pysaml2 Documentation

Roland Hedberg

Sep 23, 2022

Contents

SAML 2.0 or Security Assertion Markup Language 2.0 is a version of the SAML standard for exchanging authentication and authorization data between security domains.

CHAPTER 1

About PySAML2

PySAML2 is a pure python implementation of SAML2. It contains all necessary pieces for building a SAML2 service provider or an identity provider. The distribution contains examples of both. Originally written to work in a WSGI environment, there are extensions that allow you to use it with other frameworks.

CHAPTER 2

How to use PySAML2

Before you can use Pysaml2, you'll need to get it installed. If you have not done it yet, read the *Quick install guide*

Well, now you have it installed, and you want to do something.

And I'm sorry to tell you this, but there isn't really a lot you can do with this code on it's own.

Sure you can send a AuthenticationRequest to an IdentityProvider or an AttributeQuery to an AttributeAuthority but in order to get what they return you have to sit behind a Web server. Well, that is not really true since the AttributeQuery would be over SOAP and you would get the result over the connection you have to the AttributeAuthority.

But anyway, you may get my point. This is middleware stuff!

PySAML2 is built to fit into a WSGI application

But it can be used in a non-WSGI environment too.

So you will find descriptions of both cases here.

The configuration is the same disregarding whether you are using PySAML2 in a WSGI or non-WSGI environment.

2.1 Python compatibility

PySAML2 has transitioned to Python3. Master Apps is maintaining a fork with Python2 compatibility on [GitHub](https://github.com/masterapps-au/pysaml2).

CHAPTER 3

Table of contents

3.1 Quick install guide

Before you can use PySAML2, you'll need to get it installed. This guide will guide you to a simple, minimal installation.

3.1.1 Install PySAML2

For all this to work, you need to have Python installed. The development has been done using 2.7. There is now a 3.X version.

Prerequisites

You have to have ElementTree, which is either part of your Python distribution if it's recent enough, or if the Python is too old you have to install it, for instance by getting it from the Python Package Instance by using easy_install.

You also need xmlsec1 which you can download from http://www.aleksey.com/xmlsec/

If you're on macOS, you can get xmlsec1 installed from MacPorts or Fink.

If you're on rhel/centos 7 you will need to install xmlsec1 and xmlsec1-openssl:

```
yum install xmlsec1 xmlsec1-openssl
```

Depending on how you are going to use PySAML2 you might also need

- Mako
- pyASN1
- repoze.who
- python-memcache
- memcached

Quick build instructions

Once you have installed all the necessary prerequisites a simple:

```
python setup.py install
```

will install the basic code.

Note for rhel/centos 6: cffi depends on libffi-devel, and cryptography on openssl-devel to compile So you might want first to do: yum install libffi-devel openssl-devel

After this, you ought to be able to run the tests without a hitch. The tests are based on the pypy test environment, so:

```
cd tests
pip install -r test-requirements.txt
pytest
```

is what you should use. If you don't have py.test, get it it's part of pypy! It's really good!

3.2 Quick pysaml2 example

Release 7.2.1

Date Sep 23, 2022

In order to confirm that pysaml2 has been installed correctly and are ready to use you could run this basic example

Contents:

3.2.1 An extremely simple example of a SAML2 service provider

How it works

A SP works with authentication and possibly attribute aggregation. Both of these functions can be seen as parts of the normal Repoze.who setup. Namely the Challenger, Identifier and MetadataProvider parts.

Normal for Repoze.who Identifier and MetadataProvider plugins are that they place information in environment variables. The convention is to place identity information in environ["repoze.who.identity"]. This is a dictionary with keys like 'login', and 'repoze.who.userid'.

The SP follows this pattern and places the information gathered from the IdP that handled the authentication and possible extra information received from attribute authorities in the above mentioned dictionary under the key 'user'.

So in environ["repoze.who.identity"] you will find a dictionary with attributes and values, the attribute names used depends on what's returned from the IdP/AA. If there exists both a name and a friendly name, for instance, the friendly name is used as the key.

Setup

sp-wsgi:

- Go to the folder and copy the example files:

```
cd [your path]/pysaml2/example/sp-wsgi
cp service_conf.py.example service_conf.py
cp sp_conf.py.example sp_conf.py
```

sp_conf.py is configured to run on localhost on port 8087. If you want to you could make the necessary changes before proceeding to the next step.

- In order to generate the metadata file open a terminal:

```
cd [your path]/pysaml2/example/sp-wsgi
make_metadata.py sp_conf.py > sp.xml
```

sp-repoze:

- Go to the folder:

[your path]/pysaml2/example/sp-repoze

- Take the file named sp_conf.py.example and rename it sp_conf.py

sp_conf.py is configured to run on localhost on port 8087. If you want to you could make the necessary changes before proceeding to the next step.

- In order to generate the metadata file open a terminal:

```
cd [your path]/pysaml2/example/sp-repoze
make_metadata.py sp_conf.py > sp.xml
```

Important files:

sp_conf.py The SPs configuration

who.ini The repoze.who configuration file

Inside the folder named pki there are two files with certificates, mykey.pem with the private certificate and mycert.pem with the public part.

I'll go through these step by step.

sp_conf.py

The configuration is written as described in *Configuration of PySAML2 entities*. It means among other things that it's easily testable as to the correct syntax.

You can see the whole file in example/sp/sp_conf.py, here I will go through it line by line:

```
"service": ["sp"],
```

Tells the software what type of services the software is supposed to supply. It is used to check for the completeness of the configuration and also when constructing metadata from the configuration. More about that later. Allowed values are: "sp" (service provider), "idp" (identity provider) and "aa" (attribute authority).

```
"entityid" : "urn:mace:example.com:saml:sp",
"service_url" : "http://example.com:8087/",
```

The ID of the entity and the URL on which it is listening.:

```
"idp_url" : "https://example.com/saml2/idp/SSOService.php",
```

Since this is a very simple SP it only needs to know about one IdP, therefore there is really no need for a metadata file or a WAYF-function or anything like that. It needs the URL of the IdP and that's all.:

```
"my_name" : "My first SP",
```

This is just for informal purposes, not really needed but nice to do:

```
"debug" : 1,
```

Well, at this point in time you'd really like to have as much information as possible as to what's going on, right ?

```
"key_file" : "./mykey.pem",
"cert_file" : "./mycert.pem",
```

The necessary certificates.:

```
"xmlsec_binary" : "/opt/local/bin/xmlsec1",
```

Right now the software is built to use xmlsec binaries and not the python xmlsec package. There are reasons for this but I won't go into them here.:

```
"organization": {
    "name": "Example Co",
    #display_name
    "url":"http://www.example.com/",
},
```

Information about the organization that is behind this SP, only used when building metadata.

```
"contact": [{
    "given_name":"John",
    "sur_name": "Smith",
    "email_address": "john.smith@example.com",
    #contact_type
    #company
    #telephone_number
}]
```

Another piece of information that only matters if you build and distribute metadata.

So, now to that part. In order to allow the IdP to talk to you, you may have to provide the one running the IdP with a metadata file. If you have a SP configuration file similar to the one I've walked you through here, but with your information, you can make the metadata file by running the make_metadata script you can find in the tools directory.

Change directory to where you have the configuration file and do

```
make_metadata.py sp_conf.py > metadata.xml
```

who.ini

The file named `who.ini` is the `sp-repoze` folder

I'm not going through the INI file format here. You should read Middleware Responsibilities to get a good introduction to the concept.

The configuration of the pysaml2 part in the applications middleware are first the special module configuration, namely:

```
[plugin:saml2auth]
use = s2repoze.plugins.sp:make_plugin
saml_conf = sp_conf.py
rememberer_name = auth_tkt
```

(continues on next page)

(continued from previous page)

```
debug = 1
path_logout = .*/logout.*
```

Which contains a specification ("use") of which function in which module should be used to initialize the part. After that comes the name of the file ("saml_conf") that contains the PySaml2 configuration. The third line ("rememberer_name") points at the plugin that should be used to remember the user information.

After this, the plugin is referenced in a couple of places:

```
[identifiers]
plugins =
      saml2auth
      auth_tkt

[authenticators]
plugins = saml2auth

[challengers]
plugins = saml2auth

[mdproviders]
plugins = saml2auth
```

Which means that the plugin is used in all phases.

Run SP:

Open a Terminal:

```
cd [your path]/pysaml2/example/sp-wsgi
python sp.py sp_conf
```

Note that you should not have the .py extension on the sp_conf.py while running the program

Now you should be able to open a web browser and go to to service provider (if you didn't change sp_conf.py it should be: http://localhost:8087)

You should be redirected to the IDP and presented with a login screen.

You could enter Username:roland and Password:dianakra All users are specified in idp.py in a dictionary named PASSWD

The application

The app is, as said before, extremely simple. The only thing that is connected to the PySaml2 configuration is at the bottom, namely where the server is. You have to ascertain that this coincides with what is specified in the PySaml2 configuration. Apart from that there really is nothing in application.py that demands that you use PySaml2 as middleware. If you switched to using the LDAP or CAS plugins nothing would change in the application. In the application configuration yes! But not in the application. And that is really how it should be done.

There is one assumption, and that is that the middleware plugin that gathers information about the user places the extra information in as a value on the "user" property in the dictionary found under the key "repoze.who.identity" in the environment.

3.2.2 An extremly simple example of a SAML2 identity provider.

There are 2 example IDPs in the project's example directory:

- idp2 has a static definition of users:
  - user attributes are defined in idp_user.py
  - the password is defined in the PASSWD dict in idp.py
- idp2_repoze is using repoze.who middleware to perform authentication and attribute retrieval

Configuration

Entity configuration is described in "Configuration of pysaml2 entities" Server parameters like host and port and various command line parameters are defined in the main part of idp.py

Setup:

The folder [your path]/pysaml2/example/idp2 contains a file named idp_conf.py.example

Take the file named idp_conf.py.example and rename it idp_conf.py

Generate a metadata file based in the configuration file (idp_conf.py) by using the command:

```
make_metadata.py idp_conf.py > idp.xml
```

Run IDP:

Open a Terminal:

```
cd [your path]/pysaml2/example/idp2
python idp.py idp_conf
```

Note that you should not have the .py extension on the idp_conf.py while running the program

3.3 How to use PySAML2

Release

Date Sep 23, 2022

Before you can use Pysaml2, you'll need to get it installed. If you have not done it yet, read the *Quick install guide*

Well, now you have it installed and you want to do something.

And I'm sorry to tell you this; but there isn't really a lot you can do with this code on its own.

Sure you can send a AuthenticationRequest to an IdentityProvider or a AttributeQuery to an AttributeAuthority, but in order to get what they return you have to sit behind a Web server. Well that is not really true since the AttributeQuery would be over SOAP and you would get the result over the connection you have to the AttributeAuthority.

But anyway, you may get my point. This is middleware stuff!

PySAML2 is built to fit into a WSGI application

But it can be used in a non-WSGI environment too.

So you will find descriptions of both cases here.

The configuration is the same regardless of whether you are using PySAML2 in a WSGI or non-WSGI environment.

3.3.1 Configuration of PySAML2 entities

Whether you plan to run a PySAML2 Service Provider, Identity Provider or an attribute authority, you have to configure it. The format of the configuration file is the same regardless of which type of service you plan to run. What differs are some of the directives. Below you will find a list of all the used directives in alphabetical order. The configuration is written as a python module which contains a named dictionary ("CONFIG") that contains the configuration directives.

The basic structure of the configuration file is therefore like this:

```
from saml2 import BINDING_HTTP_REDIRECT

CONFIG = {
    "entityid": "http://saml.example.com:saml/idp.xml",
    "name": "Rolands IdP",
    "service": {
        "idp": {
            "endpoints": {
                "single_sign_on_service": [
                    (
                        "http://saml.example.com:saml:8088/sso",
                        BINDING_HTTP_REDIRECT,
                    ),
                ],
                "single_logout_service": [
                    (
                        "http://saml.example.com:saml:8088/slo",
                        BINDING_HTTP_REDIRECT,
                    ),
                ],
            },
            ...
        }
    },
    "key_file": "my.key",
    "cert_file": "ca.pem",
    "xmlsec_binary": "/usr/local/bin/xmlsec1",
    "delete_tmpfiles": True,
    "metadata": {
        "local": [
            "edugain.xml",
        ],
    },
    "attribute_map_dir": "attributemaps",
    ...
}
```

Note: You can build the metadata file for your services directly from the configuration. The make_metadata.py script in the PySAML2 tools directory will do that for you.

Configuration directives

General directives

logging

The logging configuration format is the python logging format. The configuration is passed to the python logging dictionary configuration handler, directly.

Example:

```
"logging": {
    "version": 1,
    "formatters": {
        "simple": {
            "format": "[%(asctime)s] [%(levelname)s] [%(name)s.%(funcName)s]
↪%(message)s",
        },
    },
    "handlers": {
        "stdout": {
            "class": "logging.StreamHandler",
            "stream": "ext://sys.stdout",
            "level": "DEBUG",
            "formatter": "simple",
        },
    },
    "loggers": {
        "saml2": {
            "level": "DEBUG"
        },
    },
    "root": {
        "level": "DEBUG",
```

(continues on next page)

(continued from previous page)

```
        "handlers": [
            "stdout",
        ],
    },
},
```

The example configuration above will enable DEBUG logging to stdout.

debug

Example:

```
debug: 1
```

Whether debug information should be sent to the log file.

http_client_timeout

Example:

```
http_client_timeout: 10
```

The timeout of HTTP requests, in seconds. Defaults to None.

additional_cert_files

Example:

```
additional_cert_files: ["other-cert.pem", "another-cert.pem"]
```

Additional public certs that will be listed. Useful during cert/key rotation or if you need to include a certificate chain.

Each entry in *additional_cert_files* must be a PEM formatted file with a single certificate.

entity_attributes

Generates an `Attribute` element with the given NameFormat, Name, FriendlyName and values, each as an `AttributeValue` element.

The element is added under the generated metadata `EntityDescriptor` as an `Extension` element under the `EntityAttributes` element.

And omit

Example:

```
"entity_attributes": [
  {
    "name_format": "urn:oasis:names:tc:SAML:2.0:attrname-format:uri",
    "name": "urn:oasis:names:tc:SAML:profiles:subject-id:req",
    # "friendly_name" is not set
    "values": ["any"],
```

(continues on next page)

(continued from previous page)

```
  },
]
```

assurance_certification

Example:

```
"assurance_certification": [
    "https://refeds.org/sirtfi",
]
```

Generates an `Attribute` element with name-format `urn:oasis:names:tc:SAML:2.0:attrname-format:uri` and name `urn:oasis:names:tc:SAML:attribute:assurance-certification` that contains `AttributeValue` elements with the given values from the list. The element is added under the generated metadata `EntityDescriptor` as an `Extension` element under the `EntityAttributes` element.

Read more about representing assurance information at the specification.

attribute_map_dir

Points to a directory which has the attribute maps in Python modules.

Example:

```
"attribute_map_dir": "attribute-maps"
```

A typical map file will look like this:

```
MAP = {
    "identifier": "urn:oasis:names:tc:SAML:2.0:attrname-format:basic",
    "fro": {
        'urn:mace:dir:attribute-def:aRecord': 'aRecord',
        'urn:mace:dir:attribute-def:aliasedEntryName': 'aliasedEntryName',
        'urn:mace:dir:attribute-def:aliasedObjectName': 'aliasedObjectName',
        'urn:mace:dir:attribute-def:associatedDomain': 'associatedDomain',
        'urn:mace:dir:attribute-def:associatedName': 'associatedName',
        ...
    },
    "to": {
        'aRecord': 'urn:mace:dir:attribute-def:aRecord',
        'aliasedEntryName': 'urn:mace:dir:attribute-def:aliasedEntryName',
        'aliasedObjectName': 'urn:mace:dir:attribute-def:aliasedObjectName',
        'associatedDomain': 'urn:mace:dir:attribute-def:associatedDomain',
        'associatedName': 'urn:mace:dir:attribute-def:associatedName',
        ...
    }
}
```

The attribute map module contains a MAP dictionary with three items. The *identifier* item is the name-format you expect to support. The *to* and *fro* sub-dictionaries then contain the mapping between the names.

As you see the format is again a python dictionary where the key is the name to convert from, and the value is the name to convert to.

Since *to* in most cases is the inverse of the *fro* file, the software allows you only to specify one of them, and it will automatically create the other.

contact_person

This is only used by *make_metadata.py* when it constructs the metadata for the service described by the configuration file. This is where you describe who can be contacted if questions arise about the service or if support is needed. The possible types are according to the standard **technical**, **support**, **administrative**, **billing** and **other**.:

```
contact_person: [
    {
        "givenname": "Derek",
        "surname": "Jeter",
        "company": "Example Co.",
        "mail": ["jeter@example.com"],
        "type": "technical",
    },
    {
        "givenname": "Joe",
        "surname": "Girardi",
        "company": "Example Co.",
        "mail": "girardi@example.com",
        "type": "administrative",
    },
]
```

entityid

Example:

```
entityid: "http://saml.example.com/sp"
```

The globally unique identifier of the entity.

Note: It is recommended that the entityid should point to a real webpage where the metadata for the entity can be found.

name

A string value that sets the name of the PySAML2 entity.

Example:

```
"name": "Example IdP"
```

description

A string value that sets the description of the PySAML2 entity.

Example:

```
"description": "My IdP",
```

verify_ssl_cert

Specifies if the SSL certificates should be verified. Can be `True` or `False`. The default configuration is `False`.

Example:

```
"verify_ssl_cert": True
```

key_file

Example:

```
key_file: "key.pem"
```

key_file is the name of a PEM formatted file that contains the private key of the service. This is currently used both to encrypt/sign assertions and as the client key in an HTTPS session.

cert_file

Example:

```
cert_file: "cert.pem"
```

This is the public part of the service private/public key pair. *cert_file* must be a PEM formatted file with a single certificate.

tmp_cert_file

Example:: “tmp_cert_file”: “tmp_cert.pem”

tmp_cert_file is a PEM formatted certificate file

tmp_key_file

Example:: “tmp_key_file”: “tmp_key.pem”

tmp_key_file is a PEM formatted key file.

encryption_keypairs

Indicates which certificates will be used for encryption capabilities:

```
# Encryption
'encryption_keypairs': [
    {
        'key_file': BASE_DIR + '/certificates/private.key',
        'cert_file': BASE_DIR + '/certificates/public.cert',
```

(continues on next page)

(continued from previous page)

```
    },
],
```

generate_cert_info

Specifies if information about the certificate should be generated. A boolean value can be `True` or `False`.

Example:

```
"generate_cert_info": False
```

ca_certs

This is the path to a file containing root CA certificates for SSL server certificate validation.

Example:

```
"ca_certs": full_path("cacerts.txt"),
```

metadata

Contains a list of places where metadata can be found. This can be

- a local directory accessible on the server the service runs on
- a local file accessible on the server the service runs on
- a remote URL serving aggregate metadata
- a metadata query protocol (MDQ) service URL

For example:

```
"metadata": {
    "local": [
        "/opt/metadata"
        "metadata.xml",
        "vo_metadata.xml",
    ],
    "remote": [
        {
            "url": "https://kalmar2.org/simplesaml/module.php/aggregator/?
↪id=kalmarcentral2&set=saml2",
            "cert": "kalmar2.cert",
        },
    ],
    "mdq": [
        {
            "url": "http://mdq.ukfederation.org.uk/",
            "cert": "ukfederation-mdq.pem",
            "freshness_period": "P0Y0M0DT2H0M0S",
        },
        {
            "url": "https://mdq.thaturl.org/",
```

(continues on next page)

(continued from previous page)

```
            "disable_ssl_certificate_validation": True,
            "check_validity": False,
        },
    ],
},
```

The above configuration means that the service should read two aggregate local metadata files, one aggregate metadata file from a remote server, and query a remote MDQ server. To verify the authenticity of the metadata aggregate downloaded from the remote server and the MDQ server local copies of the metadata signing certificates should be used. These public keys must be acquired by some secure out-of-band method before being placed on the local file system.

When the parameter *check_validity* is set to False metadata that have expired will be accepted as valid.

When the paramenter *disable_ssl_certificate_validation* is set to True the validity of ssl certificate will be skipped.

When using a remote metadata source, the *node_name* option can be set to define the name of the root node of the XML document, if needed. Usually, the node name will be *urn:oasis:names:tc:SAML:2.0:metadata:EntityDescriptor* or *urn:oasis:names:tc:SAML:2.0:metadata:EntityDescriptor* (node namespace and node tag name).

When using MDQ, the *freshness_period* option can be set to define a period for which the metadata fetched from the the MDQ server are considered fresh. After that period has passed the metadata are not valid anymore and must be fetched again. The period must be in the format defined in ISO 8601 or RFC3999.

By default, if *freshness_period* is not defined, the metadata are refreshed every 12 hours (*P0Y0M0DT12H0M0S*).

organization

Only used by *make_metadata.py*. Where you describe the organization responsible for the service.:

```
"organization": {
    "name": [
        ("Example Company", "en"),
        ("Exempel AB", "se")
    ],
    "display_name": ["Exempel AB"],
    "url": [
        ("http://example.com", "en"),
        ("http://exempel.se", "se"),
    ],
}
```

Note: You can specify the language of the name, or the language used on the webpage, by entering a tuple, instead of a simple string, where the second part is the language code. If you don't specify a language, the default is "en" (English).

preferred_binding

Which binding should be preferred for a service. Example configuration:

```
"preferred_binding" = {
    "single_sign_on_service": [
```

(continues on next page)

(continued from previous page)

```
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact',
    ],
    "single_logout_service": [
        'urn:oasis:names:tc:SAML:2.0:bindings:SOAP',
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
        'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact',
    ],
}
```

The available services are:

- manage_name_id_service
- assertion_consumer_service
- name_id_mapping_service
- authn_query_service
- attribute_service
- authz_service
- assertion_id_request_service
- artifact_resolution_service
- attribute_consuming_service
- single_logout_service

service

Which services the server will provide; those are combinations of "idp", "sp" and "aa". So if a server is a Service Provider (SP) then the configuration could look something like this:

```
"service": {
    "sp": {
        "name": "Rolands SP",
        "endpoints": {
            "assertion_consumer_service": ["http://localhost:8087/"],
            "single_logout_service": [
                (
                    "http://localhost:8087/slo",
                    'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
                ),
            ],
        },
        "required_attributes": [
            "surname",
            "givenname",
            "edupersonaffiliation",
        ],
        "optional_attributes": ["title"],
        "idp": {
            "urn:mace:umu.se:saml:roland:idp": None,
```

(continues on next page)

(continued from previous page)

```
            },
        }
},
```

There are two options common to all services: 'name' and 'endpoints'. The remaining options are specific to one or the other of the service types. Which one is specified alongside the name of the option.

accepted_time_diff

If your computer and another computer that you are communicating with are not in sync regarding the computer clock, then here you can state how big a difference you are prepared to accept.

Note: This will indiscriminately affect all time comparisons. Hence your server may accept a statement that in fact is too old.

allow_unknown_attributes

Indicates that attributes that are not recognized (they are not configured in attribute-mapping), will not be discarded. Default to False.

xmlsec_binary

Currently xmlsec1 binaries are used for all the signing and encryption stuff. This option defines where the binary is situated.

Example:

```
"xmlsec_binary": "/usr/local/bin/xmlsec1",
```

xmlsec_path

This option is used to define non-system paths where the xmlsec1 binary can be located. It can be used when the xmlsec_binary option is not defined.

Example:

```
"xmlsec_path": ["/usr/local/bin", "/opt/local/bin"],
```

OR:

```
from saml2.sigver import get_xmlsec_binary

if get_xmlsec_binary:
    xmlsec_path = get_xmlsec_binary(["/opt/local/bin","/usr/local/bin"])
else:
    xmlsec_path = '/usr/bin/xmlsec1'

"xmlsec_binary": xmlsec_path,
```

delete_tmpfiles

In many cases temporary files will have to be created during the encryption/decryption/signing/validation process. This option defines whether these temporary files will be automatically deleted when they are no longer needed. Setting this to False, will keep these files until they are manually deleted or automatically deleted by the OS (i.e Linux rules for /tmp). Absence of this option, defaults to True.

valid_for

How many *hours* this configuration is expected to be accurate.:

```
"valid_for": 24
```

This, of course, is only used by *make_metadata.py*. The server will not stop working when this amount of time has elapsed :-).

metadata_key_usage

This specifies the purpose of the entity's cryptographic keys used to sign data. If this option is not configured it will default to `"both"`.

The possible options for this configuration are `both`, `signing`, `encryption`.

If metadata_key_usage is set to `"signing"` or `"both"`, and a cert_file is provided the value of use in the KeyDescriptor element will be set to `"signing"`.

If metadata_key_usage is set to `"both"` or `"encryption"` and a enc_cert is provided the value of `"use"` in the KeyDescriptor will be set to `"encryption"`.

Example:

```
"metadata_key_usage" : "both",
```

secret

A string value that is used in the generation of the RelayState.

Example:

```
"secret": "0123456789",
```

crypto_backend

Defines the crypto backend used for signing and encryption. The default is `xmlsec1`. The options are `xmlsec1` and `XMLSecurity`.

If set to "XMLSecurity", the crypto backend will be pyXMLSecurity.

Example:

```
"crypto_backend": "xmlsec1",
```

verify_encrypt_advice

Specifies if the encrypted assertions in the advice element should be verified. Can be `True` or `False`.

Example:

```
def verify_encrypt_cert(cert_str):
    osw = OpenSSLWrapper()
    ca_cert_str = osw.read_str_from_file(full_path("root_cert/localhost.ca.crt"))
    valid, mess = osw.verify(ca_cert_str, cert_str)
    return valid
```

```
"verify_encrypt_cert_advice": verify_encrypt_cert,
```

verify_encrypt_cert_assertion

Specifies if the encrypted assertions should be verified. Can be `True` or `False`.

Example:

```
"verify_encrypt_cert_assertion": verify_encrypt_cert
```

Specific directives

Directives that are specific to a certain type of service.

idp/aa

Directives that are specific to an IdP or AA service instance.

sign_assertion

Specifies if the IdP should sign the assertion in an authentication response or not. Can be True or False. Default is False.

sign_response

Specifies if the IdP should sign the authentication response or not. Can be True or False. Default is False.

encrypt_assertion

Specifies if the IdP should encrypt the assertions. Can be `True` or `False`. Default is `False`.

encrypted_advice_attributes

Specifies if assertions in the advice element should be encrypted. Can be `True` or `False`. Default is `False`.

encrypt_assertion_self_contained

Specifies if all encrypted assertions should have all namespaces self contained. Can be `True` or `False`. Default is `True`.

want_authn_requests_signed

Indicates that the AuthnRequest received by this IdP should be signed. Can be `True` or `False`. The default value is `False`.

want_authn_requests_only_with_valid_cert

When verifying a signed AuthnRequest ignore the signature and verify the certificate.

policy

If the server is an IdP and/or an AA, then there might be reasons to do things differently depending on who is asking (which is the requesting service); the policy is where this behaviour is specified.

The keys are SP entity identifiers, Registration Authority names, or 'default'. First, the policy for the requesting service is looked up using the SP entityID. If no such policy is found, and if the SP metadata includes a Registration Authority then a policy for the registration authority is looked up using the Registration Authority name. If no policy is found, then the 'default' is looked up. If there is no default and only SP entity identifiers as keys, then the server will only accept connections from the specified SPs.

An example might be:

```
"service": {
    "idp": {
        "policy": {
            # a policy for a service
            "urn:mace:example.com:saml:roland:sp": {
                "lifetime": {"minutes": 5},
                "attribute_restrictions": {
                    "givenName": None,
                    "surName": None,
                },
            },

            # a policy for a registration authority
            "http://www.swamid.se/": {
                "attribute_restrictions": {
                    "givenName": None,
                },
            },

            # the policy for all other services
            "default": {
                "lifetime": {"minutes":15},
                "attribute_restrictions": None, # means all I have
                "name_form": "urn:oasis:names:tc:SAML:2.0:attrname-format:uri",
                "entity_categories": [
                    "edugain",
```

(continues on next page)

(continued from previous page)

```
                ],
            },
        }
    }
}
```

lifetime This is the maximum amount of time before the information should be regarded as stale. In an Assertion, this is represented in the NotOnOrAfter attribute.

attribute_restrictions By default, there are no restrictions as to which attributes should be returned. Instead, all the attributes and values that are gathered by the database backends will be returned if nothing else is stated. In the example above the SP with the entity identifier "urn:mace:umu.se:saml:roland:sp" has an attribute restriction: only the attributes 'givenName' and 'surName' are to be returned. There are no limitations as to what values on these attributes that can be returned.

name_form Which name-form that should be used when sending assertions. Using this information, the attribute name in the data source will be mapped to the friendly name, and the saml attribute name will be taken from the uri/oid defined in the attribute map.

nameid_format Which nameid format that should be used. Defaults to *urn:oasis:names:tc:SAML:2.0:nameid-format:transient*.

entity_categories Entity categories to apply.

sign Possible choices: "response", "assertion", "on_demand"

If restrictions on values are deemed necessary, those are represented by regular expressions.:

```
"service": {
    "aa": {
        "policy": {
            "urn:mace:umu.se:saml:roland:sp": {
                "lifetime": {"minutes": 5},
                "attribute_restrictions": {
                     "mail": [".*\.umu\.se$"],
                }
            }
        }
    }
}
```

Here only mail addresses that end with ".umu.se" will be returned.

scope

A list of string values that will be used to set the `<Scope>` element The default value of regexp is `False`.

Example:

```
"scope": ["example.org", "example.com"],
```

ui_info

This determines what information to display about an entity by configuring its mdui:UIInfo element. The configurable options include;

privacy_statement_url The URL to information about the privacy practices of the entity.

information_url Which URL contains localized information about the entity.

logo The logo image for the entity. The value is a dictionary with keys height, width and text.

display_name The localized name for the entity.

description The localized description of the entity. The value is a dictionary with keys text and lang.

keywords The localized search keywords for the entity. The value is a dictionary with keys lang and text.

Example:

```
"ui_info": {
"privacy_statement_url": "http://example.com/saml2/privacyStatement.html",
"information_url": "http://example.com/saml2/info.html",
"logo": {
    "height": "40",
    "width" : "30",
    "text": "http://example.com/logo.jpg"
},
"display_name": "Example Co.",
"description" : {"text":"Exempel Bolag","lang":"se"},
"keywords": {"lang":"en", "text":["foo", "bar"]}
}
```

name_qualifier

A string value that sets the `NameQualifier` attribute of the `<NameIdentifier>` element.

Example:

```
"name_qualifier": "http://authentic.example.com/saml/metadata",
```

session_storage

Example:

```
"session_storage": ("mongodb", "session")
```

domain

Example:

```
"domain": "umu.se",
```

sp

Directives specific to SP instances

authn_requests_signed

Indicates if the Authentication Requests sent by this SP should be signed by default. This can be overridden by application code for a specific call.

This sets the AuthnRequestsSigned attribute of the SPSSODescriptor node of the metadata so the IdP will know this SP preference.

Valid values are True or False. Default value is True.

Example:

```
"service": {
    "sp": {
        "authn_requests_signed": True,
    }
}
```

want_response_signed

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.

Valid values are True or False. Default value is True.

Example:

```
"service": {
    "sp": {
        "want_response_signed": True,
    }
}
```

force_authn

Mandates that the identity provider MUST authenticate the presenter directly rather than rely on a previous security context.

Example:

```
"service": {
    "sp": {
        "force_authn": True,
    }
}
```

name_id_policy_format

A string value that will be used to set the `Format` attribute of the `<NameIDPolicy>` element of an `<AuthnRequest>`.

Example:

```
"service": {
    "sp": {
        "name_id_policy_format": "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
↪",
    }
}
```

name_id_format_allow_create

A boolean value (`True` or `False`) that will be used to set the `AllowCreate` attribute of the `<NameIDPolicy>` element of an `<AuthnRequest>`.

Example:

```
"service": {
    "sp": {
        "name_id_format_allow_create": True,
    }
}
```

name_id_format

A list of string values that will be used to set the `<NameIDFormat>` element of the metadata of an entity.

Example:

```
"service": {
    "sp": {
        "name_id_format": [
            "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent",
            "urn:oasis:names:tc:SAML:2.0:nameid-format:transient",
        ]
    }
}
```

allow_unsolicited

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.

Example:

```
"service": {
    "sp": {
        "allow_unsolicited": True,
    }
}
```

hide_assertion_consumer_service

When set to true the AuthnRequest will not include the AssertionConsumerServiceURL and ProtocolBinding attributes.

Example:

```
"service": {
    "sp": {
        "hide_assertion_consumer_service": True,
    }
}
```

This kind of functionality is required for the eIDAS SAML profile.

> eIDAS-Connectors SHOULD NOT provide AssertionConsumerServiceURL.

Note: This is relevant only for the eIDAS SAML profile.

sp_type

Sets the value for the eIDAS SPType node. By the eIDAS specification the value can be one of *public* and *private*.

Example:

```
"service": {
    "sp": {
        "sp_type": "private",
    }
}
```

Note: This is relevant only for the eIDAS SAML profile.

sp_type_in_metadata

Whether the SPType node should appear in the metadata document or as part of each AuthnRequest.

Example:

```
"service": {
    "sp": {
        "sp_type_in_metadata": True,
    }
}
```

Note: This is relevant only for the eIDAS SAML profile.

requested_attributes

A list of attributes that the SP requires from an eIDAS-Service (IdP). Each attribute is an object with the following attributes:

- friendly_name
- name

- required
- name_format

Where friendly_name is an attribute name such as *DateOfBirth*, name is the full attribute name such as *http://eidas.europa.eu/attributes/naturalperson/DateOfBirth*, required indicates whether this attributed is required for authentication, and name_format indicates the name format for that attribute, such as *urn:oasis:names:tc:SAML:2.0:attrname-format:uri*.

It is mandatory that at least name or friendly_name is set. By default attributes are assumed to be required. Missing attributes are inferred based on the attribute maps data.

Example:

```
"service": {
    "sp": {
        "requested_attributes": [
            {
                "name": "http://eidas.europa.eu/attributes/naturalperson/
↪PersonIdentifier",
            },
            {
                "friendly_name": "DateOfBirth",
                "required": False,
            },
        ],
    }
}
```

Note: This is relevant only for the eIDAS SAML profile.

This option is different from the required_attributes and optional_attributes parameters that control the requested attributes in the metadata of an SP.

idp

Defines the set of IdPs that this SP is allowed to use; if unset, all listed IdPs may be used. If set, then the value is expected to be a list with entity identifiers for the allowed IdPs. A typical configuration, when the allowed set of IdPs are limited, would look something like this:

```
"service": {
    "sp": {
        "idp": ["urn:mace:umu.se:saml:roland:idp"],
    }
}
```

In this case, the SP has only one IdP it can use.

optional_attributes

Attributes that this SP would like to receive from IdPs.

Example:

```
"service": {
    "sp": {
        "optional_attributes": ["title"],
    }
}
```

Since the attribute names used here are the user-friendly ones an attribute map must exist, so that the server can use the full name when communicating with other servers.

required_attributes

Attributes that this SP demands to receive from IdPs.

Example:

```
"service": {
    "sp": {
        "required_attributes": [
            "surname",
            "givenName",
            "mail",
        ],
    }
}
```

Again as for *optional_attributes* the names given are expected to be the user-friendly names.

want_assertions_signed

Indicates if this SP wants the IdP to send the assertions signed. This sets the WantAssertionsSigned attribute of the SPSSODescriptor node of the metadata so the IdP will know this SP preference.

Valid values are True or False. Default value is False.

Example:

```
"service": {
    "sp": {
        "want_assertions_signed": True,
    }
}
```

want_assertions_or_response_signed

Indicates that *either* the Authentication Response *or* the assertions contained within the response to this SP must be signed.

Valid values are True or False. Default value is False.

This configuration directive **does not** override `want_response_signed` or `want_assertions_signed`. For example, if `want_response_signed` is True and the Authentication Response is not signed an exception will be thrown regardless of the value for this configuration directive.

Thus to configure the SP to accept either a signed response or signed assertions set `want_response_signed` and `want_assertions_signed` both to False and this directive to True.

Example:

```
"service": {
    "sp": {
        "want_response_signed": False,
        "want_assertions_signed": False,
        "want_assertions_or_response_signed": True,
    }
}
```

discovery_response

This configuration allows the SP to include one or more Discovery Response Endpoints. The discovery_response can be the just the URL:

```
"discovery_response":["http://example.com/sp/ds"],
```

or it can be a 2 tuple of the URL+Binding:

```
from saml2.extension.idpdisc import BINDING_DISCO

"discovery_response": [("http://example.com/sp/ds", BINDING_DISCO)]
```

ecp

This configuration option takes a dictionary with the ecp client IP address as the key and the entity ID as the value.

Example:

```
"ecp": {
    "203.0.113.254": "http://example.com/idp",
}
```

requested_attribute_name_format

This sets the NameFormat attribute in the `<RequestedAttribute>` element. The name formats are defined in saml2.saml.py. If not configured the default is `NAME_FORMAT_URI` which corresponds to `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`.

Example:

```
from saml2.saml import NAME_FORMAT_BASIC
```

```
"requested_attribute_name_format": NAME_FORMAT_BASIC
```

requested_authn_context

This configuration option defines the `<RequestedAuthnContext>` for an AuthnRequest by a client. The value is a dictionary with two fields

- `authn_context_class_ref` a list of string values representing `<AuthnContextClassRef>` elements.

- `comparison` a string representing the Comparison xml-attribute value of the `<RequestedAuthnContext>` element. Per the SAML core specificiation the value should be one of "exact", "minimum", "maximum", or "better". The default is "exact".

Example:

```
"service": {
    "sp": {
        "requested_authn_context": {
            "authn_context_class_ref": [
                "urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport",
                "urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient",
            ],
            "comparison": "minimum",
        }
    }
}
```

idp/aa/sp

If the configuration is covering both two or three different service types (like if one server is actually acting as both an IdP and an SP) then in some cases you might want to have these below different for the different services.

endpoints

Where the endpoints for the services provided are. This directive has as value a dictionary with one or more of the following keys:

- artifact_resolution_service (aa, idp and sp)
- assertion_consumer_service (sp)
- assertion_id_request_service (aa, idp)
- attribute_service (aa)
- manage_name_id_service (aa, idp)
- name_id_mapping_service (idp)
- single_logout_service (aa, idp, sp)
- single_sign_on_service (idp)

The value per service is a list of endpoint specifications. An endpoint specification can either be just the URL:

```
"http://localhost:8088/A"
```

or it can be a 2-tuple (URL+binding):

```
from saml2 import BINDING_HTTP_POST
("http://localhost:8087/A", BINDING_HTTP_POST)
```

or a 3-tuple (URL+binding+index):

```
from saml2 import BINDING_HTTP_POST
("http://lingon.catalogix.se:8087/A", BINDING_HTTP_POST, 1)
```

If no binding is specified, no index can be set. If no index is specified, the index is set based on the position in the list.

Example:

```
"service":
    "idp": {
        "endpoints": {
            "single_sign_on_service": [
                ("http://localhost:8088/sso", BINDING_HTTP_REDIRECT),
            ],
            "single_logout_service": [
                ("http://localhost:8088/slo", BINDING_HTTP_REDIRECT),
            ],
        },
    },
},
```

only_use_keys_in_metadata

If set to False, the certificate contained in a SAML message will be used for signature verification. Default True.

validate_certificate

Indicates that the certificate used in sign SAML messages must be valid. Default to False.

logout_requests_signed

Indicates if this entity will sign the Logout Requests originated from it.

This can be overridden by application code for a specific call.

Valid values are True or False. Default value is False.

Example:

```
"service": {
    "sp": {
        "logout_requests_signed": False,
    }
}
```

signing_algorithm

Default algorithm to be used. Example:

```
"service": {
    "sp": {
        "signing_algorithm": "http://www.w3.org/2001/04/xmldsig-more#rsa-sha512",
        "digest_algorithm": "http://www.w3.org/2001/04/xmlenc#sha512",
    }
}
```

digest_algorithm

Default algorithm to be used. Example:

```
"service": {
    "idp": {
        "signing_algorithm": "http://www.w3.org/2001/04/xmldsig-more#rsa-sha512",
        "digest_algorithm": "http://www.w3.org/2001/04/xmlenc#sha512",
    }
}
```

logout_responses_signed

Indicates if this entity will sign the Logout Responses while processing a Logout Request.

This can be overridden by application code when calling `handle_logout_request`.

Valid values are True or False. Default value is False.

Example:

```
"service": {
    "sp": {
        "logout_responses_signed": False,
    }
}
```

subject_data

The name of a database where the map between a local identifier and a distributed identifier is kept. By default, this is a shelve database. So if you just specify a name, then a shelve database with that name is created. On the other hand, if you specify a tuple, then the first element in the tuple specifies which type of database you want to use and the second element is the address of the database.

Example:

```
"subject_data": "./idp.subject.db",
```

or if you want to use for instance memcache:

```
"subject_data": ("memcached", "localhost:12121"),
```

shelve and *memcached* are the only database types that are currently supported.

virtual_organization

Gives information about common identifiers for virtual_organizations:

```
"virtual_organization": {
    "urn:mace:example.com:it:tek": {
        "nameid_format": "urn:oid:1.3.6.1.4.1.1466.115.121.1.15-NameID",
        "common_identifier": "umuselin",
    }
},
```

Keys in this dictionary are the identifiers for the virtual organizations. The arguments per organization are 'nameid_format' and 'common_identifier'. Useful if all the IdPs and AAs that are involved in a virtual organization have common attribute values for users that are part of the VO.

Complete example

We start with a simple but fairly complete Service provider configuration:

```
from saml2 import BINDING_HTTP_REDIRECT

CONFIG = {
    "entityid": "http://example.com/sp/metadata.xml",
    "service": {
        "sp": {
            "name": "Example SP",
            "endpoints": {
                "assertion_consumer_service": ["http://example.com/sp"],
                "single_logout_service": [
                    ("http://example.com/sp/slo", BINDING_HTTP_REDIRECT),
                ],
            },
        }
    },
    "key_file": "./mykey.pem",
    "cert_file": "./mycert.pem",
    "xmlsec_binary": "/usr/local/bin/xmlsec1",
    "delete_tmpfiles": True,
    "attribute_map_dir": "./attributemaps",
    "metadata": {
        "local": ["idp.xml"]
    }
    "organization": {
        "display_name": ["Example identities"]
    }
    "contact_person": [
        {
            "givenname": "Roland",
            "surname": "Hedberg",
            "phone": "+46 90510",
            "mail": "roland@example.com",
            "type": "technical",
        },
    ]
}
```

This is the typical setup for an SP. A metadata file to load is *always* needed, but it can, of course, contain anything from 1 up to many entity descriptions.

A slightly more complex configuration:

```
from saml2 import BINDING_HTTP_REDIRECT

CONFIG = {
    "entityid": "http://sp.example.com/metadata.xml",
    "service": {
```

(continues on next page)

(continued from previous page)

```
        "sp": {
            "name": "Example SP",
            "endpoints": {
                "assertion_consumer_service": ["http://sp.example.com/"],
                "single_logout_service": [
                    ("http://sp.example.com/slo", BINDING_HTTP_REDIRECT),
                ],
            },
            "subject_data": ("memcached", "localhost:12121"),
            "virtual_organization": {
                "urn:mace:example.com:it:tek": {
                    "nameid_format": "urn:oid:1.3.6.1.4.1.1466.115.121.1.15-NameID",
                    "common_identifier": "eduPersonPrincipalName",
                }
            },
        }
    },
    "key_file": "./mykey.pem",
    "cert_file": "./mycert.pem",
    "xmlsec_binary": "/usr/local/bin/xmlsec1",
    "delete_tmpfiles": True,
    "metadata": {
        "local": ["example.xml"],
        "remote": [
            {
                "url":"https://kalmar2.org/simplesaml/module.php/aggregator/?
↪id=kalmarcentral2&set=saml2",
                "cert":"kalmar2.pem",
            }
        ]
    },
    "attribute_maps": "attributemaps",
    "organization": {
        "display_name": ["Example identities"]
    }
    "contact_person": [
        {
            "givenname": "Roland",
            "surname": "Hedberg",
            "phone": "+46 90510",
            "mail": "roland@example.com",
            "type": "technical",
        },
    ]
}
```

Uses metadata files, both local and remote, and will talk to whatever IdP that appears in any of the metadata files.

Other considerations

Entity Categories

Entity categories and their attributes are defined in src/saml2/entity_category/<registrar-of-entity-category>.py. We can configure Entity Categories in PySAML2 in two ways:

1. Using the configuration options *entity_category_support* or *entity_category*, to generate the appropriate Entity-

Attribute metadata elements.

2. Using the configuration option *entity_categories* as part of the policy configuration, to make the entity category work as a filter on the attributes that will be released.

If the entity categories are configured as metadata, as follow:

```
'debug' : True,
'xmlsec_binary': get_xmlsec_binary([/usr/bin/xmlsec1']),
'entityid': '%s/metadata' % BASE_URL,

# or entity_category: [ ... ]
'entity_category_support': [
    edugain.COCO, # "http://www.geant.net/uri/dataprotection-code-of-conduct/v1"
    refeds.RESEARCH_AND_SCHOLARSHIP,
],

'attribute_map_dir': 'data/attribute-maps',
'description': 'SAML2 IDP',

'service': {
    'idp': {
...
```

In the metadata we'll then have:

```
<md:Extensions>
  <mdattr:EntityAttributes>
    <saml:Attribute Name="http://macedir.org/entity-category-support" NameFormat=
↪"urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type=
↪"xs:string">http://www.geant.net/uri/dataprotection-code-of-conduct/v1</
↪saml:AttributeValue>
      <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type=
↪"xs:string">http://refeds.org/category/research-and-scholarship</
↪saml:AttributeValue>
    </saml:Attribute>
  </mdattr:EntityAttributes>
```

If the entity categories are configurated in the policy section, they will act as filters on the released attributes.

Example:

```
"policy": {
  "default": {
    "lifetime": {"minutes": 15},
    # if the SP is not conform to entity_categories
    # the attributes will not be released
    "entity_categories": ["refeds",],
```

3.4 How sp_test works internally

Release

Date Sep 23, 2022

Here are a few hints on how sp_test works internally. It helps to extend it with new test classes

When you want to test a SAML2 entity with this tool, you need the following things:

1. The Test Driver Configuration, an example can be found in tests/idp_test/config.py
2. Attribute Maps mapping URNs, OIDs and friendly names
3. Key files for the test tool
4. A metadata file representing the tool
5. The Test Target Configuration file describes how to interact with the entity to be tested. The metadata for the entity is part of this file. An example can be found in tests/idp_test/test_target_config.py.

These files should be stored outside the saml2test package to have a clean separation between the package and a particular test configuration. To create a directory for the configuration files, copy the saml2test/tests including its contents.

3.4.1 (1) Class and Object Structure

Client (sp_test/__init__.py)

Its life cycle is responsible for following activities:

- read config files and command line arguments (the test driver's config is "json_config")
- initialize the test driver IDP
- initialize a Conversation
- start the Conversion with .do_sequence_and_tests()
- post-process log messages

Conversation (sp_test/base.py)

Operation (oper)

- Comprises an id, name, sequence and tests
- Example: 'sp-00': {"name": 'Basic Login test', "sequence": [(Login, AuthnRequest, AuthnResponse, None)], "tests": {"pre": [], "post": []}

OPERATIONS

- set of operations provided in sp_test
- can be listed with the -l command line option

Sequence

- A list of flows
- Example: see "sequence" item in operation dict

Test (in the context of an operation)

- class to be executed as part of an operation, either before ("pre") or after ("post") the sequence or in between a SAML request and response ("mid"). There are standard tests with the Request class (VerifyAuthnRequest) and operation-specific tests.
- Example for an operation-specific "mid" test: VerifyIfRequestIsSigned
- A test may be specified together with an argument as a tuple.

Flow

- A tuple of classes that together implement a SAML request-response pair between IDP and SP (and possibly other actors, such as a discovery service or IDP-proxy). A class can be derived from Request, Response (or other), Check or Operation.
- A flow for a solicited authentication consists of 4 classes:
  - flow[0]: Operation (Handling a login flow such as discovery or WAYF - not implemented yet)
  - flow[1]: Request (process the authentication request)
  - flow[2]: Response (send the authentication response)
  - flow[3]: Check (optional - can be None. E.g. check the response if a correct error status was raised when sending a broken response)

Check (and subclasses)

- An optional class that is executed on receiving the SP's HTTP response(s) after the SAML response. If there are redirects, it will be called for each response.
- Writes a structured test report to conv.test_output
- It can check for expected errors, which do not cause an exception but in contrary are reported as a success

Interaction

- An interaction automates a human interaction. It searches a response from a test target for some constants, and if there is a match, it will create a response suitable response.

3.4.2 (2) Simplified Flow

The following pseudo code is an extract showing an overview of what is executed for test sp-00:

```
do_sequence_and_test(self, oper, test):
    self.test_sequence(tests["pre"])  # currently no tests defined for sp_test
    for flow in oper:
        self.do_flow(flow)

do_flow(flow):
    if len(flow) >= 3:
        self.wb_send_GET_startpage()  # send start page GET request
        self.intermit(flow[0]._interaction)  # automate human user interface
        self.parse_saml_message()    # read relay state and saml message
    self.send_idp_response(flow[1], flow[2])  # construct, sign & send a nice␣
↪Response from config, metadata and request
```

(continues on next page)

(continued from previous page)

```
    if len(flow) == 4:
        self.handle_result(flow[3])  # pass optional check class
    else:
        self.handle_result()

send_idp_response(req_flow, resp_flow):
    self.test_sequence(req_flow.tests["mid"])   # execute "mid"-tests (request has
↪"VerifyContent"-test built in; others from config)
    # this line stands for a part that is a bit more involved .. see source

    args.update(resp._response_args)    # set userid, identity

test_sequence(sequence):
    # execute tests in sequence (first invocation usually with check.VerifyContent)
    for test in sequence:
        self.do_check(test, **kwargs)

do_check(test, **kwargs):
    # executes the test class using the __call__ construct

handle_result(response=None):
    if response:
        if isinstance(response(), VerifyEchopageContents):
            if 300 < self.last_response.status_code <= 303:
                self._redirect(self.last_response)
            self.do_check(response)
        elif isinstance(response(), Check):
            self.do_check(response)
        else:
            # A HTTP redirect or HTTP Post (not sure this is ever executed)
            ...
    else:
        if 300 < self.last_response.status_code <= 303:
            self._redirect(self.last_response)

        _txt = self.last_response.content
        if self.last_response.status_code >= 400:
            raise FatalError("Did not expected error")
```

3.4.3 (3) Status Reporting

The proper reporting of results is at the core of saml2test. Several conditions must be considered:

1. An operation that was successful because the test target reports OK (e.g. HTTP 200)
2. An operation that was successful because the test target reports NOK as expected, e.g. because of an invalid signature - HTTP 500 could be the correct response
3. An error in SAML2Test
4. An error in the configuration of SAML2Test

Status values are defined in saml2test.check like this: INFORMATION = 0, OK = 1, WARNING = 2, ERROR = 3, CRITICAL = 4, INTERACTION = 5

There are two targets to write output to: * Test_output is written to conv.test_output during the execution of the flows.

- genindex

- modindex
- search