djangosaml2

Release 0.2.4

Giuseppe De Marco

Aug 25, 2021

SETUP

A Django application that builds a fully compliant SAML2 Service Provider on top of PySAML2 library. Djangosaml2 protects your project with a SAML2 SSO Authentication, supporting features like **HTTP-REDIRECT** and **HTTP-POST SSO Binding**, **Single logout**, **Discovery Service**, **Wayf page** with customizable html template, **IdP Hinting**, **IdP Scoping** and **Samesite cookie** SSO workaround.

The entire project code is open sourced and therefore licensed under the Apache 2.0.

CHAPTER

ONE

SETUP

1.1 Prepare Environment and Install Requirements

PySAML2 uses xmlsec1 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.

Now you can install the djangosaml2 package using pip. This will also install PySAML2 and its dependencies automatically:

```
apt install python3-pip xmlsec python3-dev libssl-dev libsasl2-dev
pip3 install virtualenv
mkdir djangosaml2_project && cd "$_"
virtualenv -ppython3 env
source env/bin/activate
pip install djangosaml2
```

CHAPTER

TWO

CONFIGURATION

There are three things you need to setup to make djangosaml2 work in your Django project:

1. **settings.py** as you may already know, it is the main Django configuration file.
2. **urls.py** is the file where you will include djangosaml2 urls.
3. **pysaml2** specific files such as an attribute map directory and a certificates involved in SAML2 signature and encryption operations.

The first thing you need to do is add `djangosaml2` to the list of installed apps:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.admin',

    'djangosaml2',  # new application
)
```

2.1 SameSite cookie

Add the SAML Session Middleware as follow, this is needed for SameSite Cookies:

```
MIDDLEWARE.append('djangosaml2.middleware.SamlSessionMiddleware')
```

By default, djangosaml2 handle the saml2 session in a separate cookie. The storage linked to it is accessible by default at *request.saml_session*. You can even configure the SAML cookie name as follows:

```
SAML_SESSION_COOKIE_NAME = 'saml_session'
```

Remember that in your browser "SameSite=None" attribute MUST also have the "Secure" attribute, which is required in order to use "SameSite=None":

```
SESSION_COOKIE_SECURE = True
```

Note: djangosaml2 will attempt to set the `SameSite` attribute of the SAML session cookie to `None` so that it can be used in cross-site requests, but this is only possible with Django 3.1 or higher. If you are experiencing issues with

unsolicited requests or cookies not being sent (particularly when using the HTTP-POST binding), consider upgrading to Django 3.1 or higher. If you can't do that, configure "allow_unsolicited" to True in pySAML2 configuration.

2.2 Authentication backend

Then you have to add the `djangosaml2.backends.Saml2Backend` authentication backend to the list of authentications backends. By default only the ModelBackend included in Django is configured. A typical configuration would look like this:

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
    'djangosaml2.backends.Saml2Backend',
)
```

It is possible to subclass the provided Saml2Backend and customize the behaviour by overriding some methods. This way you can perform your custom cleaning or authorization policy, and modify the way users are looked up and created.

2.3 Default Login path

Finally we have to tell Django what the new login url we want to use is:

```
LOGIN_URL = '/saml2/login/'
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

Here we are telling Django that any view that requires an authenticated user should redirect the user browser to that url if the user has not been authenticated before. We are also telling that when the user closes his browser, the session should be terminated. This is useful in SAML2 federations where the logout protocol is not always available.

Note: The login url starts with `/saml2/` as an example but you can change that if you want. Check the section about changes in the `urls.py` file for more information.

If you want to allow several authentication mechanisms in your project you should set the LOGIN_URL option to another view and put a link in such view to djangosaml2 wb path, like `/saml2/login/`.

2.4 Handling Post-Login Redirects

It is often desireable for the client to maintain the URL state (or at least manage it) so that the URL once authentication has completed is consistent with the desired application state (such as retaining query parameters, etc.) By default, the HttpRequest objects get_host() method is used to determine the hostname of the server, and redirect URL's are allowed so long as the destination host matches the output of get_host(). However, in some cases it becomes desireable for additional hostnames to be used for the post-login redirect. In such cases, the setting:

```
SAML_ALLOWED_HOSTS = []
```

May be set to a list of allowed post-login redirect hostnames (note, the URL components beyond the hostname may be specified by the client - typically with the ?next= parameter.)

In the absence of a ?next=parameter, the ACS_DEFAULT_REDIRECT_URL or LOGIN_REDIRECT_URL setting will be used (assuming the destination hostname either matches the output of get_host() or is included in the SAML_ALLOWED_HOSTS setting)

2.5 Preferred sso binding

Use the following setting to choose your preferred binding for SP initiated sso requests:

```
SAML_DEFAULT_BINDING
```

For example:

```
import saml2
SAML_DEFAULT_BINDING = saml2.BINDING_HTTP_POST
```

2.6 Preferred Logout binding

Use the following setting to choose your preferred binding for SP initiated logout requests:

```
SAML_LOGOUT_REQUEST_PREFERRED_BINDING
```

For example:

```
import saml2
SAML_LOGOUT_REQUEST_PREFERRED_BINDING = saml2.BINDING_HTTP_POST
```

2.7 Ignore Logout errors

When logging out, a SAML IDP will return an error on invalid conditions, such as the IDP-side session being expired. Use the following setting to ignore these errors and perform a local Django logout nonetheless:

```
SAML_IGNORE_LOGOUT_ERRORS = True
```

2.8 Discovery Service

If you want to use a SAML Discovery Service, all you need is adding:

```
SAML2_DISCO_URL = 'https://your.ds.example.net/'
```

Of course, with the real URL of your preferred Discovery Service.

2.9 Idp hinting

If the SP uses an AIM Proxy it is possible to suggest the authentication IDP by adopting the *idphint* parameter. The name of the *idphint* parameter is default, but it can also be changed using this parameter:

```
SAML2_IDPHINT_PARAM = 'idphint'
```

This will ensure that the user will not get a possible discovery service page for the selection of the IdP to use for the SSO. When Djagosaml2 receives an HTTP request at the resource, web path, configured for the saml2 login, it will detect the presence of the *idphint* parameter. If this is present, the authentication request will report this URL parameter within the http request relating to the SAML2 SSO binding.

For example:

```
import requests
import urllib
idphint = {'idphint': [
             urllib.parse.quote_plus(b'https://that.idp.example.org/metadata'),
             urllib.parse.quote_plus(b'https://another.entitydi.org')]
          }
param = urllib.parse.urlencode(idphint)
# param is "idphint=%5B%27https%253A%252F%252Fthat.idp.example.org%252Fmetadata%27%2C+
↪%27https%253A%252F%252Fanother.entitydi.org%27%5D"
requests.get(f'http://djangosaml2.sp.fqdn.org/saml2/login/?{param}')
```

see AARC Blueprint specs here.

2.10 IdP scoping

The SP can suggest an IdP to a proxy by using the Scoping and IDPList elements in a SAML AuthnRequest. This is done using the *scoping* parameter to the login URL.

`https://sp.example.org/saml2/login/?scoping=https://idp.example.org`

This parameter can be combined with the IdP parameter if multiple IdPs are present in the metadata, otherwise the first is used.

`https://sp.example.org/saml2/login/?scoping=https://idp.example.org&idp=https://proxy.example.com/metadata`

Currently there is support for a single IDPEntry in the IDPList.

2.11 Authn Context

We can define the authentication context in settings.SAML_CONFIG[‘service’][‘sp’] as follows:

```
'requested_authn_context': {
    'authn_context_class_ref': [saml2.saml.AUTHN_PASSWORD_PROTECTED],
    'comparison': "exact"
}
```

2.12 Custom and dynamic configuration loading

By default, djangosaml2 reads the pysaml2 configuration options from the SAML_CONFIG setting but sometimes you want to read this information from another place, like a file or a database. Sometimes you even want this configuration to be different depending on the request.

Starting from djangosaml2 0.5.0 you can define your own configuration loader which is a callable that accepts a request parameter and returns a saml2.config.SPConfig object. In order to do so you set the following setting:

```
SAML_CONFIG_LOADER = 'python.path.to.your.callable'
```

2.13 Bearer Assertion Replay Attack Prevention

In SAML standard doc, section 4.1.4.5 it states

The service provider MUST ensure that bearer assertions are not replayed, by maintaining the set of used ID values for the length of time for which the assertion would be considered valid based on the NotOnOrAfter attribute in the <SubjectConfirmationData>

djangosaml2 provides a hook 'is_authorized' for the SP to store assertion IDs and implement replay prevention with your choice of storage.

```
def is_authorized(self, attributes: dict, attribute_mapping: dict, idp_entityid: str,␣
↪assertion: object, **kwargs) -> bool:
    if not assertion:
        return True

    # Get your choice of storage
    cache_storage = storage.get_cache()
    assertion_id = assertion.get('assertion_id')

    if cache.get(assertion_id):
        logger.warn("Received SAMLResponse assertion has been already used.")
        return False

    expiration_time = assertion.get('not_on_or_after')
    time_delta = isoparse(expiration_time) - datetime.now(timezone.utc)
    cache_storage.set(assertion_id, 'True', ex=time_delta)
    return True
```

CHAPTER

THREE

USERS, ATTRIBUTES AND ACCOUNT LINKING

In the SAML 2.0 authentication process the Identity Provider (IdP) will send a security assertion to the Service Provider (SP) upon a successful authentication. This assertion contains attributes about the user that was authenticated. It depends on the IdP configuration what exact attributes are sent to each SP it can talk to.

When such assertion is received on the Django side it is used to find a Django user and create a session for it. By default djangosaml2 will do a query on the User model with the USERNAME_FIELD attribute but you can change it to any other attribute of the User model. For example, you can do this lookup using the 'email' attribute. In order to do so you should set the following setting:

```
SAML_DJANGO_USER_MAIN_ATTRIBUTE = 'email'
```

Please, use an unique attribute when setting this option. Otherwise the authentication process may fail because djangosaml2 will not know which Django user it should pick.

If your main attribute is something inherently case-insensitive (such as an email address), you may set:

```
SAML_DJANGO_USER_MAIN_ATTRIBUTE_LOOKUP = '__iexact'
```

(This is simply appended to the main attribute name to form a Django query. Your main attribute must be unique even given this lookup.)

Another option is to use the SAML2 name id as the username by setting:

```
SAML_USE_NAME_ID_AS_USERNAME = True
```

You can configure djangosaml2 to create such user if it is not already in the Django database or maybe you don't want to allow users that are not in your database already. For this purpose there is another option you can set in the settings.py file:

```
SAML_CREATE_UNKNOWN_USER = True
```

This setting is True by default.

The following setting lets you specify a URL for redirection after a successful authentication:

```
ACS_DEFAULT_REDIRECT_URL = reverse_lazy('some_url_name')
```

Particularly useful when you only plan to use IdP initiated login and the IdP does not have a configured RelayState parameter. If not set Django's `LOGIN_REDIRECT_URL` or `/` will be used.

The other thing you will probably want to configure is the mapping of SAML2 user attributes to Django user attributes. By default only the User.username attribute is mapped but you can add more attributes or change that one. In order to do so you need to change the SAML_ATTRIBUTE_MAPPING option in your settings.py:

```
SAML_ATTRIBUTE_MAPPING = {
    'uid': ('username', ),
    'mail': ('email', ),
    'cn': ('first_name', ),
    'sn': ('last_name', ),
}
```

where the keys of this dictionary are SAML user attributes and the values are Django User attributes.

If you are using Django user profile objects to store extra attributes about your user you can add those attributes to the SAML_ATTRIBUTE_MAPPING dictionary. For each (key, value) pair, djangosaml2 will try to store the attribute in the User model if there is a matching field in that model. Otherwise it will try to do the same with your profile custom model. For multi-valued attributes only the first value is assigned to the destination field.

Alternatively, custom processing of attributes can be achieved by setting the value(s) in the SAML_ATTRIBUTE_MAPPING, to name(s) of method(s) defined on a custom django User object. In this case, each method is called by djangosaml2, passing the full list of attribute values extracted from the <saml:AttributeValue> elements of the <saml:Attribute>. Among other uses, this is a useful way to process multi-valued attributes such as lists of user group names.

For example:

Saml assertion snippet:

```
<saml:Attribute Name="groups" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
↪format:basic">
      <saml:AttributeValue>group1</saml:AttributeValue>
      <saml:AttributeValue>group2</saml:AttributeValue>
      <saml:AttributeValue>group3</saml:AttributeValue>
</saml:Attribute>
```

Custom User object:

```
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):

  def process_groups(self, groups):
    # process list of group names in argument 'groups'
    pass;
```

settings.py:

```
SAML_ATTRIBUTE_MAPPING = {
    'groups': ('process_groups', ),
}
```

Learn more about Django profile models at:

https://docs.djangoproject.com/en/dev/topics/auth/customizing/#substituting-a-custom-user-model

CHAPTER

FOUR

CUSTOM USER ATTRIBUTES PROCESSING

Sometimes you need to use special logic to update the user object depending on the SAML2 attributes and the mapping described above is simply not enough. For these cases djangosaml2 provides hooks that can be overriden with custom functionality.

First of all reference the modified Saml2Backend in settings.py file:

```
AUTHENTICATION_BACKENDS = [
    'your_package.authentication.ModifiedSaml2Backend',
]
```

For example:

```
from djangosaml2.backends import Saml2Backend


class ModifiedSaml2Backend(Saml2Backend):
    def save_user(self, user, *args, **kwargs):
        user.save()
        user_group = Group.objects.get(name='Default')
        user.groups.add(user_group)
        return super().save_user(user, *args, **kwargs)
```

Keep in mind save_user is only called when there was a reason to save the User model (ie. first login), and it has no access to SAML attributes for authorization. If this is required, it can be achieved by overriding the _update_user:

```
from djangosaml2.backends import Saml2Backend

class ModifiedSaml2Backend(Saml2Backend):
    def _update_user(self, user, attributes: dict, attribute_mapping: dict, force_save:␣
↪bool = False):
        if 'eduPersonEntitlement' in attributes:
            if 'some-entitlement' in attributes['eduPersonEntitlement']:
                user.is_staff = True
                force_save = True
            else:
                user.is_staff = False
                force_save = True
        return super()._update_user(user, attributes, attribute_mapping, force_save)
```

CHAPTER

FIVE

URLS

Changes in the urls.py file. The next thing you need to do is to include `djangosaml2.urls` module in your main `urls.py` module:

```
urlpatterns = patterns(
    '',
    #  lots of url definitions here

    (r'saml2/', include('djangosaml2.urls')),

    #  more url definitions
)
```

CHAPTER

SIX

PYSAML2 SPECIFIC FILES AND CONFIGURATION

Once you have finished configuring your Django project you have to start configuring PySAML2, please consult its official documentation before start. If you use just that library you have to put your configuration options in a file and initialize PySAML2 with the path to that file. In djangosaml2 you just put the same information in the Django settings.py file under the SAML_CONFIG option. We will see a typical configuration for protecting a Django project:

```
from os import path
import saml2
import saml2.saml
BASEDIR = path.dirname(path.abspath(__file__))

SAML_CONFIG = {
  # full path to the xmlsec1 binary programm
  'xmlsec_binary': '/usr/bin/xmlsec1',

  # your entity id, usually your subdomain plus the url to the metadata view
  'entityid': 'http://localhost:8000/saml2/metadata/',

  # directory with attribute mapping
  'attribute_map_dir': path.join(BASEDIR, 'attribute-maps'),

  # Permits to have attributes not configured in attribute-mappings
  # otherwise...without OID will be rejected
  'allow_unknown_attributes': True,

  # this block states what services we provide
  'service': {
      # we are just a lonely SP
      'sp' : {
          'name': 'Federated Django sample SP',
          'name_id_format': saml2.saml.NAMEID_FORMAT_TRANSIENT,

          # For Okta add signed logout requets. Enable this:
          # "logout_requests_signed": True,

          'endpoints': {
              # url and binding to the assetion consumer service view
              # do not change the binding or service name
              'assertion_consumer_service': [
                  ('http://localhost:8000/saml2/acs/',
                   saml2.BINDING_HTTP_POST),
```

(continues on next page)

(continued from previous page)

```
                ],
            # url and binding to the single logout service view
            # do not change the binding or service name
            'single_logout_service': [
                # Disable next two lines for HTTP_REDIRECT for IDP's that only support␣
↪HTTP_POST. Ex. Okta:
                ('http://localhost:8000/saml2/ls/',
                 saml2.BINDING_HTTP_REDIRECT),
                ('http://localhost:8000/saml2/ls/post',
                 saml2.BINDING_HTTP_POST),
                ],
            },

        'signing_algorithm':  saml2.xmldsig.SIG_RSA_SHA256,
        'digest_algorithm':  saml2.xmldsig.DIGEST_SHA256,

         # Mandates that the identity provider MUST authenticate the
         # presenter directly rather than rely on a previous security context.
        'force_authn': False,

         # Enable AllowCreate in NameIDPolicy.
        'name_id_format_allow_create': False,

         # attributes that this project need to identify a user
        'required_attributes': ['givenName',
                                'sn',
                                'mail'],

         # attributes that may be useful to have but not required
        'optional_attributes': ['eduPersonAffiliation'],

        'want_response_signed': True,
        'authn_requests_signed': True,
        'logout_requests_signed': True,
        # Indicates that Authentication Responses to this SP must
        # be signed. If set to True, the SP will not consume
        # any SAML Responses that are not signed.
        'want_assertions_signed': True,

        'only_use_keys_in_metadata': True,

        # When set to true, the SP will consume unsolicited SAML
        # Responses, i.e. SAML Responses for which it has not sent
        # a respective SAML Authentication Request.
        'allow_unsolicited': False,

        # in this section the list of IdPs we talk to are defined
        # This is not mandatory! All the IdP available in the metadata will be␣
↪considered instead.
        'idp': {
            # we do not need a WAYF service since there is
            # only an IdP defined here. This IdP should be
```

(continues on next page)

(continued from previous page)

```
            # present in our metadata

            # the keys of this dictionary are entity ids
            'https://localhost/simplesaml/saml2/idp/metadata.php': {
                'single_sign_on_service': {
                    saml2.BINDING_HTTP_REDIRECT: 'https://localhost/simplesaml/saml2/
↪idp/SSOService.php',
                    },
                'single_logout_service': {
                    saml2.BINDING_HTTP_REDIRECT: 'https://localhost/simplesaml/saml2/
↪idp/SingleLogoutService.php',
                    },
                },
            },
        },
    },

  # where the remote metadata is stored, local, remote or mdq server.
  # One metadatastore or many ...
  'metadata': {
      'local': [path.join(BASEDIR, 'remote_metadata.xml')],
      'remote': [{"url": "https://idp.testunical.it/idp/shibboleth"},],
      'mdq': [{"url": "https://ds.testunical.it",
               "cert": "certficates/others/ds.testunical.it.cert",
               }]
      },

  # set to 1 to output debugging information
  'debug': 1,

  # Signing
  'key_file': path.join(BASEDIR, 'private.key'),  # private part
  'cert_file': path.join(BASEDIR, 'public.pem'),  # public part

  # Encryption
  'encryption_keypairs': [{
      'key_file': path.join(BASEDIR, 'private.key'),  # private part
      'cert_file': path.join(BASEDIR, 'public.pem'),  # public part
  }],

  # own metadata settings
  'contact_person': [
      {'given_name': 'Lorenzo',
       'sur_name': 'Gil',
       'company': 'Yaco Sistemas',
       'email_address': 'lgs@yaco.es',
       'contact_type': 'technical'},
      {'given_name': 'Angel',
       'sur_name': 'Fernandez',
       'company': 'Yaco Sistemas',
       'email_address': 'angel@yaco.es',
       'contact_type': 'administrative'},
```

(continues on next page)

(continued from previous page)

```
    ],
  # you can set multilanguage information here
  'organization': {
      'name': [('Yaco Sistemas', 'es'), ('Yaco Systems', 'en')],
      'display_name': [('Yaco', 'es'), ('Yaco', 'en')],
      'url': [('http://www.yaco.es', 'es'), ('http://www.yaco.com', 'en')],
      },
  }
```

Note: Please check the PySAML2 documentation for more information about these and other configuration options.

There are several external files and directories you have to create according to this configuration.

The xmlsec1 binary was mentioned in the installation section. Here, in the configuration part you just need to put the full path to xmlsec1 so PySAML2 can call it as it needs.

6.1 Signed Logout Request

Idp's like Okta require a signed logout response to validate and logout a user. Here's a sample config with all required SP/IDP settings:

```
"logout_requests_signed": True,
```

6.2 Attribute Map

The `attribute_map_dir` points to a directory with attribute mappings that are used to translate user attribute names from several standards. It's usually safe to just copy the default PySAML2 attribute maps that you can find in the `tests/attributemaps` directory of the source distribution.

6.3 Metadata

The `metadata` option is a dictionary where you can define several types of metadata for remote entities. Usually the easiest type is the `local` where you just put the name of a local XML file with the contents of the remote entities metadata. This XML file should be in the SAML2 metadata format.

Note: Don't use `remote` option for fetching metadata in production. Try to use `mdq` and introduce a MDQ server instead, it's more efficient.

6.4 Certificates

The `key_file` and `cert_file` options reference the two parts of a standard x509 certificate. You need it to sign your metadata. For assertion encryption/decryption support please configure another set of `key_file` and `cert_file`, but as inner attributes of `encryption_keypairs` option.

Note: Check your openssl documentation to generate a certificate suitable for SAML2 operations.

SAML2 certificate creation example:

```
openssl req -nodes -new -x509 -newkey rsa:2048 -days 3650 -keyout private.key -out␣
↪public.cert
```

PySAML2 certificates are files, in the form of strings that contains a filesystem path. What about configuring the certificates in a different way, in case we are using a container based deploy?

- You could supply the cert & key as environment variables (base64 encoded) then create the files when the container starts, either in an entry point shell script or in your settings.py file.
- Using Python Tempfile In the settings create two temp files, then write the content configured in environment variables in them, then use tmpfile.name as key/cert values in pysaml2 configuration.

CHAPTER

SEVEN

GETTING STARTED

Prepare Database and Preload example data

```
./manage.py migrate
./manage.py createsuperuser
./manage.py runserver
```

CHAPTER

EIGHT

TEST IDP

Congratulations, you have finished configuring the SP side of the federation. Now you need to send the entity id and the metadata of this new SP to the IdP administrators so they can add it to their list of trusted services.

You can get this information starting your Django development server and going to the **http://localhost:8000/saml2/metadata/** url. If you have included the djangosaml2 urls under a different url prefix you need to correct this url.

There are many saml2 idps suitable for testing, such as samltest.id. If you are looking for a django IdP, you can try uniAuth or djangosaml2idp.

CHAPTER

NINE

TESTING

One way to check if everything is working as expected is to enable the following url:

```
urlpatterns = patterns(
    '',
    #  lots of url definitions here

    (r'saml2/', include('djangosaml2.urls')),
    (r'test/', 'djangosaml2.views.EchoAttributesView.as_view()'),

    #  more url definitions
)
```

Now if you go to the /test/ url you will see your SAML attributes and also a link to do a global logout.

CHAPTER

TEN

UNIT TESTS

Djangosaml2 have a legacy way to do tests, using an example project in *tests* directory. This means that to run tests you have to clone the repository, then install djangosaml2, then run tests using the example project.

example:

```
pip install -r requirements-dev.txt
# or
pip install djangosaml2[test]
```

then:: cd tests ./manage.py migrate ./manage.py test djangosaml2

If you have tox installed you can simply call *tox* inside the root directory and it will run the tests in multiple versions of Python.

CHAPTER

ELEVEN

CODE COVERAGE

example:

```
cd tests/
coverage erase
coverage run ./manage.py test djangosaml2 testprofiles
coverage report -m
```

CHAPTER

TWELVE

CUSTOM ERROR HANDLER

When an error occurs during the authentication flow, djangosaml2 will render a simple error page with an error message and status code. You can customize this behaviour by specifying the path to your own error handler in the settings:

```
SAML_ACS_FAILURE_RESPONSE_FUNCTION = 'python.path.to.your.view'
```

This should be a view which takes a request, optional exception which occured and status code, and returns a response to serve the user. E.g. The default implementation looks like this:

```
def template_failure(request, exception=None, status=403, **kwargs):
    """ Renders a simple template with an error message. """
    return render(request, 'djangosaml2/login_error.html', {'exception': exception},␣
↪status=status)
```

CHAPTER

THIRTEEN

CONTRIBUTING

Please open Issues to start debate regarding the requested features, or the patch that you would apply. We do not use a strict submission format, please try to be more concise as possibile.

The Pull Request MUST be done on the dev branch, please don't push code directly on the master branch.

CHAPTER

FOURTEEN

SIMPLESAMLPHP ISSUES

As of SimpleSAMLphp 1.8.2 there is a problem if you specify attributes in the SP configuration. When the SimpleSAMLphp metadata parser converts the XML into its custom php format it puts the following option:

```
'attributes.NameFormat' => 'urn:oasis:names:tc:SAML:2.0:attrname-format:uri'
```

But it need to be replaced by this one:

```
'AttributeNameFormat' => 'urn:oasis:names:tc:SAML:2.0:attrname-format:uri'
```

Otherwise the Assertions sent from the IdP to the SP will have a wrong Attribute Name Format and pysaml2 will be confused.

Furthermore if you have a AttributeLimit filter in your SimpleSAMLphp configuration you will need to enable another attribute filter just before to make sure that the AttributeLimit does not remove the attributes from the authentication source. The filter you need to add is an AttributeMap filter like this:

```
10 => array(
          'class' => 'core:AttributeMap', 'name2oid'
     ),
```

CHAPTER

FIFTEEN

OKTA FEDERATION

Okta settings to configure on your Idp's SAML app advanced settings:

```
Single Logout URL: http://localhost:8000/saml2/ls/post/
SP Issuer : http://localhost:8000/saml2/metadata/
```

Okta sample configuration for setting up an Okta SSO with Django:

```
'service': {
# we are just a lonely SP
'sp': {
    'name': 'XXX',
    'allow_unsolicited': True,
    'want_assertions_signed': True,  # assertion signing (default=True)
    'want_response_signed': True,
    "want_assertions_or_response_signed": True,  # is response signing required
    'name_id_format': NAMEID_FORMAT_UNSPECIFIED,

    # Must for signed logout requests
    "logout_requests_signed": True,
    'endpoints': {
        # url and binding to the assetion consumer service view
        # do not change the binding or service name
        'assertion_consumer_service': [
            ('http://localhost:8000/saml2/acs/',
             saml2.BINDING_HTTP_POST),
        ],
        # url and binding to the single logout service view
        # do not change the binding or service name
        'single_logout_service': [
            # ('http://localhost:8000/saml2/ls/',
            #  saml2.BINDING_HTTP_REDIRECT),
            ('http://localhost:8000/saml2/ls/post/',
             saml2.BINDING_HTTP_POST),
        ],
    },
    # Mandates that the identity provider MUST authenticate the
    # presenter directly rather than rely on a previous security context.
    'force_authn': False,

    "allow_unsolicited": True,
```

(continues on next page)

(continued from previous page)

```
    # Enable AllowCreate in NameIDPolicy.
    'name_id_format_allow_create': False,

    # attributes that this project need to identify a user
    'required_attributes': ['email'],

    # in this section the list of IdPs we talk to are defined
    'idp': {
        # we do not need a WAYF service since there is
        # only an IdP defined here. This IdP should be
        # present in our metadata

        # the keys of this dictionary are entity ids
        'https://xxx.okta.com/app/XXXXXXXXX/sso/saml/metadata': {
            # Okta only uses HTTP_POST disable this
            # 'single_sign_on_service': {
            #     saml2.BINDING_HTTP_REDIRECT: 'https://xxx.okta.com/app/APPNAME/
↪xxxxxxxxx/sso/saml',
            # },
            'single_logout_service': {
                saml2.BINDING_HTTP_POST: 'https://xxx.okta.com/app/APPNAME/xxxxxxxxxx/
↪slo/saml',
            },
        },
    },

 },
},
```

CHAPTER

SIXTEEN

FAQ

Why can't SAML be implemented as an Django Authentication Backend?

well SAML authentication is not that simple as a set of credentials you can put on a login form and get a response back. Actually the user password is not given to the service provider at all. This is by design. You have to delegate the task of authentication to the IdP and then get an asynchronous response from it.

Given said that, djangosaml2 does use a Django Authentication Backend to transform the SAML assertion about the user into a Django user object.

Why not put everything in a Django middleware class and make our lifes easier?

Yes, that was an option I did evaluate but at the end the current design won. In my opinion putting this logic into a middleware has the advantage of making it easier to configure but has a couple of disadvantages: first, the middleware would need to check if the request path is one of the SAML endpoints for every request. Second, it would be too magical and in case of a problem, much harder to debug.

Why not call this package django-saml as many other Django applications?

Following that pattern then I should import the application with import saml but unfortunately that module name is already used in pysaml2.

saml2.response.UnsolicitedResponse: Unsolicited response

If you are experiencing issues with unsolicited requests this is due to the fact that cookies not being sent when using the HTTP-POST binding. You have to configure samesite djangosaml2 middleware (see setup documentation) and also consider upgrading to Django 3.1 or higher. If you can't do that, configure "allow_unsolicited" to True in pySAML2 configuration.