# djangosaml2idp Documentation

***Release 0.7.2***

**Mathieu Hinderyckx**

**Mar 02, 2021**

# Contents:

CHAPTER 1

# djangosaml2idp

djangosaml2idp implements the Identity Provider side of the SAML2 protocol for Django. It builds on top of PySAML2, and is production-ready.

Any contributions, feature requests, proposals, ideas ... are welcome! See the CONTRIBUTING document for some info.

# CHAPTER 2

# Table of contents

## 2.1 Installation

PySAML2 uses XML Security Library binary to sign SAML assertions, so you need to install it either through your operating system package or by compiling the source code. It doesn't matter where the final executable is installed because you will need to set the full path to it in the configuration stage. XmlSec is available (at least) for Debian, OSX and Alpine Linux.

Now you can install the djangosaml2idp package using pip. This will also install PySAML2 and its dependencies automatically:

```
pip install djangosaml2idp
```

## 2.2 Configuration & Usage

The first thing you need to do is add `djangosaml2idp` to the list of installed apps:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'djangosaml2idp',
    ...
)
```

Now include `djangosaml2idp` in your project by adding it in the url config:

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^idp/', include('djangosaml2idp.urls')),
    url(r'^admin/', admin.site.urls),
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

Run the migrations for the app.

In your Django settings, configure your IdP. Configuration follows the PySAML2 configuration. The IdP from the example project looks like this:

```
import saml2
from saml2.saml import NAMEID_FORMAT_EMAILADDRESS, NAMEID_FORMAT_UNSPECIFIED
from saml2.sigver import get_xmlsec_binary

LOGIN_URL = '/login/'
BASE_URL = 'http://localhost:9000/idp'

SAML_IDP_CONFIG = {
    'debug' : DEBUG,
    'xmlsec_binary': get_xmlsec_binary(['/opt/local/bin', '/usr/bin/xmlsec1']),
    'entityid': '%s/metadata' % BASE_URL,
    'description': 'Example IdP setup',

    'service': {
        'idp': {
            'name': 'Django localhost IdP',
            'endpoints': {
                'single_sign_on_service': [
                    ('http://localhost:9000/idp/sso/post/', saml2.BINDING_HTTP_POST),
                    ('http://localhost:9000/idp/sso/redirect/', saml2.BINDING_HTTP_
↪REDIRECT),
                ],
                "single_logout_service": [
                    ("http://localhost:9000/idp/slo/post/", saml2.BINDING_HTTP_POST),
                    ("http://localhost:9000/idp/slo/redirect/", saml2.BINDING_HTTP_
↪REDIRECT)
                ],
            },
            'name_id_format': [NAMEID_FORMAT_EMAILADDRESS, NAMEID_FORMAT_UNSPECIFIED],
            'sign_response': True,
            'sign_assertion': True,
            'want_authn_requests_signed': True,
        },
    },

    # Signing
    'key_file': BASE_DIR + '/certificates/private.key',
    'cert_file': BASE_DIR + '/certificates/public.cert',
    # Encryption
    'encryption_keypairs': [{
        'key_file': BASE_DIR + '/certificates/private.key',
        'cert_file': BASE_DIR + '/certificates/public.cert',
    }],
    'valid_for': 365 * 24,
}
```

Notice the configuration requires a private key and public certificate to be available on the filesystem in order to sign and encrypt messages.

Next the Service Providers and their configuration need to be added, this is done via the Django admin interface. Add an entry for each SP which speaks to thie IdP. Add a copy of the local metadata xml, or set a remote metadata url. Add an attribute mapping for user attributes to SAML fields or leave the default mapping which will be prefilled.

Several attributes can be overriden per SP. If they aren't overridden explicitly, they will use the 'global' settings which can be configured for your Django installation. If those aren't set, some defaults will be used, as indicated in the admin when you configre a SP. The resulting configuration of a SP, with merged settings of its own and the instance settings and defaults, is shown in the admin as a summary.

## 2.3 Further optional configuration options

In the `SAML_IDP_SPCONFIG` setting you can define a `processor`, its value being a string with dotted path to a class. This is a hook to customize some access control checks. By default, the included *BaseProcessor* is used, which allows every user to login on the IdP. You can customize this behaviour by subclassing the *BaseProcessor* and overriding its *has_access(self, request)* method. This method should return true or false, depending if the user has permission to log in for the SP / IdP. The processor has the SP entity ID available as *self._entity_id*, and received the request (with an authenticated request.user on it) as parameter to the *has_access* function. This way, you should have the necessary flexibility to perform whatever checks you need. An example processor subclass can be found in the IdP of the included example. Use this metadata xml to configure your SP. Place the metadata xml from that SP in the location specified in the config dict (sp_metadata.xml in the example above).

Without custom setting, users will be identified by the `USERNAME_FIELD` property on the user Model you use. By Django defaults this will be the username. You can customize which field is used for the identifier by adding `SAML_IDP_DJANGO_USERNAME_FIELD` to your settings with as value the attribute to use on your user instance.

Other settings you can set as defaults to be used if not overriden by an SP are *SAML_AUTHN_SIGN_ALG*, *SAML_AUTHN_DIGEST_ALG*, and *SAML_ENCRYPT_AUTHN_RESPONSE*. They can be set if desired in the django settings, in which case they will be used for all ServiceProviders configuration on this instance if they don't override it. E.g.:

```
SAML_AUTHN_SIGN_ALG = saml2.xmldsig.SIG_RSA_SHA256
SAML_AUTHN_DIGEST_ALG = saml2.xmldsig.DIGEST_SHA256
```

In case your SP does not properly expose validuntil in metadata, you can provide fallback setting for it using:

```
SAML_IDP_FALLBACK_EXPIRATION_DAYS = 30
```

The default value for the fields `processor` and `attribute_mapping` can be set via the settings (the values displayed here are the defaults):

```
SAML_IDP_SP_FIELD_DEFAULT_PROCESSOR = 'djangosaml2idp.processors.BaseProcessor'
SAML_IDP_SP_FIELD_DEFAULT_ATTRIBUTE_MAPPING = {"email": "email", "first_
↪name", "last_name": "last_name", "is_staff": "is_staff", "is_superuser": "is_
↪superuser"}
```

## 2.4 Customizing error handling

djangosaml2idp renders a very basic error page if it encounters an error, indicating an error occured, which error, and possibly an extra message. The HTTP status code is dependant on which error occured. It also logs the exception with error severity. You can customize this by using the `SAML_IDP_ERROR_VIEW_CLASS` setting. Set this to a dotted import path to your custom (class based) view in order to use that one. You'll likely want this to use your own template and styling to display and error message. If you subclass the provided *djangosaml2idp.error_views.SamlIDPErrorView*, you have the following variables available for use in the template:

**exception** the exception instance that occurred

**exception_type** the class of the exception that occurred

**exception_msg** the message from the exception (by doing *str(exception)*)

**extra_message** if no specific exception given, a message indicating something went wrong, or an additional message next to the *exception_msg*

The simplest override is to subclass the *SamlIDPErrorView* and only using your own error template. You can use any Class-Based-View for this; it's not necessary to subclass the builtin error view. The example project contains a ready to use example of this; uncomment the *SAML_IDP_ERROR_VIEW_CLASS* setting and it will use a custom view with custom template.

## 2.5 Multi Factor Authentication support

There are three main components to adding multiple factor support.

1. Subclass djangosaml2idp.processors.BaseProcessor as outlined above. You will need to override the *enable_multifactor()* method to check whether or not multifactor should be enabled for a user. (If it should allways be enabled for all users simply hard code to True). By default it unconditionally returns False and no multifactor is enforced.
2. Sublass the *djangosaml2idp.views.ProcessMultiFactorView* view to make the appropriate calls for your environment. Implement your custom verification logic in the *multifactor_is_valid* method: this could call a helper script, an internal SMS triggering service, a data source only the IdP can access or an external second factor provider (e.g. Symantec VIP). By default this view will log that it was called then redirect.
3. Update your urls.py and add an override for name='saml_multi_factor' - ensure it comes before importing the djangosaml2idp urls file so your custom view is used instead of the built-in one.

## 2.6 Example SP/IdP implementation

This is a barebone example implementation of a functional setup with a Service Provider and Identity Provider. The SP is implemented using djangosaml2, the IdP using djangosaml2idp. Both are default django projects with only the bare minimum of added code for a functional demo. This is meant to easily highlight only the parts necessary to implement the SP/IdP functionality in your own without other code clutter serving as distraction.

A docker-compose file is included, providing a minimum-entry-barrier setup to get up and running, without complicated & error-prone setup requirements. The example will run equally on Mac/Windows/Linux using docker.

### 2.6.1 How to run

Go to this folder in a terminal and start the containers:

```
docker-compose up -d
```

Give it a minute the first time to download and build the required images. They'll be cached for successive runs. You now have a SP running at http://localhost:8000/ and a IdP at http://localhost:9000/ (you can check your containers using `docker-compose ps`), configured to talk with each other. In order to do an actual login, you will need to create a user account on the IdP. Run this in a terminal with your containers running to create a new user:

> docker exec -it djangosaml2idp_idp python manage.py createsuperuser

If you don't want to use docker, simply do in a terminal from the idp directory pip install -r requirements.txt

python manage.py migrate

python manage.py runserver 0.0.0.0:9000 (8000 for the SP) in a terminal

### 2.6.2 How to use

There are two flows illustrated with this demo:

1. **SP-initiated login**
    - Go to the SP in your browser and verify you are not logged in.
    - Click on the login link. You'll get redirected to a login form on the IdP instance.
    - Log in with your credentials from the user you just created. You get redirected back to the SP instance.
    - You are now logged in on the SP. The page shows the user information stemming from the IdP.
2. **IDP-initiated login**
    - Go to the SP in your browser and verify you are not logged in.
    - Go to the IDP in your browser. You are not logged in.
    - Click on the login link. You'll get redirected to a login form on the IdP instance.
    - Log in with your credentials from the user you just created. You get back to the IDP landing page, but now you are logged in.
    - Click the link on the bottom saying "Perform IDP-initiated login . . . ".
    - You get redirected to the SP and are now logged in.

At no point in the two steps below did you login on the SP; all logins happen on the IDP. The authentication information is then passed to the SP. And that is essentially what SSO with SAML2 does :)

### 2.6.3 Cleanup

To stop the containers:

```
docker-compose stop
```

### 2.6.4 Certificate generation

The provided self-signed certificates included in this example are valid until 2028. Should you need to regenerate them, you can use the *generate.sh* script to generate new ones. For e.g. the IdP certificate:

1. Ensure the subj has the correct hostname in the openssl command: */CN=idp.localhost.com*
2. Execute the script: *./generate.sh*. This will create a private.key & public.cert pair
3. Copy the newly create files to the idp/certificates folder
4. Copy the public.cert content without the first and last line into the sp/saml2_config/idp_metadata.xml tags *ns2:X509Certificate* (there are 2 of them)

For the SP certificates, the process is the same but the hostname is *sp.localhost.com* and in step 3 & 4 switch idp with sp folder.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search