyyy-mm-ddThh:mm:ss(.s+)?Z to a UNIX timestamp. The sub-second part is ignored.
- **parseDuration** Interprets a ISO8601 duration value relative to a given timestamp.
- **getExpireTime** Compares 2 dates and returns the earliest.
- **query** Extracts nodes from the DOMDocument.
- **isSessionStarted** Checks if the session is started or not.
- **deleteLocalSession** Deletes the local session.
- **calculateX509Fingerprint** Calculates the fingerprint of a x509cert.
- **formatFingerPrint** Formates a fingerprint.
- **generateNameId** Generates a nameID.
- **getStatus** Gets Status from a Response.
- **decryptElement** Decrypts an encrypted element.
- **addSign** Adds signature key and senders certificate to an element (Message or Assertion).
- **validateSign** Validates a signature (Message or Assertion).

For more info, look at the source code; each method is documented and details about what does and how to use it are provided. Make sure to also check the doc folder where HTML documentation about the classes and methods is provided for SAML and SAML2.

## Demos included in the toolkit

The toolkit includes 3 demo apps to teach how use the toolkit, take a look on it.

Demos require that SP and IdP are well configured before test it.

# Demo1

## SP setup

The Onelogin's PHP Toolkit allows you to provide the settings info in 2 ways:

- Use a settings.php file that we should locate at the base folder of the toolkit.
- Use an array with the setting data.

In this demo we provide the data in the second way, using a setting array named $settingsInfo. This array users the settings_example.php included as a template to create the settings.php settings and store it in the demo1 folder. Configure the SP part and later review the metadata of the IdP and complete the IdP info.

If you check the code of the index.php file you will see that the settings.php file is loaded in order to get the $settingsInfo var to be used in order to initialize the Setting class.

Notice that in this demo, the setting.php file that could be defined at the base folder of the toolkit is ignored and the libs are loaded using the _toolkit_loader.php located at the base folder of the toolkit.

## IdP setup

Once the SP is configured, the metadata of the SP is published at the metadata.php file. After that, configure the IdP based on that information.

## How it works

1. First time you access to index.php view, you can select to login and return to the same view or login and be redirected to the attrs.php view.
2. When you click:2.1 in the first link, we access to (index.php?sso) an AuthNRequest is sent to the IdP, we authenticate at the IdP and then a Response is sent to the SP, specifically the Assertion Consumer Service view: index.php?acs, notice that a RelayState parameter is set to the url that initiated the process, the index.php view.2.2 in the second link we access to (attrs.php) have the same process described at 2.1 with the diference that as RelayState is set the attrs.php
    1. The SAML Response is processed in the ACS (index.php?acs), if the Response is not valid, the process stops here and a message is shown. Otherwise we are redirected to the RelayState view. a) index.php or b) attrs.php
    2. We are logged in the app and the user attributes are showed. At this point, we can test the single log out functionality.
    3. The single log out funcionality could be tested by 2 ways.
3. 5.1 SLO Initiated by SP. Click on the "logout" link at the SP, after that a Logout Request is sent to the IdP, the session at the IdP is closed and replies to the SP a Logout Response (sent to the Single Logout Service endpoint). The SLS endpoint (index.php?sls)of the SP process the Logout Response and if is valid, close the user session of the local app. Notice that the SLO Workflow starts and ends at the SP.5.2 SLO Initiated by IdP. In this case, the action takes place on the IdP side, the logout process is initiated at the idP, sends a Logout Request to the SP (SLS endpoint,

index.php?sls). The SLS endpoint of the SP process the Logout Request and if is valid, close the session of the user at the local app and send a Logout Response to the IdP (to the SLS endpoint of the IdP). The IdP receives the Logout Response, process it and close the session at of the IdP. Notice that the SLO Workflow starts and ends at the IdP.

Notice that all the SAML Requests and Responses are handled at a unique file, the index.php file and how GET paramters are used to know the action that must be done.

# Demo2

## SP setup

The Onelogin's PHP Toolkit allows you to provide the settings info in 2 ways:

- Use a settings.php file that we should locate at the base folder of the toolkit.
- Use an array with the setting data.

The first is the case of the demo2 app. The setting.php file and the setting_extended.php file should be defined at the base folder of the toolkit. Review the setting_example.php and the advanced_settings_example.php to learn how to build them.

In this case as Attribute Consume Service and Single Logout Service we are going to use the files located in the endpoint folder (acs.php and sls.php).

## IdP setup

Once the SP is configured, the metadata of the SP is published at the metadata.php file. Based on that info, configure the IdP.

## How it works

At demo1, we saw how all the SAML Request and Responses were handler at an unique file, the index.php file. This demo1 uses hight-level programming.

At demo2, we have several views: index.php, sso.php, slo.php, consume.php and metadata.php. As we said, we gonna use the endpoints that are defined in the toolkit (acs.php, sls.php of the endpoints folder). This demo2 uses low-level programming.

Notice that the SSO action can be initiated at index.php or sso.php.

The SAML workflow that take place is similar that the workflow defined in the demo1, only changes the targets.

1. When you access index.php or sso.php for the first time, an AuthNRequest is sent to the IdP automatically, (as RelayState is sent the origin url). We authenticate at the IdP and then a Response is sent to the SP, to the ACS endpoint, in this case acs.php of the endpoints folder.
2. The SAML Response is processed in the ACS, if the Response is not valid, the process stops here and a message is shown. Otherwise we are redirected to the RelayState view (sso.php or index.php). The sso.php detects if the user is logged and redirects to index.php, so we will be in

the index.php at the end.
3. We are logged into the app and the user attributes are shown. At this point, we can test the single log out functionality.
4. The single log out funcionality could be tested by 2 ways.4.1 SLO Initiated by SP. Click on the "logout" link at the SP, after that we are redirected to the slo.php view and there a Logout Request is sent to the IdP, the session at the IdP is closed and replies to the SP a Logout Response (sent to the Single Logout Service endpoint). In this case The SLS endpoint of the SP process the Logout Response and if is valid, close the user session of the local app. Notice that the SLO Workflow starts and ends at the SP.5.2 SLO Initiated by IdP. In this case, the action takes place on the IdP side, the logout process is initiated at the idP, sends a Logout Request to the SP (SLS endpoint sls.php of the endpoint folder). The SLS endpoint of the SP process the Logout Request and if is valid, close the session of the user at the local app and sends a Logout Response to the IdP (to the SLS endpoint of the IdP).The IdP receives the Logout Response, process it and close the session at of the IdP. Notice that the SLO Workflow starts and ends at the IdP.

# Demo Old

## SP setup

This demo uses the old style of the version 1 of the toolkit. An object of the class OneLogin_Saml_Settings must be provided to the constructor of the AuthRequest.

You will find an example_settings.php file at the demo-old's folder that could be used as a template for you settings.php file.

In that template, SAML settings are divided into two parts, the application specific (const_assertion_consumer_service_url, const_issuer, const_name_identifier_format) and the user/account specific idp_sso_target_url, x509certificate). You'll need to add your own code here to identify the user or user origin (e.g. by subdomain, ip_address etc.).

## IdP setup

Once the SP is configured, the metadata of the SP is published at the metadata.php file. After that, configure the IdP based on that information.

## How it works

At the metadata.php view is published the metadata of the SP.

The index.php file acts as an initiater for the SAML conversation if it should should be initiated by the application. This is called Service Provider Initiated SAML. The service provider creates a SAML Authentication Request and sends it to the identity provider (IdP).

The consume.php is the ACS endpoint. Receives the SAML assertion. After Response validation, the userdata and the nameID will be available, using getNameId() or getAttributes() we obtain them.

Since the version 1 of the php toolkit does not support SLO we don't show how handle SLO in this

demo-old.