Authentic2 Documentation

Release 2.0.2

Entr'ouvert

May 11, 2012

CONTENTS

Authentic 2 is a versatile identity provider addressing a broad range of needs, from simple to advanced setups, around web authentication, attribute sharing and namespace mapping.

Authentic 2 supports many protocols and standards, including SAML2, CAS, OpenID, LDAP, X509, OATH, and can bridge between them.

Authentic 2 is under the GNU AGPL version 3 licence.

It has support for SAMLv2 thanks to Lasso, a free (GNU GPL) implementation of the Liberty Alliance and OASIS specifications of SAML2, ID-FF1.2 and ID-WSF2.

- Authentic 2 project site
- Authentic 2 roadmap

CHAPTER

ONE

DOCUMENTATION CONTENT

1.1 Features

Authentic can authenticate users against:

- an LDAP directory,
- a SAML 2.0 identity provider,
- an OpenID identity provider,
- with an X509 certificate.

Authentic can provide authentication to web applications using the following protocols:

- OpenID,
- SAML 2.0,
- CAS 1.0 & CAS 2.0.

Authentic can proxy authentication between any two different protocols it support.

1.2 Installation

Authentic 2 installation script handles all the dependencies, except Lasso, relying on the Setuptools and the pypi repository.

To run Authentic 2 you need to install Lasso >=2.3.6. You can obtain Lasso from:

- From sources: http://lasso.entrouvert.org/download
- Debian based distribution: http://deb.entrouvert.org/

The other Authentic 2 dependencies are:

- django >= 1.3
- django-profiles >= 0.2
- south >= 0.7.3
- django-authopenid >= 0.9.6
- django-debug-toolbar >= 0.9.0

Their management depends on how you install Authentic 2:

- You can *Install Authentic directly from pypi*
- You can *Obtain the last package archive from pypi*
- You can *Obtain the last sources from the Git repository*

1.2.1 Lasso installation mock-up

Please see the Lasso website for installation details. This is a quick installation example.

Install the following Lasso dependencies:

- autoconf
- automake
- autotools-dev
- libtool
- gtk-doc-tools
- zlib1g-dev
- libglib2.0-dev
- openssl-dev
- libxml2-dev
- libxmlsec1-dev
- python2.6-dev
- python-setuptools

Obtain Lasso:

```
$wget https://dev.entrouvert.org/lasso/lasso-2.3.6.tar.gz
$tar xzvf lasso-2.3.6.tar.gz
$cd lasso-2.3.6
$./autogen.sh
```

Be sure that the Python bindings is selected as follows:

```
=============
Configuration
=============

Main
----

Compiler:                gcc
CFLAGS:
Install prefix:          /usr/local
Debugging:               no
Experimental ID-WSF:     no

Optionals builds
----------------

Available languages:    java(4.6.1) python(2.7) perl(5.12.4)

Java binding:           yes
```

```
Perl binding:           yes
PHP 5 binding:          no
Python binding:         yes

C API references:       yes
Tests suite:            no


Now type 'make install' to install lasso.
```

As indicated, build and install:

```
$make install
$ldconfig
```

Set the lasso python binding in you python path, e.g.:

```
$export PYTHONPATH="$PYTHONPATH:/usr/local/lib/python2.6/site-packages"
```

Test trying to import Lasso:

```
$python
>>> import lasso
```

1.2.2 Install Authentic directly from pypi

Using pip:

```
pip install authentic2
```

or easy_install:

```
easy_install authentic2
```

You can now run Authentic from the installation directory, e.g.:

```
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py syncdb
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py runserv
```

You should see the following output:

```
Validating models...
0 errors found

Django version 1.4, using settings 'authentic.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

You can access the running application on http://127.0.0.1:8000/
```

1.2.3 Obtain the last package archive from pypi

Download the archive on http://pypi.python.org/pypi/authentic2/.

Then, you can install it directly from the archive using pip:

```
pip install authentic2-x.z.y.tar.gz
```

or easy_install:

```
easy_install authentic2-x.z.y.tar.gz
```

You can now run Authentic from the installation directory, e.g.:

```
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py syncdb
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py runserv
```

You should see the following output:

```
Validating models...
0 errors found

Django version 1.4, using settings 'authentic.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

You can access the running application on http://127.0.0.1:8000/
```

You may not want to install the authentic2 package or you may want to manage the dependencies

Then, extract the archive:

```
tar xzvf authentic2-x.z.y.tar.gz
cd authentic2-x.z.y
```

You can now install the dependencies by hands or use pypi to install them as follows, either:

```
pip install django django-profiles south django-authopenid django-debug-toolbar
```

or using the dependencies version requirements:

```
python setup.py egg_info
pip install -r authentic2.egg-info/requires.txt
```

Then run Authentic from the extracted directory:

```
python authentic2/manage.py syncdb --migrate
python authentic2/manage.py runserver
```

You should see the following output:

```
Validating models...
0 errors found

Django version 1.4, using settings 'authentic.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

You can access the running application on http://127.0.0.1:8000/
```

1.2.4 Obtain the last sources from the Git repository

Clone the repository:

```
git clone http://repos.entrouvert.org/authentic.git
```

Then, you can install it directly using pip:

```
pip install ./authentic
```

or easy_install:

```
easy_install ./authentic
```

You can now run Authentic from the installation directory, e.g.:

```
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py syncdb
python /usr/local/lib/python2.6/site-packages/authentic2-x.y.z-py2.6.egg/authentic2/manage.py runserv
```

You should see the following output:

```
Validating models...
0 errors found

Django version 1.4, using settings 'authentic.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

You can access the running application on http://127.0.0.1:8000/
```

You may not want to install the authentic2 package or you may want to manage the dependencies

Then, extract the archive:

```
cd authentic
```

You can now install the dependencies by hands or use pypi to install them as follows, either:

```
pip install django django-profiles south django-authopenid django-debug-toolbar
```

or using the dependencies version requirements:

```
python setup.py egg_info
pip install -r authentic2.egg-info/requires.txt
```

Then run Authentic:

```
python authentic2/manage.py syncdb --migrate
python authentic2/manage.py runserver
```

You should see the following output:

```
Validating models...
0 errors found

Django version 1.4, using settings 'authentic.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

You can access the running application on http://127.0.0.1:8000/
```

1.3 Deploy Authentic 2 in production

1.3.1 DEBUG Mode by default, static files and the Django debug toolbar dependency ?

By default, Authentic 2 is in the DEBUG mode. We made this default choice because most of the Authentic 2's users will begin with Authentic 2 using the Django development server (runserver command) and we want to avoid them a bad first impression because static files would not be served. As a matter of fact, static files are served by the Django development server only when the project is run in the DEBUG mode.

In the DEBUG mode, the Django debug toolbar is used what adds a dependency.

In production, the Django development server should not be used to serve Authentic 2 and a dedicated server should also be used to serve the static files.

1.3.2 Set Authentic into the no DEBUG Mode

It is enough to edit authentic2/settings.py and set:

```
DEBUG = False
```

From then on the django-debug-toolbar package is not necessary anymore.

1.3.3 Use dedicated HTTP servers and serve static files

The best is to use a server dedicated to serve the Django applications and a different server to serve the static files.

You could for instance use apache with mod_wsgi to serve Authentic 2. You will find configuration file examples in the debian directory of the Authentic 2 sources.

Then you may want to use nginx to serve the static files.

First you need to collect the Authentic 2 static files. The static files are collected using the collectstatic command that is configured in the settings.py.

By default, running collectstatic will create a static directory in the parent directory of the authentic2 directory:

```
$python authentic2/manage.py collectstatic
```

That static directory will contain all the static files of Authentic 2.

If you want to change the path of the static directory you can edit STATIC_ROOT of the settings file.

See https://docs.djangoproject.com/en/dev/ref/contrib/staticfiles/ for more information about collectstatic.

1.4 Upgrading

1.4.1 How to upgrade to a new version of authentic ?

Authentic stores all its data in a relational database as specified in its settings.py or local_settings.py file. So in order to upgrade to a new version of authentic you have to update your database schema using the migration command — you will need to have installed the dependency django-south, see the beginning of this README file.:

```
python ./manage.py migrate
```

Then you will need to create new tables if there are.:

```
python ./manage.py syncdb
```

1.5 Specifying a different database

This is done by modifying the DATABASES dictionary in your local_settings.py file (create it in Authentic project directory); for example:

```
DATABASES['default'] = {
  'ENGINE': 'django.db.backends.postgresql',
  'NAME': 'authentic',
  'USER': 'admindb',
  'PASSWORD': 'foobar',
  'HOST': 'db.example.com',
  'PORT': '', # empty string means default value
}
```

You should refer to the Django documentation on databases settings at http://docs.djangoproject.com/en/dev/ref/settings/#databases for all the details.

1.6 General settings

1.6.1 How do I configure the general settings ?

Edit the file settings.py in the project directory authentic2.

A settings file is a Python module with module-level variables. So configure general settings is done by modifying those variables and reloading your application server.

See the django documentation for more details about the settings files management.

1.6.2 Activate or deactivate debug mode

Variable: DEBUG

Values:

- False: deactivate debug mode
- True: activate debug mode

1.6.3 Manage session cookie duration

Variable: SESSION_EXPIRE_AT_BROWSER_CLOSE

Values:

- False: Cookies are not removed when browser is closed.
- True: Cookies are removed when browser is closed.

Variable: SESSION_COOKIE_AGE

Value:

- Seconds (36000 equal 10 hours)

1.6.4 Time zone selection

Variable: TIME_ZONE

Values:

- See http://en.wikipedia.org/wiki/List_of_tz_zones_by_name

1.6.5 Activate or deactivate SSL authentication

Variable: AUTH_SSL

Values:

- False: deactivate SSL authentication
- True: activate SSL authentication

1.6.6 Activate or deactivate SAML2 authentication, Authentic 2 is a SAML2 service provider

Variable: AUTH_SAML2

Values:

- False: deactivate SAML2 authentication
- True: activate SAML2 authentication

1.6.7 Activate or deactivate OpenID authentication, Authentic 2 is an OpenID relying party

Variable: AUTH_OPENID

Values:

- False: deactivate OpenID authentication
- True: activate OpenID authentication

1.6.8 Activate or deactivate one-time password authentication

Variable: AUTH_OATH

Values:

- False: deactivate one-time password authentication
- True: activate one-time password authentication

1.6.9 Activate or deactivate Authentic 2 as a SAML2 identity provider

Variable: IDP_SAML2

Values:

- False: deactivate SAML2 identity provider
- True: activate SAML2 identity provider

1.6.10 Configure SAML2 keys

- SAML_SIGNATURE_PUBLIC_KEY: Certtificate or public key for signature
- SAML_SIGNATURE_PRIVATE_KEY: Private key for signature
- SAML_ENCRYPTION_PUBLIC_KEY: Certtificate or public key for encryption
- SAML_ENCRYPTION_PRIVATE_KEY: Private key for encryption

Values are pem files of X509 certificate or key, e.g.: SAML_SIGNATURE_PRIVATE_KEY = ‘’‘—–BEGIN RSA PRIVATE KEY—– MII...WA== —–END RSA PRIVATE KEY—–‘’‘

If SAML_ENCRYPTION_PUBLIC_KEY or SAML_ENCRYPTION_PRIVATE_KEY are not given, the signature keys are used for encryption.

1.6.11 Activate or deactivate Authentic 2 as an OpenID provider

Variable: IDP_OPENID

Values:

- False: deactivate OpenID provider
- True: activate OpenID provider

1.6.12 Activate or deactivate Authentic 2 as a CAS server

Variable: IDP_CAS

Values:

- False: deactivate CAS server
- True: activate CAS server

1.7 Authentication with an existing LDAP directory

Authentic use the module django_auth_ldap to synchronize the Django user tables with an LDAP. For complex use case, we will refer you to the django_auth_ldap documentation, see http://packages.python.org/django-auth-ldap/.

1.7.1 How to authenticate users against an LDAP server with anonymous binding ?

1. Install the django_auth_ldap module for Django:

   ```
   pip install django_auth_ldap
   ```

2. Configure your local_settings.py file for authenticating against LDAP.

The next lines must be added:

```
AUTHENTICATION_BACKENDS += ( 'django_auth_ldap.backend.LDAPBackend', )

import ldap
from django_auth_ldap.config import LDAPSearch

# Here put the LDAP URL of your server
AUTH_LDAP_SERVER_URI = 'ldap://ldap.example.com'
# Let the bind DN and bind password blank for anonymous binding
AUTH_LDAP_BIND_DN = ""
AUTH_LDAP_BIND_PASSWORD = ""
# Lookup user under the branch o=base and by mathcing their uid against the
# received login name
AUTH_LDAP_USER_SEARCH = LDAPSearch("o=base",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
```

1.7.2 How to allow members of an LDAP group to manage Authentic ?

1. First you must know the objectClass of groups in your LDAP schema, this FAQ will show you the configuration for two usual classes: groupOfNames and groupOfUniqueNames.
2. Find the relevant groupname. We will say it is: cn=admin,o=mycompany
3. Add the following lines:

   ```
   from django_auth_ldap.config import GroupOfNamesType
   AUTH_LDAP_GROUP_TYPE = GroupOfNamesType()
   AUTH_LDAP_GROUP_SEARCH = LDAPSearch("o=mycompany",
       ldap.SCOPE_SUBTREE, "(objectClass=groupOfNames)")
   AUTH_LDAP_USER_FLAGS_BY_GROUP = {
       "is_staff": "cn=admin,o=mycompany"
   }
   ```

For an objectClass of groupOfUniqueNames you would change the string GroupOfNamesType to GroupOfUniqueNamesType and grouOfNames to groupOfUniqueNames. For more complex cases see the django_auth_ldap documentation.

1.8 Authentication on Authentic 2 with PAM

This module is copied from https://bitbucket.org/wnielson/django-pam/ by Weston Nielson and the pam ctype module by Chris Atlee http://atlee.ca/software/pam/.

Add 'authentic2.vendor.dpam.backends.PAMBackend' to your `settings.py`:

```
AUTHENTICATION_BACKENDS = (
    ...
```

```
    'authentic2.vendor.dpam.backends.PAMBackend',
    ...
)
```

Now you can login via the system-login credentials. If the user is successfully authenticated but has never logged-in before, a new `User` object is created. By default this new `User` has both `is_staff` and `is_superuser` set to `False`. You can change this behavior by adding `PAM_IS_STAFF=True` and `PAM_IS_SUPERUSER` in your `settings.py` file.

The default PAM service used is `login` but you can change it by setting the `PAM_SERVICE` variable in your `settings.py` file.

1.9 How global policies are used in Authentic 2 administration

The policy management with global policies is nearly used for any kind of policy in Authentic 2.

For each kind of these policies, the system takes in account two special global policies named 'Default' and 'All':

- If no other policy applies, the policy 'Default' will apply.
- A policy can be created and attached to any related object. This policy is authoritative on policy 'Default'.
- If the policy 'All' exists, it is authoritative on any other policy.
- The global policies must be created by the administrator if necessary.

A policy is taken in account only if it is enabled.

When a regular policy is associated with an object, it is taken in account only if the option 'enable the following policy' is checked.

```
def get_sample_policy(any_object):
    # Look for a global policy 'All'
    try:
        return SamplePolicy.objects.get(name='All', enabled=True)
    except SamplePolicy.DoesNotExist:
        pass
    # Look for a regular policy
    if any_object.enable_following_sample_policy:
        if any_object.sample_policy:
            return any_object.sample_policy
    # Look for a global policy 'Default'
    try:
        return SamplePolicy.objects.get(name='Default', enabled=True)
    except SamplePolicy.DoesNotExist:
        pass
    return None
```

It is advised to add a 'Default' global policy when it is expected to apply a policy to all related objects. Add e regular policy to some objects are then used to handle particular configurations.

A 'Default' global policy should be defined to avoid misonfiguration.

A 'All' global policy should be used to enforce a global configuration for all related objects or for testing purposes.

1.10 Where do I find the Authentic 2 SAML2 metadata?

The SAML2 metadata are automatically generated.

Authentic 2 will infer from environment variables the host and port to generate the URLs contained in the medatada.

The metadata of Authentic 2 SAML2 identity provider are available at:

http[s]://your.domain.com/idp/saml2/metadata

The metadata of Authentic 2 SAML2 service provider are available at:

http[s]://your.domain.com/authsaml2/metadata

1.11 Configure SAML 2.0 service providers

1.11.1 How do I authenticate against Authentic 2 with a SAML2 service provider?

1. Declare Authentic 2 as a SAML2 identity provider on your SAML2 service provider using the SAML2 identity provider metadata of Authentic 2.

Go to http[s]://your.domain.com/idp/saml2/metadata

2. Add and configure a SAML2 service provider in Authentic 2 using the metadata of the service provider.

1.11.2 How do I add and configure a SAML2 service provider in Authentic 2?

You first need to create a new SAML2 service provider entry. This requires the SAML2 metadata of the service provider.

If your service provider is Authentic 2, the metadata are available at:

http[s]://your.domain.com/authsaml2/metadata

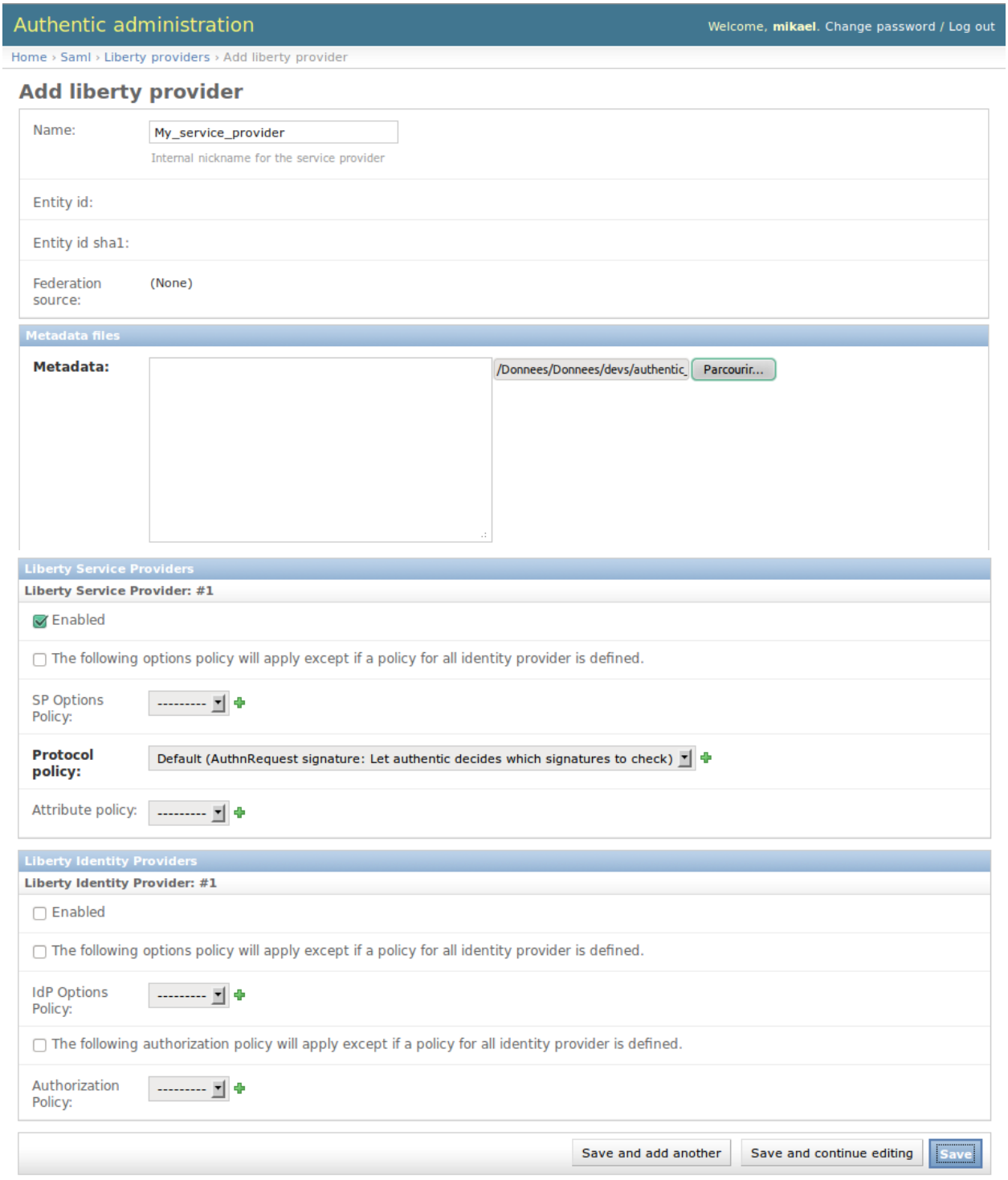
See *Where do I find the Authentic 2 SAML2 metadata?* for more information.

Create a SAML2 service provider entry

1. Go to

   http[s]://your.domain.com/admin/saml/libertyprovider/add/

2. Fill the form fields

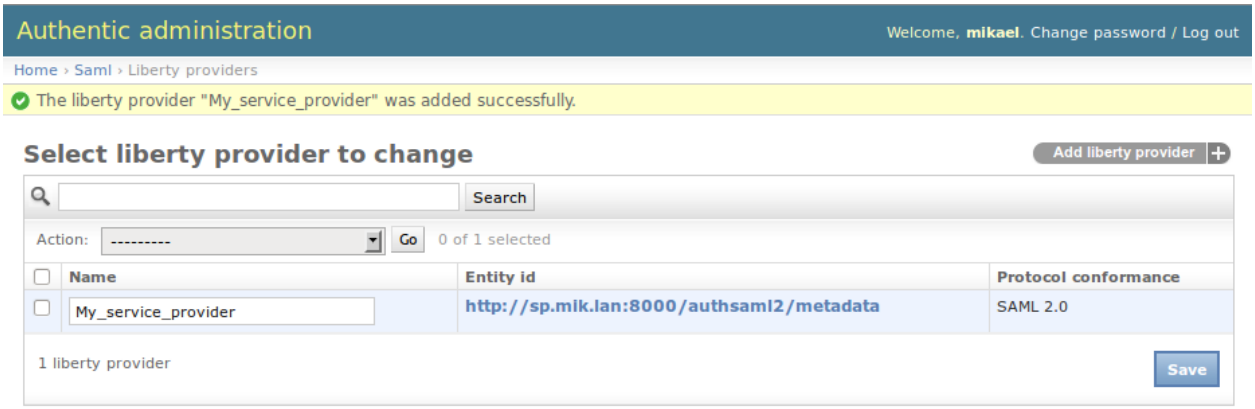


The service provider must be enabled.

See below about configuring the service provider with policies:

- options of the service provider
- protocol policy
- attribute policy

3. Save



Apply a SAML2 service provider options policy

The SAML2 options of the service provider are configured using sp options policies.

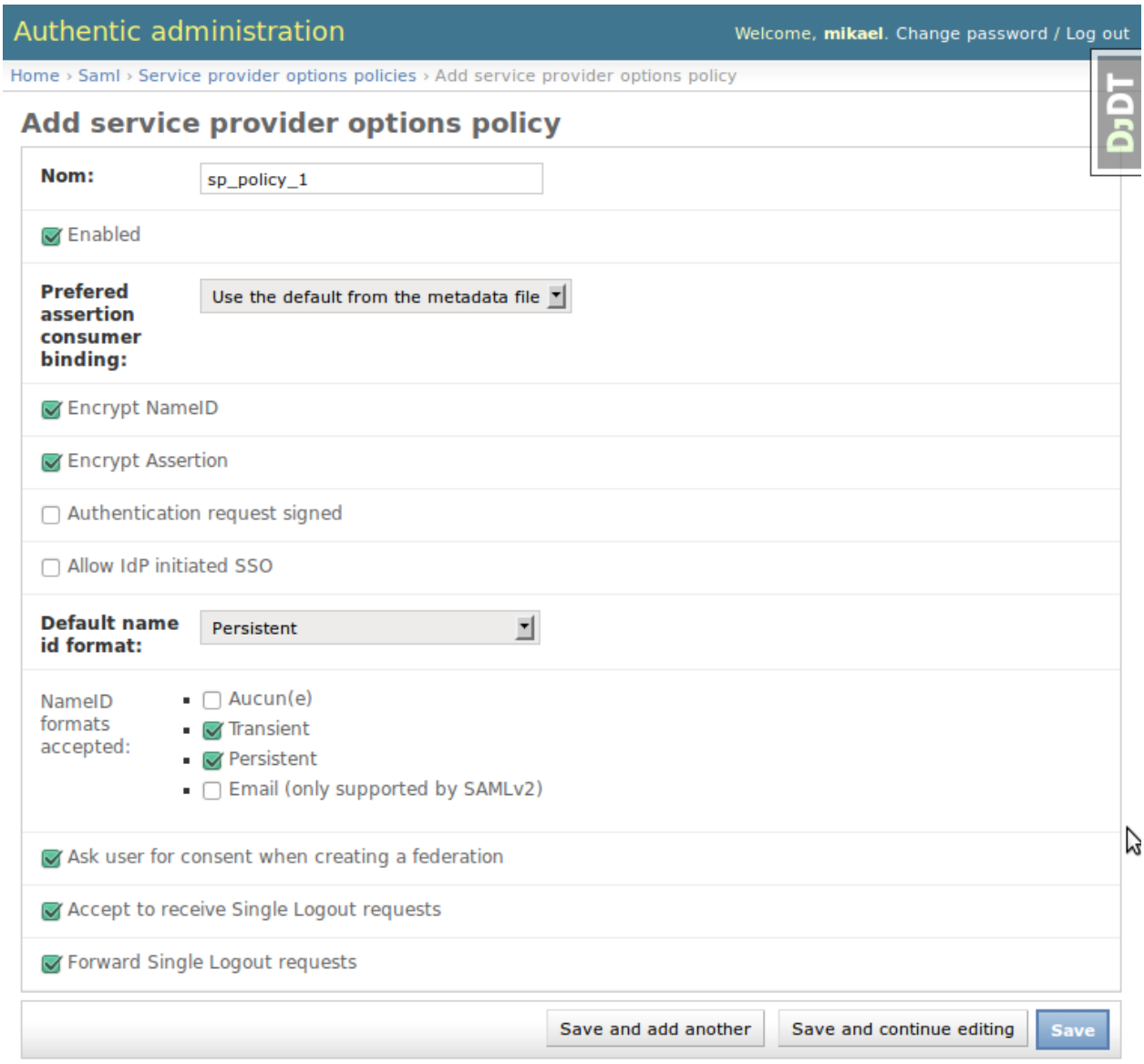
See the *administration with policy principle* page *How global policies are used in Authentic 2 administration*.

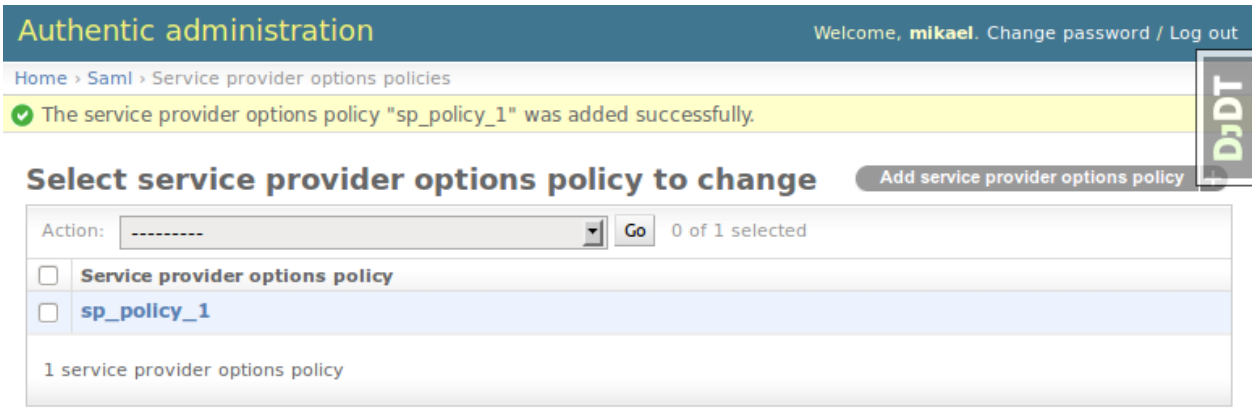
You may create a regular policy and configure your service provider to use it.

Go to:

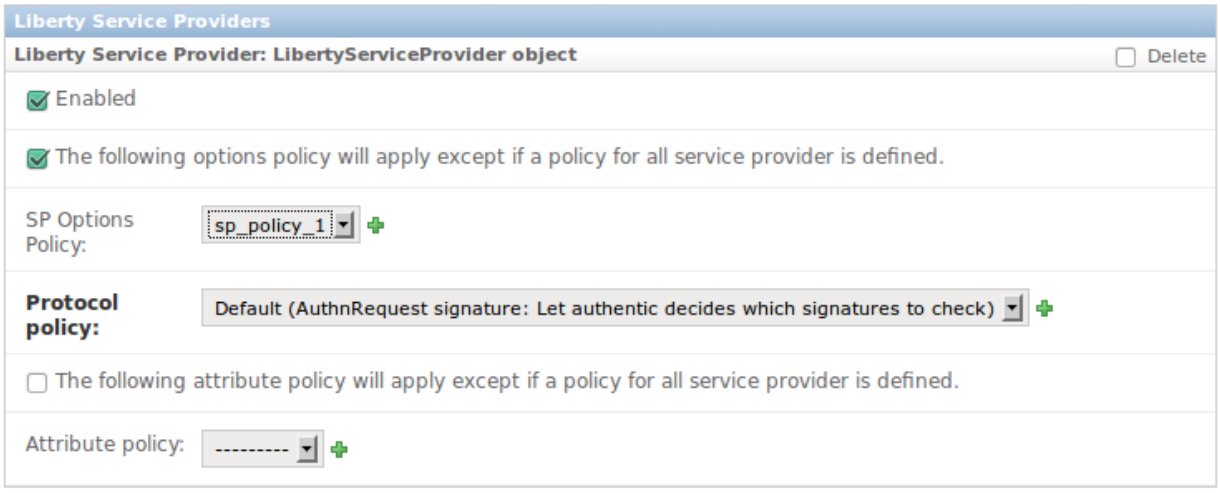
> http[s]://your.domain.com/admin/saml/spoptionsidppolicy/add/

Configure your policy and save:

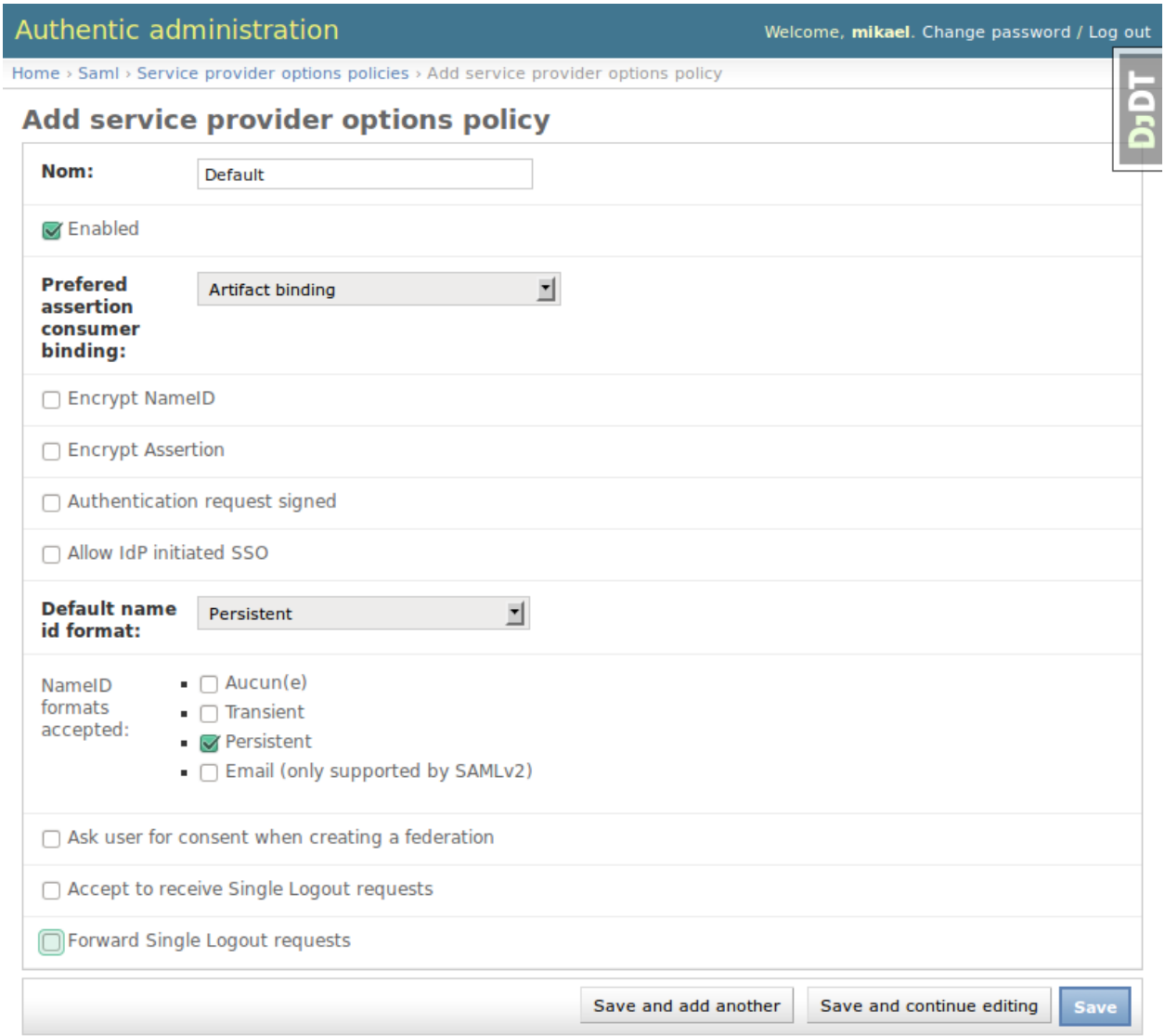




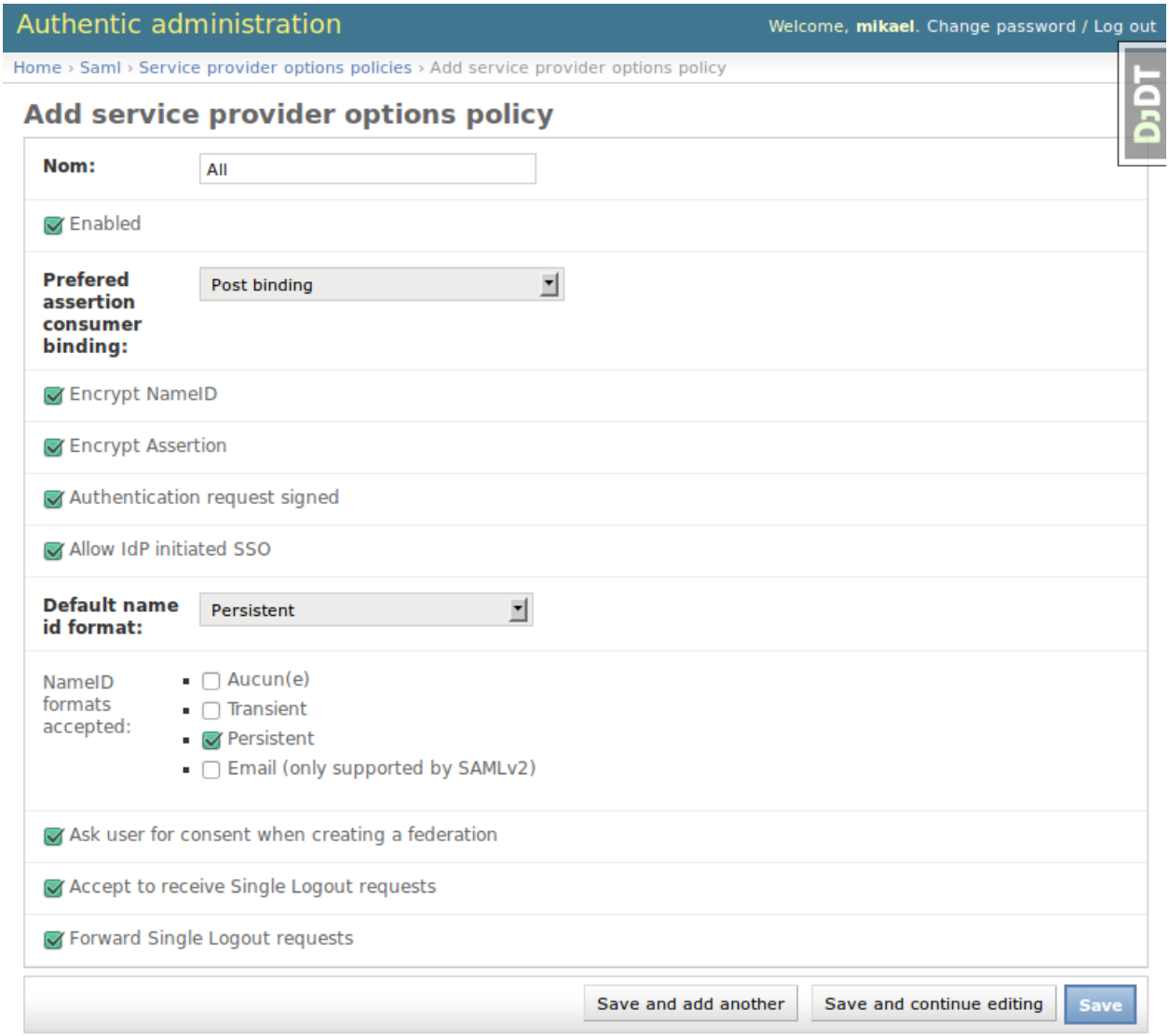
Apply the policy to the service provider:



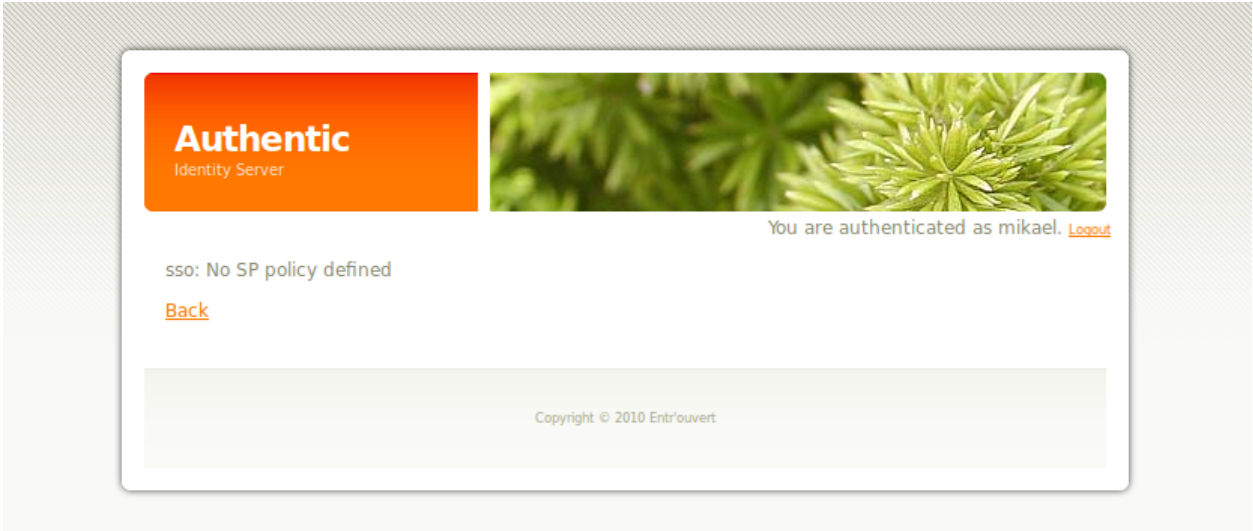
Example with a policy 'Default':



Example with a policy 'All':



If no policy is found for the configuration of the SAML2 options of a service provider, the following error is displayed to the users when a SSO request is received.



Configure the SAML2 service provider protocol options

This kind of policy does not use the policy management using global policies.

You should use the default option except if your service provider is a Shibboleth service provider, then you should use the option "Shibboleth SP (AuthnRequest Signature: Does not check signatures)".

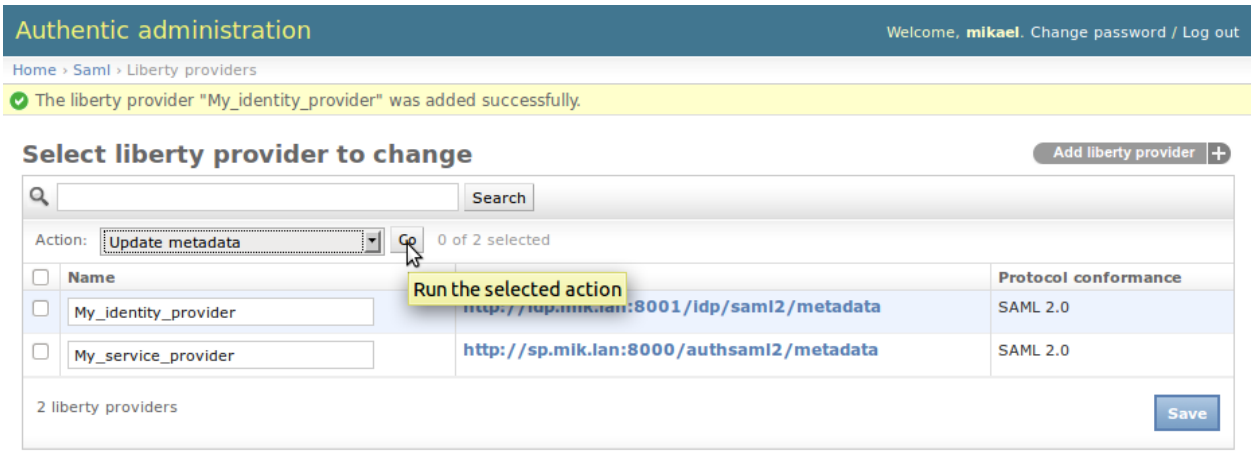
Configure the attribute policy of the service provider

See the attribute management page *Attribute Management in Authentic 2*.

1.11.3 How to refresh the metadata of a service provider hosted at a Well-Known Location?

The Well-Known Location (WKL) means that the entity Id of the provider is a URL at which the provider metadata are hosted.

To refresh them, select the provider on the list of provider, then select in the menu 'Update metadata', then click on 'Go'.



1.11.4 How to create in bulk service providers with the sync-metadata script?

See the page explaining the use of the script sync-metadata *How to create/import and delete in bulk SAML2 identity and service providers with the sync-metadata script?*.

1.12 Configure Authentic 2 as a SAML2 service provider or a SAML2 proxy

The configuration to make Authentic 2 a SAML2 service provider or a SAML2 proxy is the same. The difference comes from that Authentic 2 is may be configured or not as a SAML2 identity provider.

1.12.1 How do I authenticate against a third SAML2 identity provider?

1. Declare Authentic 2 as a SAML2 service provider on your SAML2 identity provider using the SAML2 service provider metadata of Authentic 2.

Go to http[s]://your.domain.com/authsaml2/metadata

2. Add and configure a SAML2 identity provider entry in Authentic 2 using the metadata of the identity provider.

1.12.2 How do I add and configure a SAML2 identity provider in Authentic 2?

You first need to create a SAML2 identity provider entry with the SAML2 metadata of the identity provider. Then, you configure it.

If your identity provider is Authentic 2, the metadata are available at:

http[s]://your.domain.com/idp/saml2/metadata

See *Where do I find the Authentic 2 SAML2 metadata?* for more information.

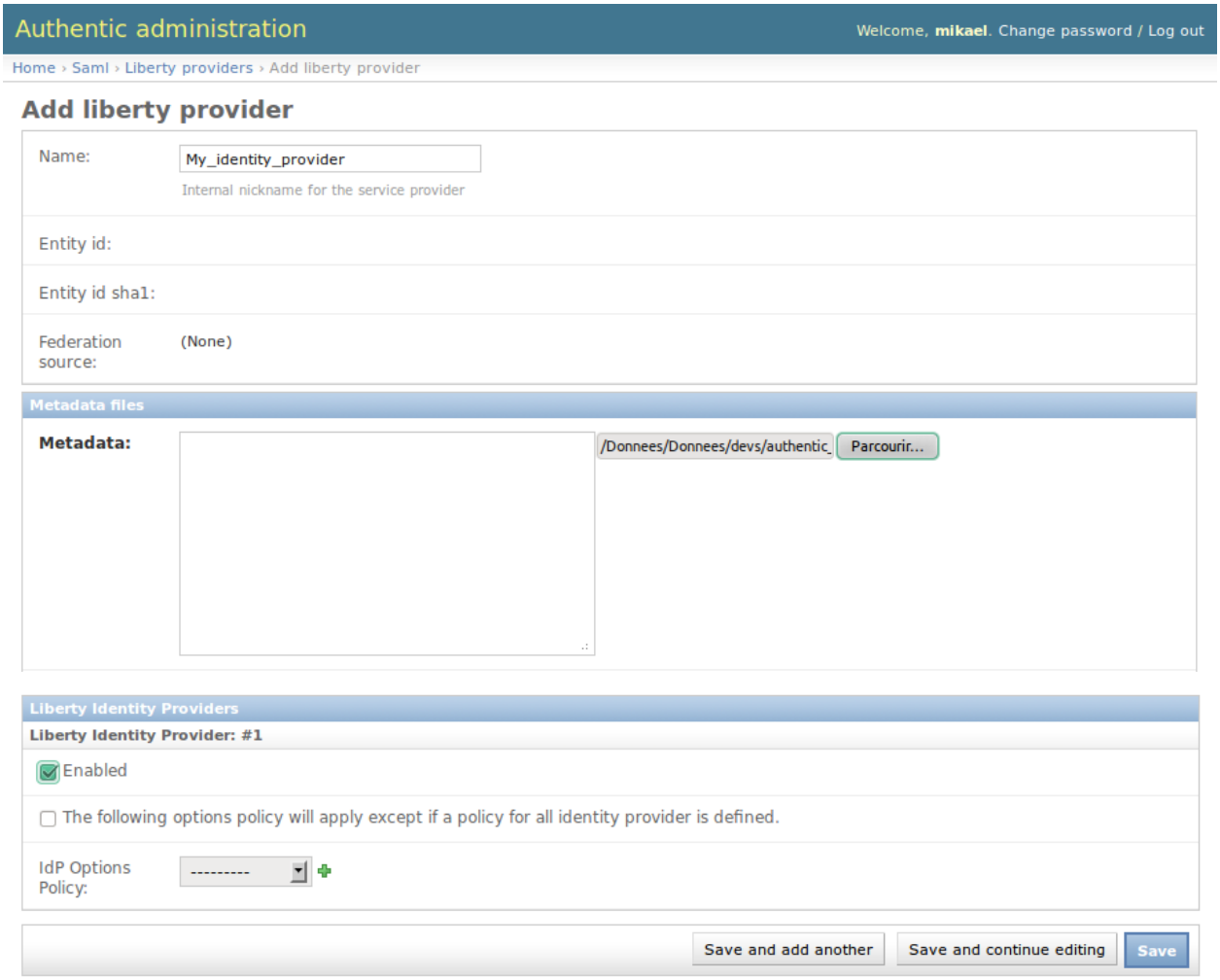
Create a SAML2 identity provider entry

You first need to create a new SAML2 identity provider entry. This requires the SAML2 metadata of the identity provider.

1. Go to

   http[s]://your.domain.com/admin/saml/libertyprovider/add/

2. Fill the form fields

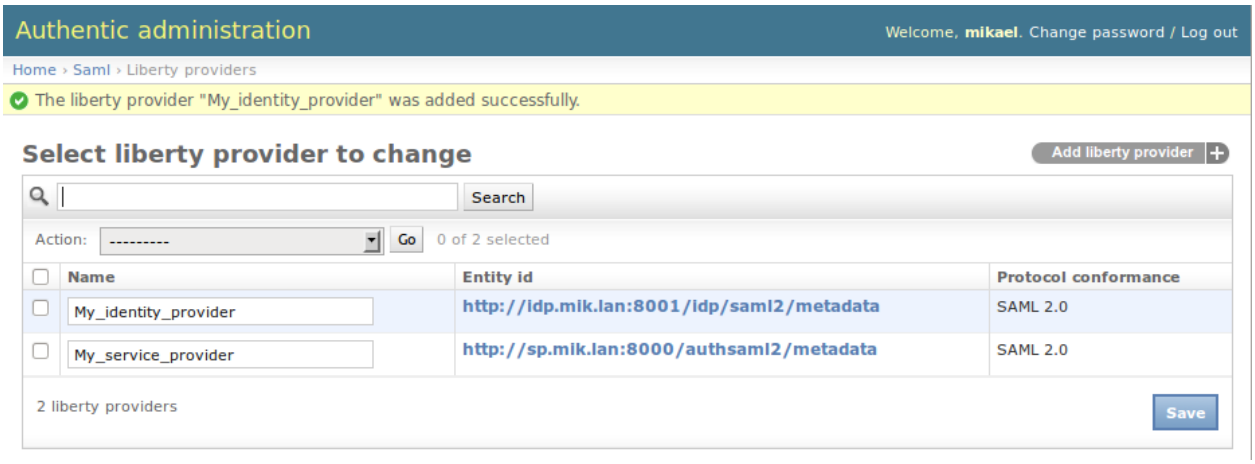


The identity provider must be enabled.

See below about configuring the identity provider with policies:

- options of the identity provider

3. Save



Apply a SAML2 identity provider options policy

The SAML2 options of the identity provider are configured using idp options policies. For the explanation of the options see the following section.

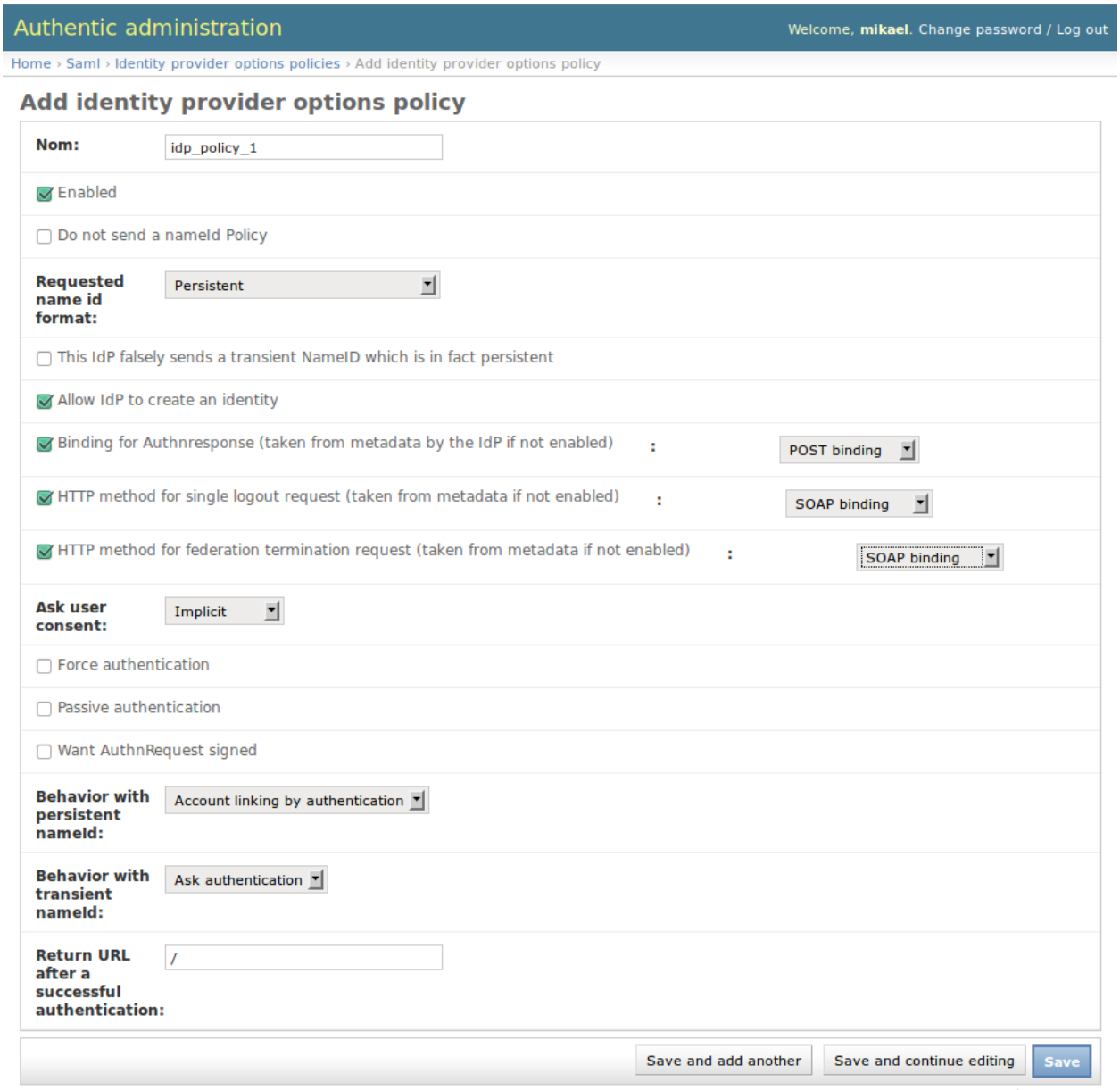
See the *administration with policy principle* page *How global policies are used in Authentic 2 administration*.

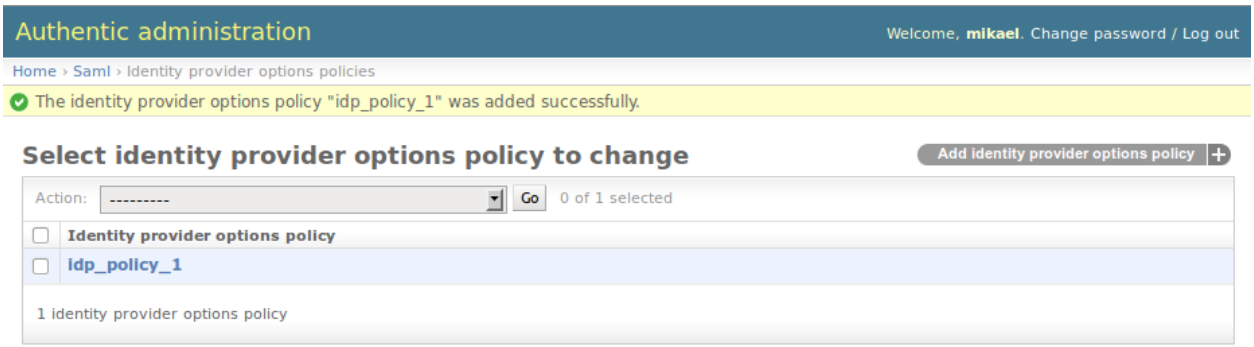
You may create a regular policy and configure your service provider to use it.

Go to:

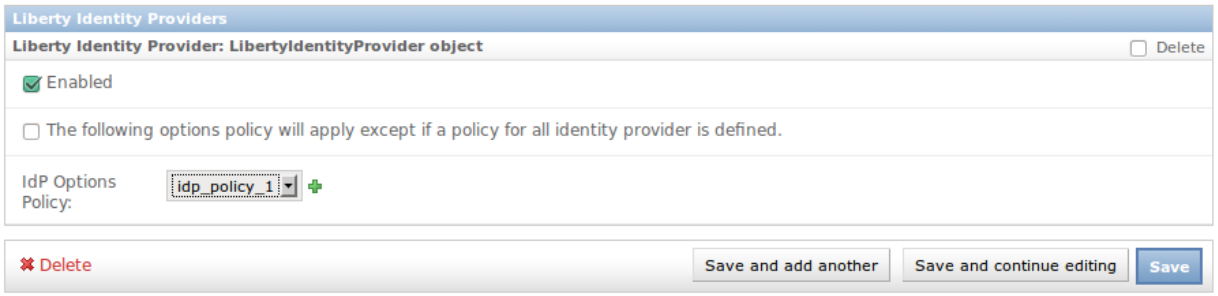
http[s]://your.domain.com/admin/saml/idpoptionssppolicy/add/

Configure your policy and save:

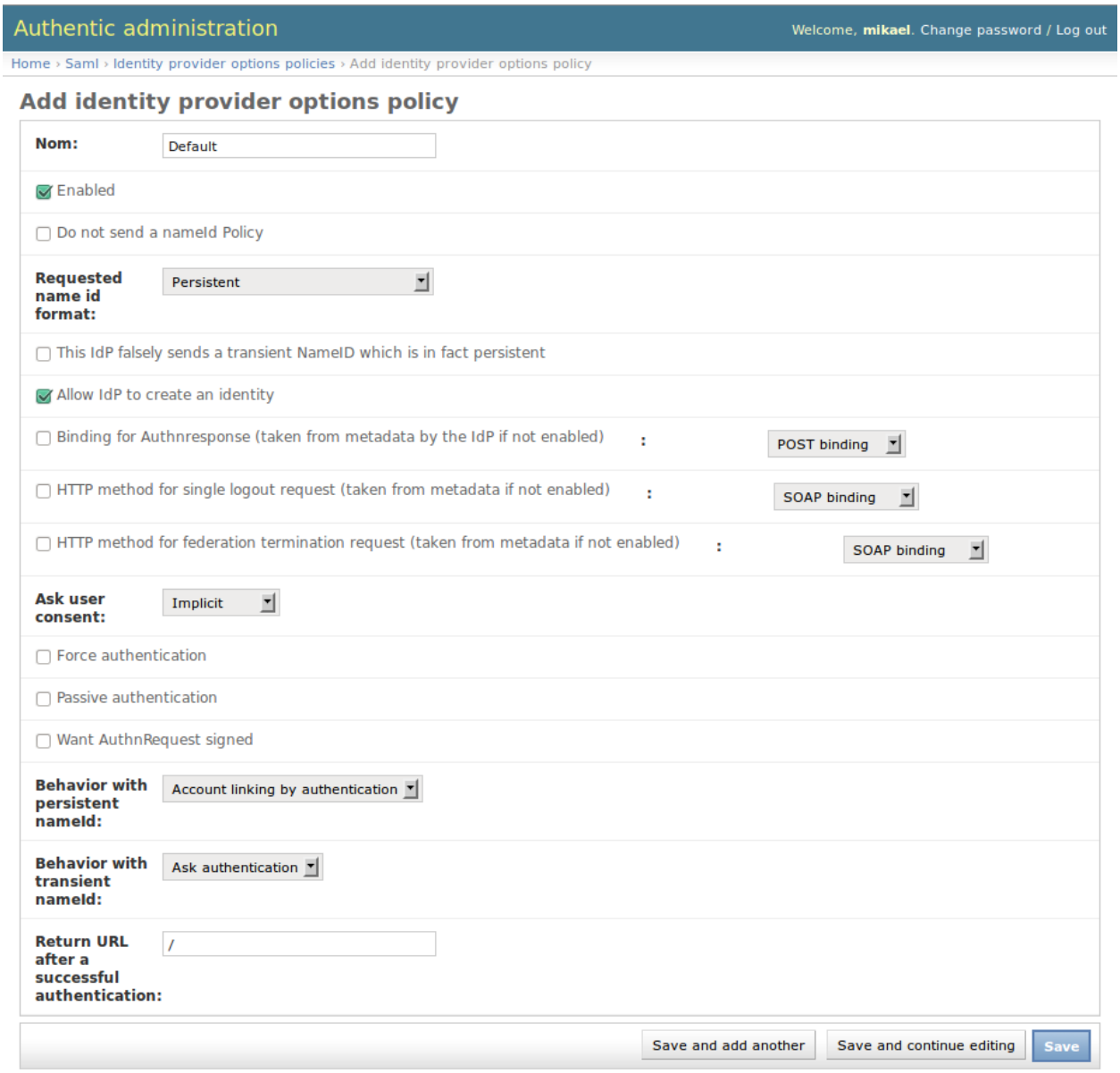




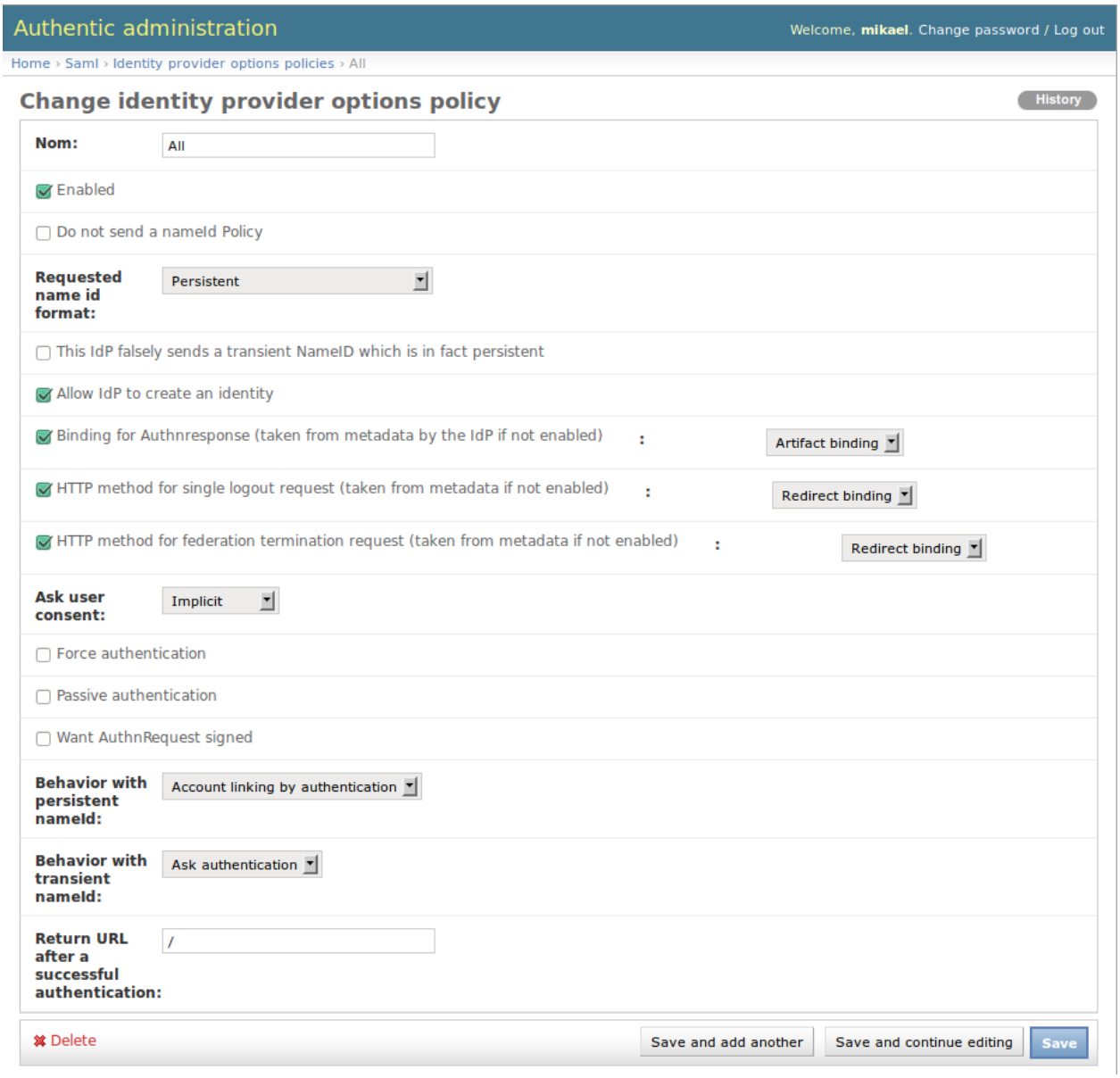
Apply the policy to the identity provider:



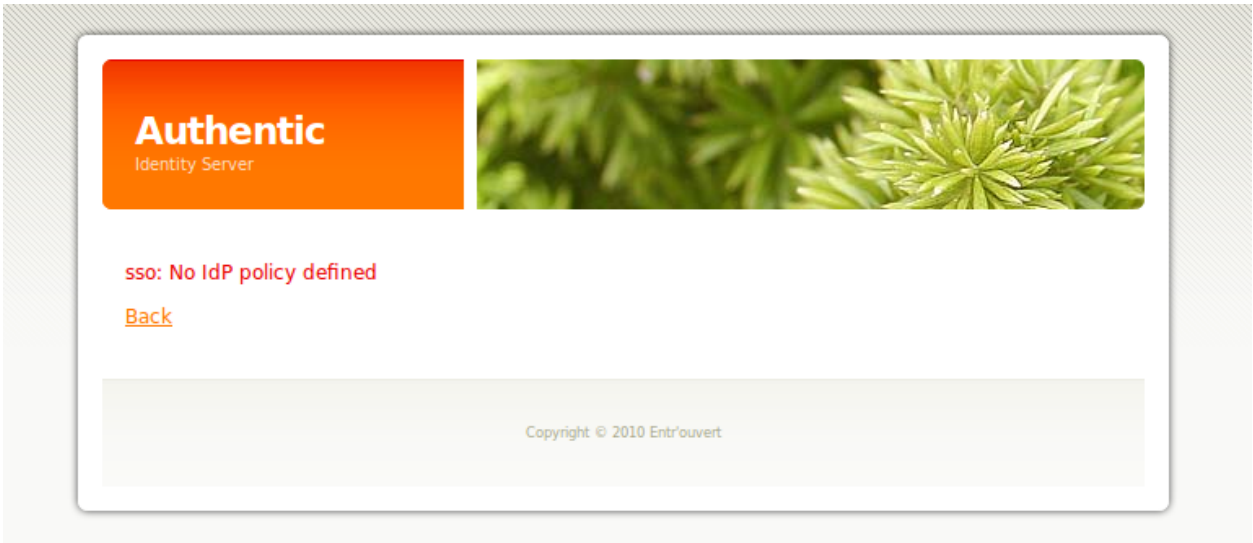
Example with a policy ‘Default’:



Example with a policy ‘All’:



If no policy is found for the configuration of the SAML2 options of an identity provider, the following error is displayed to the users when a SSO request is initiated.



SAML2 identity provider options explained

Behavior with persistent nameID

This option applies when an assertion with a persistent nameID is received and the nameID is not recognized as an existing federation.

Two values are possible: “Create new account” and “Account linking by authentication”.

The value “Create new account” makes Authentic 2 create a user account associated to the nameID received.

The value “Account linking by authentication” makes Authentic 2 ask the user to authenticate with an existing account to associate the nameID to this account.

Behavior with transient nameID

This option applies when an assertion with a transient nameID is received and there isn’t a session opened for the user yet.

Two values are possible: “Open a session” and “Ask authentication”.

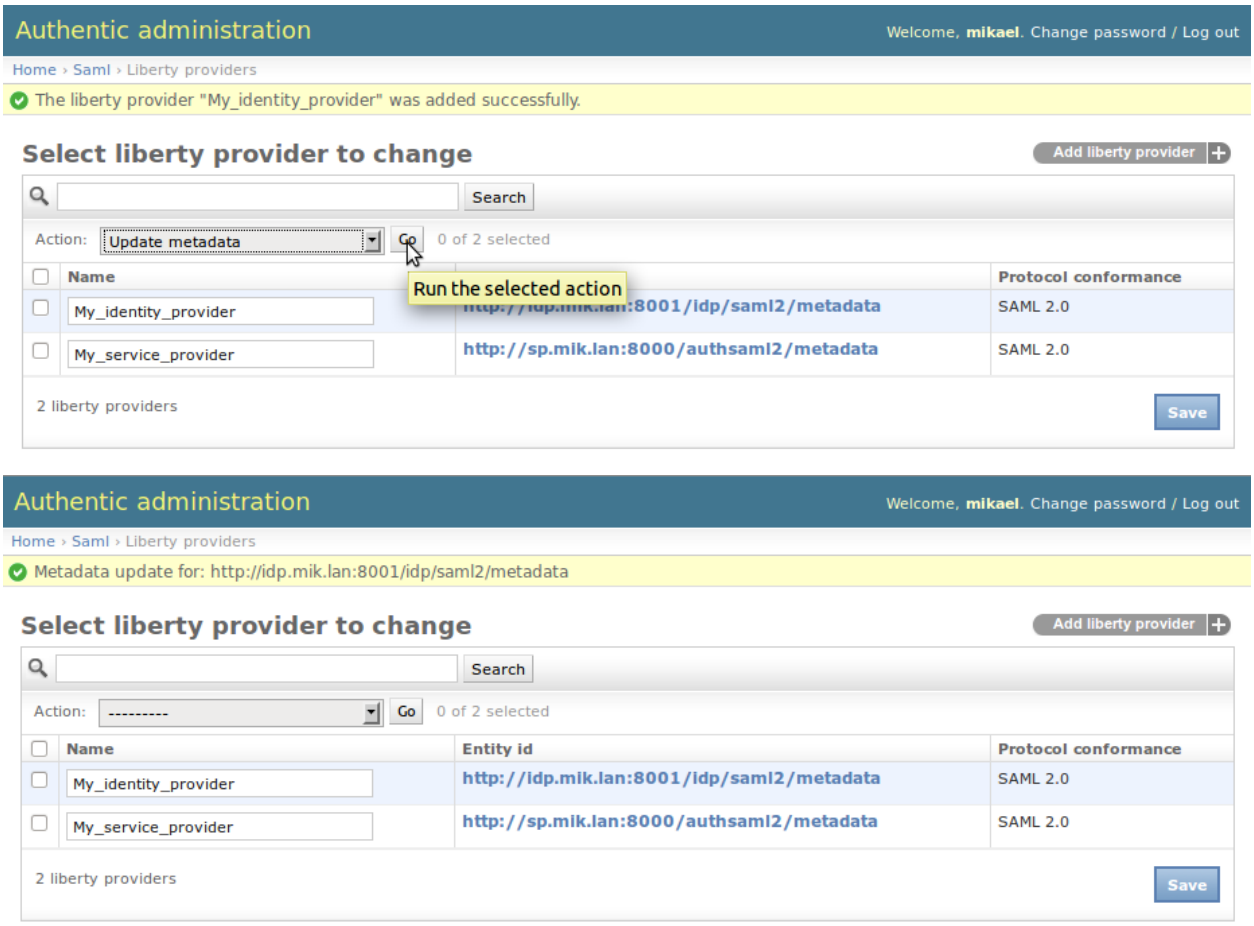
The value “Open a session” makes Authentic 2 open a session.

The value “Ask authentication” makes Authentic 2 ask for a user authentication, even when a valid assertion is received. That may have sense for instance if the SSO login is used only to receive signed attributes for users with existing accounts.

1.12.3 How to refresh the metadata of an identity provider hosted at a Well-Known Location?

The Well-Known Location (WKL) means that the entity Id of the provider is a URL at which the provider metadata are hosted.

To refresh them, select the provider on the list of provider, then select in the menu ‘Update metadata’, then click on ‘Go’.



1.12.4 How to create in bulk identity providers with the sync-metadata script?

See the page explaining the use of the script sync-metadata *How to create/import and delete in bulk SAML2 identity and service providers with the sync-metadata script?*.

1.13 Single Logout Management in Authentic 2

1.13.1 Explanation

Authentic 2 implements the single logout profile of SAML2 (SLO). Single Logout is used to realise to close user session on distributed applications. The Single Logout is managed by the IdP. However, its exists many profiles all supported by Authentic 2:

- SLO IdP initiated by SOAP
- SLO IdP initiated by Redirect
- SLO SP initiated by SOAP
- SLO SP initiated by Redirect

Then, Authentic 2 acting as an IdP but also as a SP (for proxying), a logout request can be received from:

- the logout button on the user interface;

- a service provider;
- a third identity provider.

The configuration by policy allows to refuse SLO request coming from a SP or an IdP.

The the SLO request is accepted or comes from the user interface, at the end of the process the local session on Authentic 2 will always be closed.

During the process of treatment of the logout request, when the logout request comes from a SP, if the local session was established through a third SAML2 IdP, Authentic 2 sends it a logout request (SLO proxying). Then, Authentic 2 sends logout resuests to all service providers with an active session but the requesting service provider.

During the process of treatment of the logout request, when the logout request comes from an IdP, Authentic 2 sends logout resuests to all service providers with an active session.

The configuration by policy allows to select which IdP and SP to logout forwarding is done.

Note: When a logout request comes from an IdP, the logout request is always forwarded by soap to the service providers.

Note: When a logout request comes from an SP: - if done by SOAP, the logout request is always forwarded by soap to the identity provider and service providers. - if done by Redirect, the logout request is forwarded to the identity provider according to the idp options policy and to the service providers according to their metadata.

Note: When a logout request comes from the user interface, the logout request is forwarded to the identity provider according to the idp options policy and to the service providers according to their metadata.

Note: To make the SLO works, a policy must be found for the source or the desitnation of the logout request. By default, when creating a sp options policy or an IdP options policy the SLO is accepted and forwarded.

See the *administration with policy principle* page *How global policies are used in Authentic 2 administration*.

1.13.2 How to know if a service provider supports the logout request?

Look for the following elements in the service provider metadata:

- Redirect binding:

```
<ns0:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect" Location="
```

- SOAP binding:

```
<ns0:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Location="http://sp
```

1.13.3 How to know if an identity provider supports the logout request?

Look for the following elements in the identity provider metadata:

- Redirect binding:

```
<ns0:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect" Location="
```

- SOAP binding:

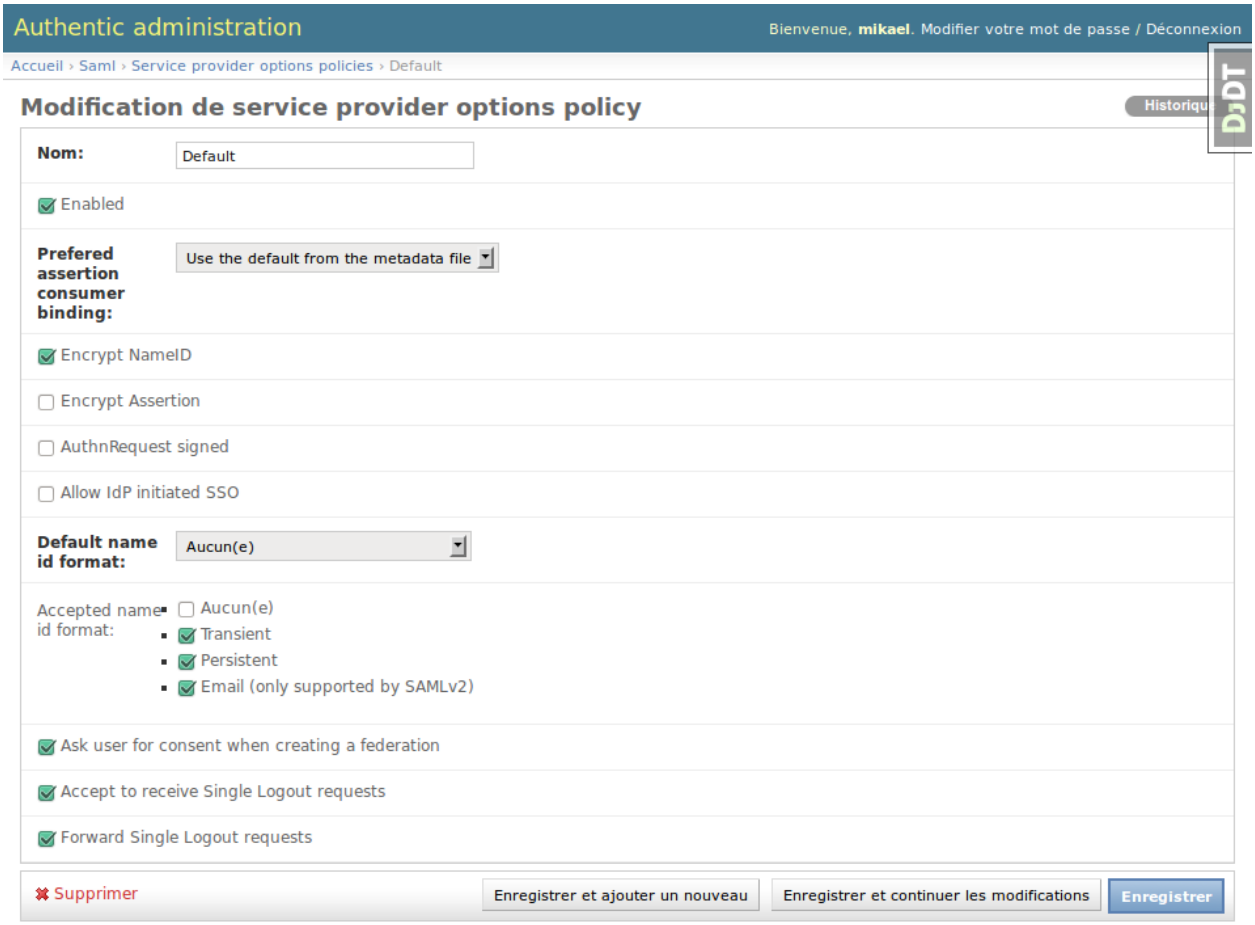
```
<ns0:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Location="http://id
```

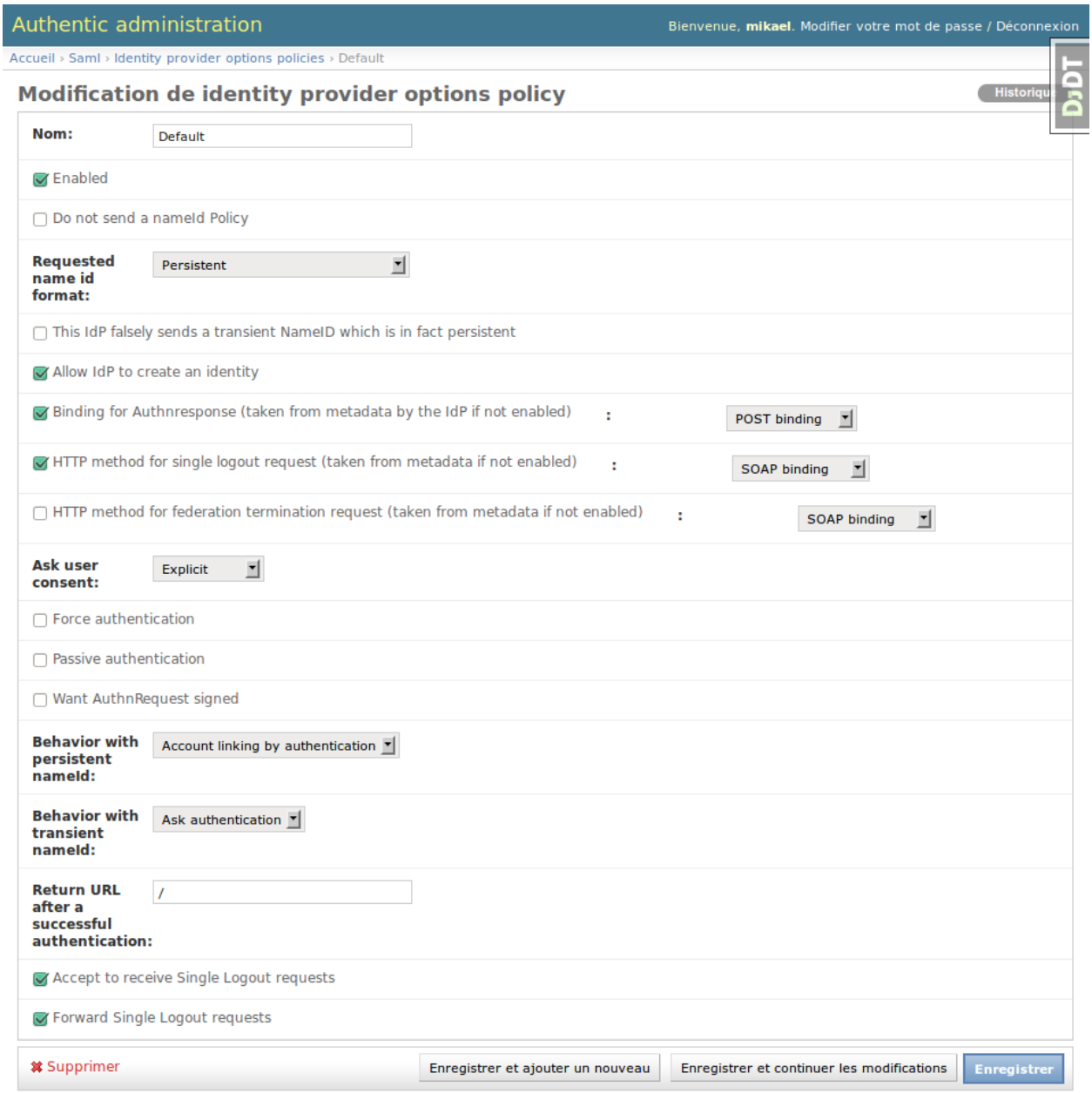
1.13.4 How activate the SLO?

No activation is required. However it is required a policy be found for the source or the desitnation of the logout request.

The sp options policy or idp options policy that applies has parameters to indicate if the sp or idp which the policy applies is allowed to send and receive logout requests.

Then, create the 'default' options policy and check the both options *Accept to receive Single Logout requests* and *Forward Single Logout requests* as follows:



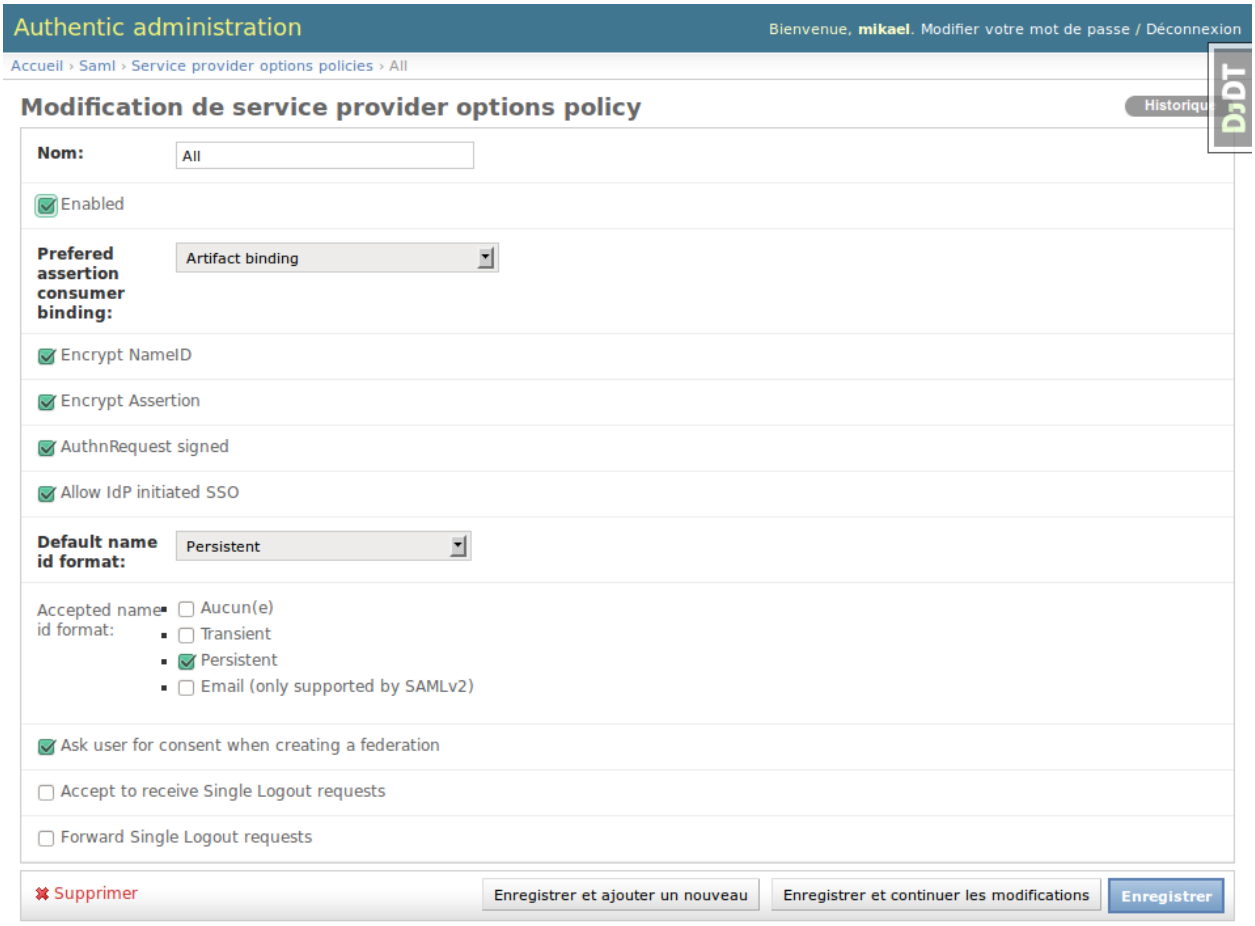


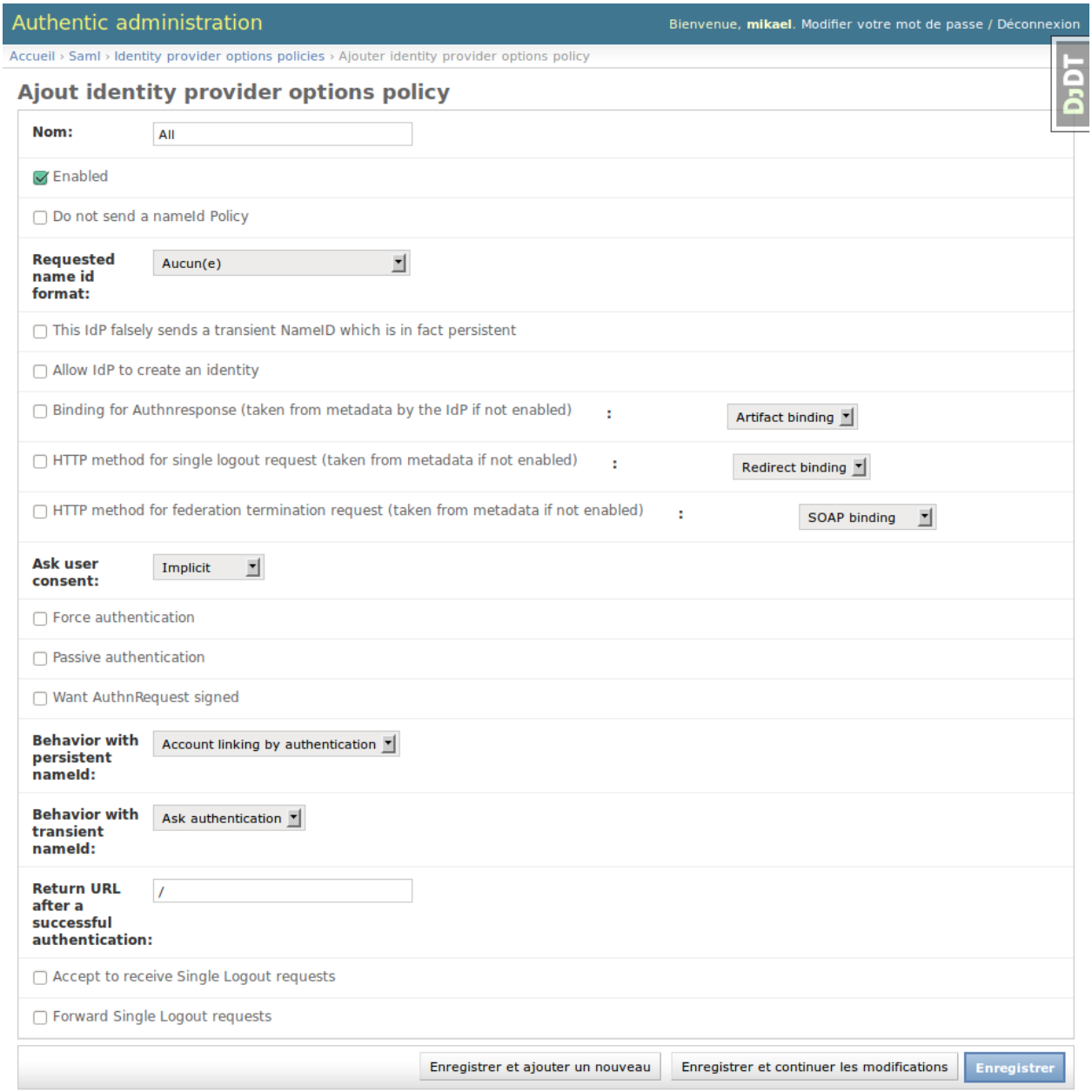
1.13.5 How deactivate the SLO?

There is no real deactivation process. When it is possible and authorized, Authentic 2 send logout requests when a logout request is received.

If an options policy is not found for the source or the destination of the logout request, the logout requests are not accepted nor forwarded.

However it is not the right way. The best is to create the 'all' options policies with the options *Accept to receive Single Logout requests* and *Forward Single Logout requests* unchecked as follows:


Authentic administration
Bienvenue, mikael. Modifier votre mot de passe / Déconnexion
Accueil › Saml › Service provider options policies › All
Modification de service provider options policy
Historiqu
Nom:
All
Enabled
Prefered assertion consumer binding:
Artifact binding
Encrypt NameID
Encrypt Assertion
AuthnRequest signed
Allow IdP initiated SSO
Default name id format:
Persistent
Accepted name-id format:
Aucun(e)
Transient
Persistent
Email (only supported by SAMLv2)
Ask user for consent when creating a federation
Accept to receive Single Logout requests
Forward Single Logout requests
Supprimer
Enregistrer et ajouter un nouveau
Enregistrer et continuer les modifications
Enregistrer



Take care that the 'all' policies are authoritative. To desactivate the SLO but for particular providers, the best is to unchecked these options on the 'default' options policies and apply regular policies to those particular providers.

1.13.6 How refuse SLO from an identity provider?

Uncheck the option *Accept to receive Single Logout requests* of the policy that applies to that identity provider.

1.13.7 How refuse SLO from a service provider?

Uncheck the option *Accept to receive Single Logout requests* of the policy that applies to that service provider.

1.13.8 How indicate identity providers to not forward logout request?

Uncheck the option *Forward Single Logout requests* of the policies that applies to the identity providers logout requests must not be forwarded.

1.13.9 How indicate service providers to not forward logout request?

Uncheck the option *Forward Single Logout requests* of the policies that applies to the service providers logout requests must not be forwarded.

1.13.10 How do manage the SLO without closing the local session?

Not implemented.

1.14 How to create/import and delete in bulk SAML2 identity and service providers with the sync-metadata script?

This section explains hot to use the script sync-metadata.

1.14.1 Presentation

This script allows to create/import and deleted in bulk SAML2 identity and service providers using standard SAML2 metadata files containing entity descriptors.

An example of such a file used in production is the global metadata file of the identity federation of French universities that can be found at http://...

Use the following command:

```
path_to_project/authentic2$ python manage.py sync-metadata file_name [options]
```

1.14.2 Options

- idp

  Load only identity providers of the metadata file.

- sp

  Load only service providers of the metadata file.

- source

  Used to tag all imported providers with a label. This option is used to metadata reloading and deletion in bulk.

  Reloading a metadata file, when a provider with same entity is found, it is updated. If a provider in the metadata file does not exist it is created. If a provider exists in the system but not in the metadata file, it is removed.

  For reloading, a source can only be associated with a unique metadata file. This is due to the fact that all providers of a source not found in the metadata file are removed.

```
path_to_project/authentic2$ python manage.py sync-metadata file_name --source=french_federation
```

- sp-policy

  To configure the SAML2 parameters of service providers imported with the script, a policy of type SPOptionsIdPPolicy must be created in the the administration interface. Either it is a global policy 'Default' or 'All' or it is a regular policy. If it is a regular policy, the policy name can be specified in parameter of the script with this option. The policy is then associated to all service providers created.

```
path_to_project/authentic2$ python manage.py sync-metadata file_name --sp-policy=sp_policy_name
```

- idp-policy

  To configure the SAML2 parameters of identity providers imported with the script, a policy of type IdPOptionsSPPolicy must be created in the the administration interface. Either it is a global policy 'Default' or 'All' or it is a regular policy. If it is a regular policy, the policy name can be specified in parameter of the script with this option. The policy is then associated to all service providers created.

```
path_to_project/authentic2$ python manage.py sync-metadata file_name --idp-policy=idp_policy_name
```

- delete

  With no options, all providers are deleted.

  With the source option, only providers with the source name given are deleted.

  This option can not be combined with options idp and sp.

- ignore-errors

  If loading of one EntityDescriptor fails, continue loading

1.15 Configure Authentic 2 as a CAS server

1.15.1 How to use Authentic 2 as a CAS 1.0 or CAS 2.0 identity provider ?

1. Activate CAS IdP support in settings.py:

   ```
   IDP_CAS = True
   ```

2. Then create the database table to hold CAS service tickets:

   ```
   python authentic2/manage.py syncdb --migrate
   ```

3. Also configure authentic2 to authenticate against your LDAP directory (see above) if your want your user attributes to be accessible from your service, if it is not necessary you can use the normal relational database storage for you users.

4. Finally configure your service to point to the CAS endpoint at:

   http[s]://your.domain.com/idp/cas/

5. If needed configure your service to resolve authenticated user with your LDAP directory (if user attributes are needed for your service)

1.16 Attribute Management in Authentic 2

1.16.1 Summary

Attribute management currently allows to configure attribute policies associated with SAML2 service providers to define attributes that are pushed in SAML2 successful authentication response delivered by Authentic 2.

User attributes can be taken from LDAP directories, the user Django profile or taken from the user Django session if Authentic 2 is also configured as a SAML2 service provider.

Indeed, when Authentic 2 acts also as a SAML2 service provider, attributes contained in the SAML2 assertion received from third IdP are put in the user session.

Attributes can thus be proxyfied during SSO with Authentic 2 configured as a SAML2 proxy.

If there is no attribute policy associate with a service provider, no attribute is forwarded to it.

The namespace of attributes received from another SAML2 IdP and of attributes pushed in the assertion given to service providers can be configured per attribute or per service provider.

By default, the namespace and format of attributes in assertion is conformant to the SAMLV2.0 X500/LDAP Attribute profile:

```
<saml:Attribute
    xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    Name="urn:oid:2.5.4.42" FriendlyName="givenName">
    <saml:AttributeValue xsi:type="xs:string"
        x500:Encoding="LDAP">Mikaël</saml:AttributeValue>
</saml:Attribute>
```

But the http://schemas.xmlsoap.org/ws/2005/05/identity/claims from the ISI profile can also be used, for instance:

```
<saml:Attribute
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"
    FriendlyName="First Name">
    <saml:AttributeValue>Mikaël</saml:AttributeValue>
</saml:Attribute>
```

1.16.2 Configuration

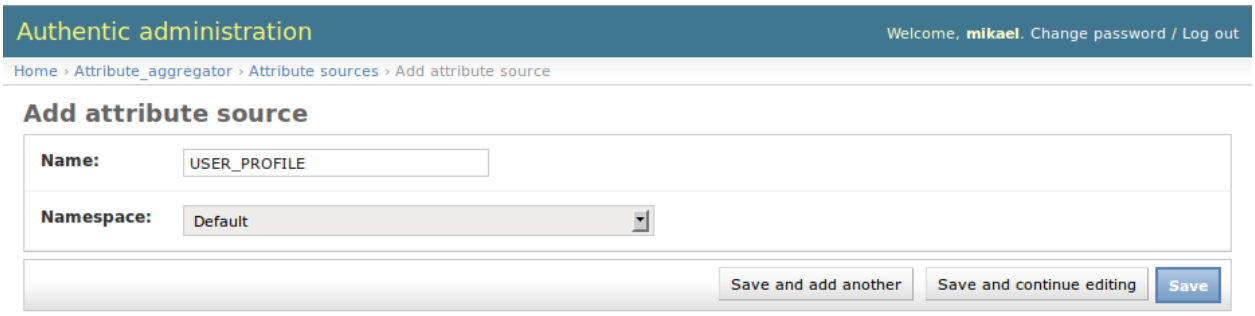
Configure local sources of attributes

The source of attributes for authentic2 are of two kinds. The LDAP sources and the user django profile.

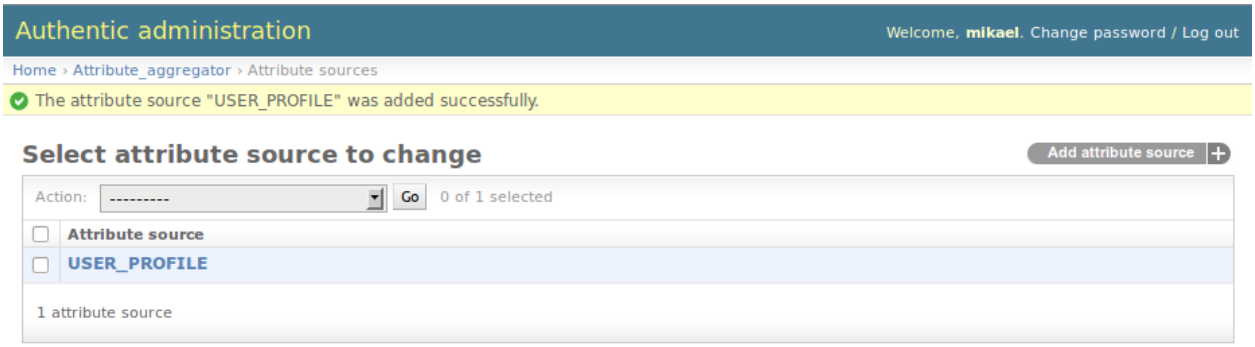
Declare the Django profile source

Add an attribute source named USER_PROFILE with namespace 'Default'.

1. Go to http[s]://your.domain.com/admin/attribute_aggregator/attributesource/add/
2. Write 'USER_PROFILE' in name field



3. Save



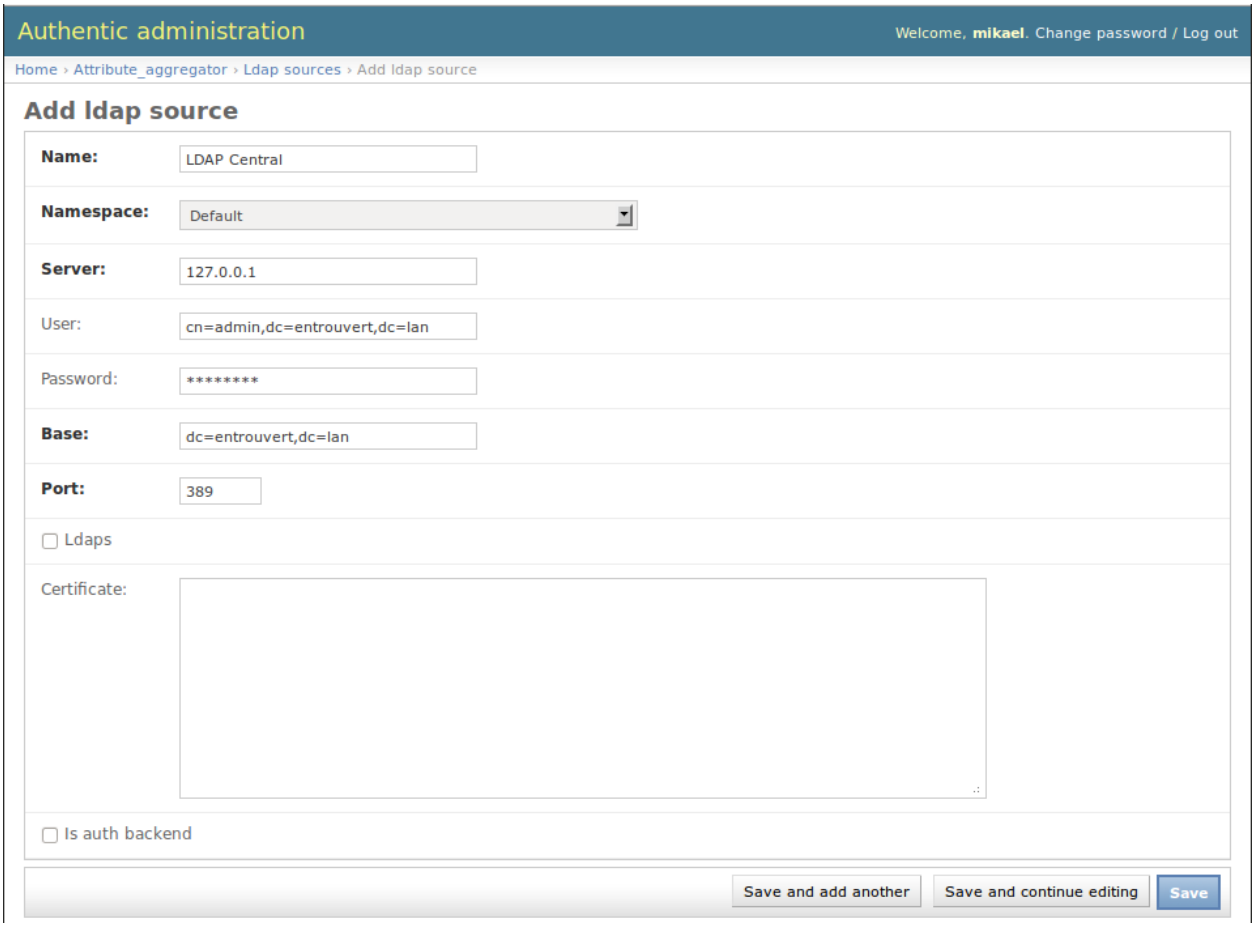
Add an LDAP Source

For LDAP sources, objects of type 'LDAPSource' must be created.

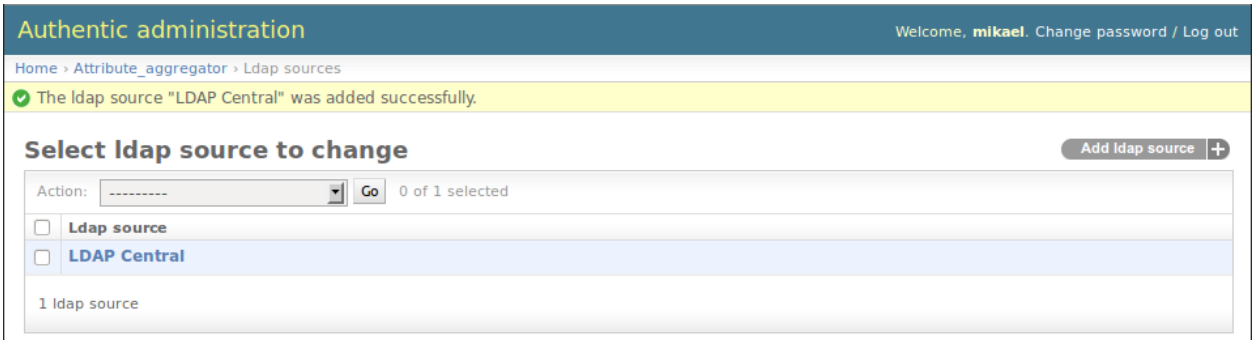
Even if the authentication is based on LDAP authentification, thus that a server is configured in settings.py, it is necessary to create a corresponding 'LDAPSource' to use it as a source of attribute.

1. Go to http[s]://your.domain.com/admin/attribute_aggregator/ldapsource/add/
2. Fill form fields

Only the field Name, Server, User, Password, Base and Port are used for now. **The namespace of LDAP source must be kept to 'Default', since the system namespace is based on LDAP.**



3. Save

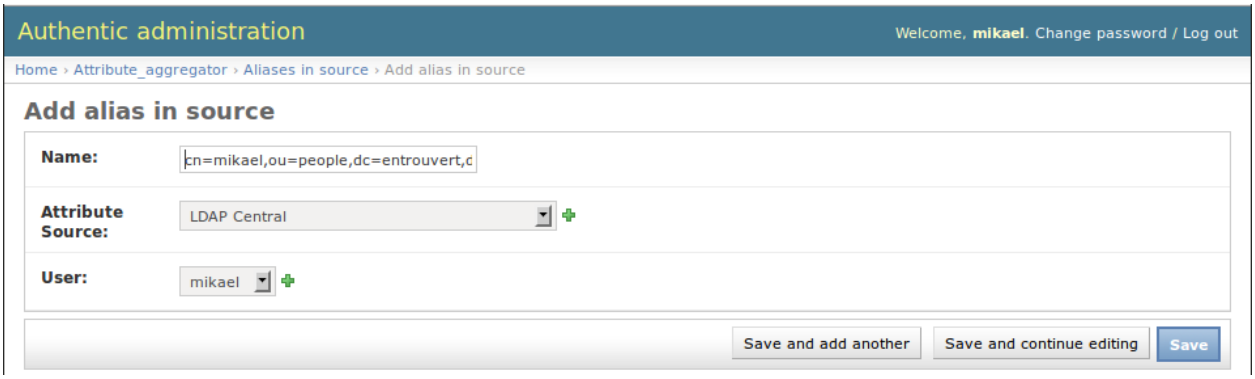


Manage user distinguished names in LDAP directories

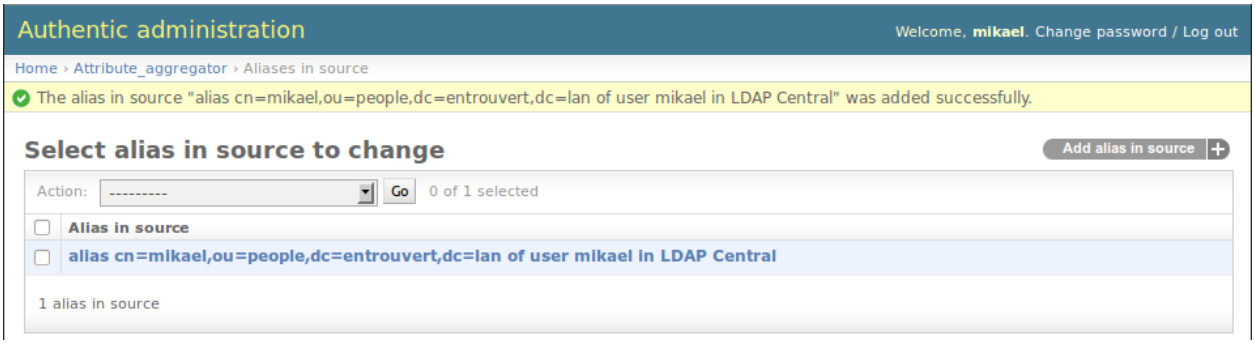
To find the user in a LDAP directory, authentic2 must know its distinguished name (DN). If this LDAP has been used when the user has authenticated, Authentic 2 learn the user DN. Nothing has to be done from this point of view.

However, if it is expected that user attributes be taken in a directory that is not used by the user for authentication, it is necessary to manually indicate to Authentic 2 what is the user DN in the directory. For this, a user alias in source is created for the user:

1. Go to http[s]://your.domain.com/admin/attribute_aggregator/useraliasinsource/add/
2. Fill form fields



3. Save



Configure attributes from local sources pushed to SAML2 service providers in SSO response

Reminder:

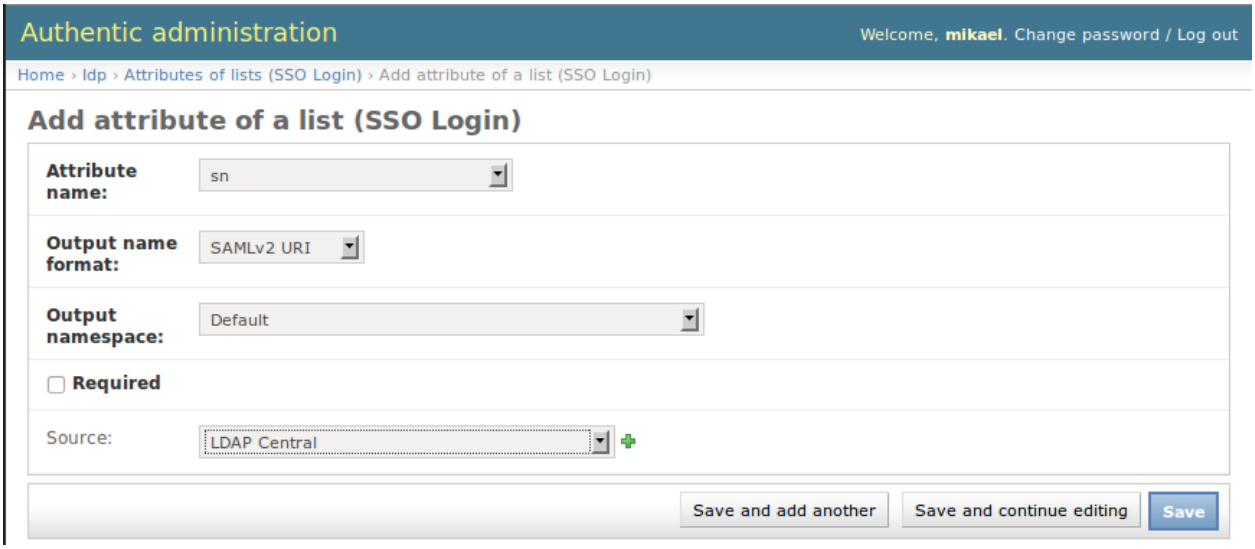
- The default name format in SAML2 assertions is URI
- The default namespace called 'Default' is LDAP

In summary:

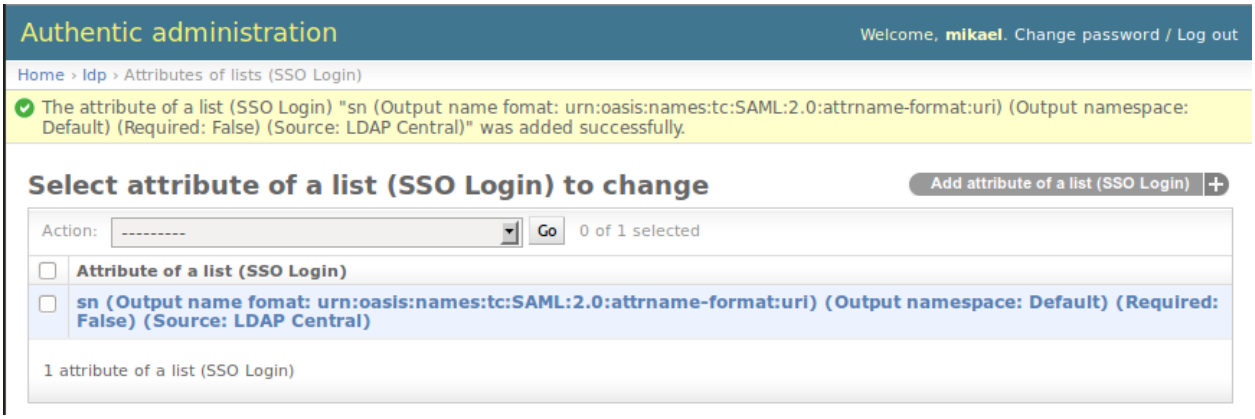
1. Create attribute items indicating an attribute name, a source, the name format expected and the namespace expected for the attribute name and friendly name if any.
2. Create a named list of attribute items.
3. Create an attribute policy and associate the previous list or associate the previous list to a existing attribute policy.
4. Associate the policy to a service provider.

Create attribute items

1. Go to http[s]://your.domain.com/admin/idp/attributeitem/add/
2. Fill form fields

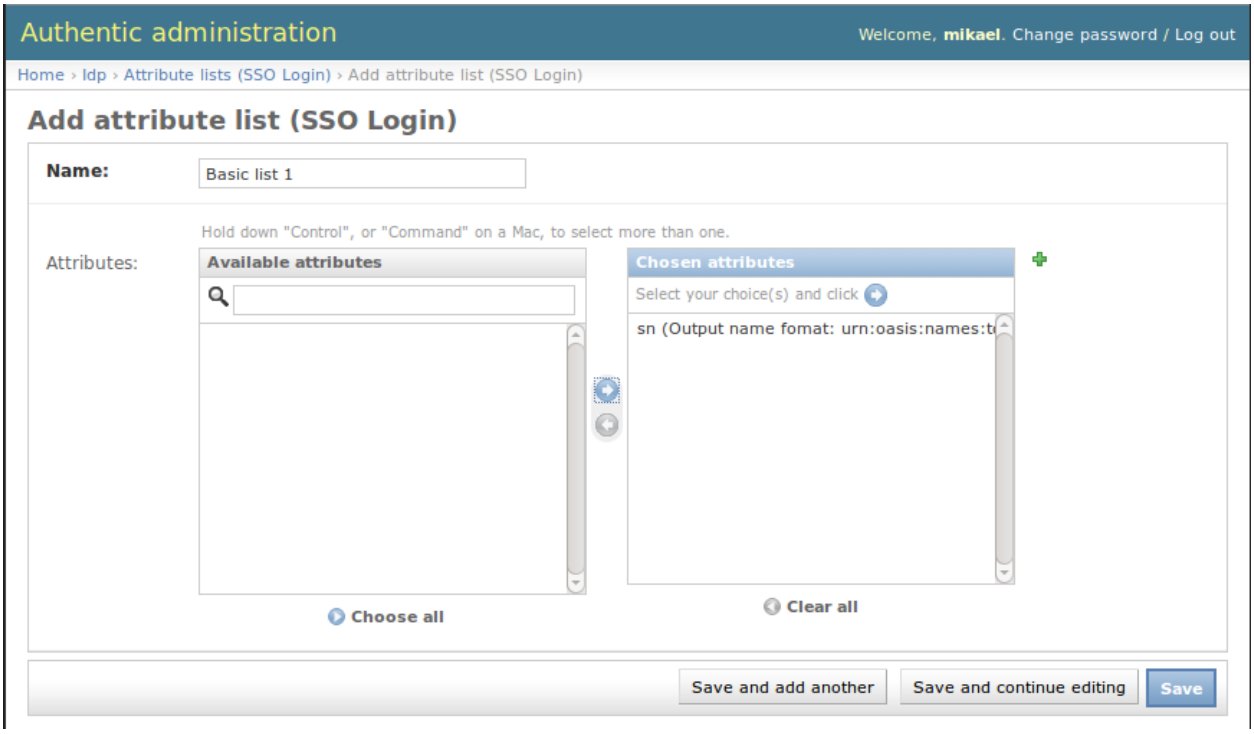


3. Save

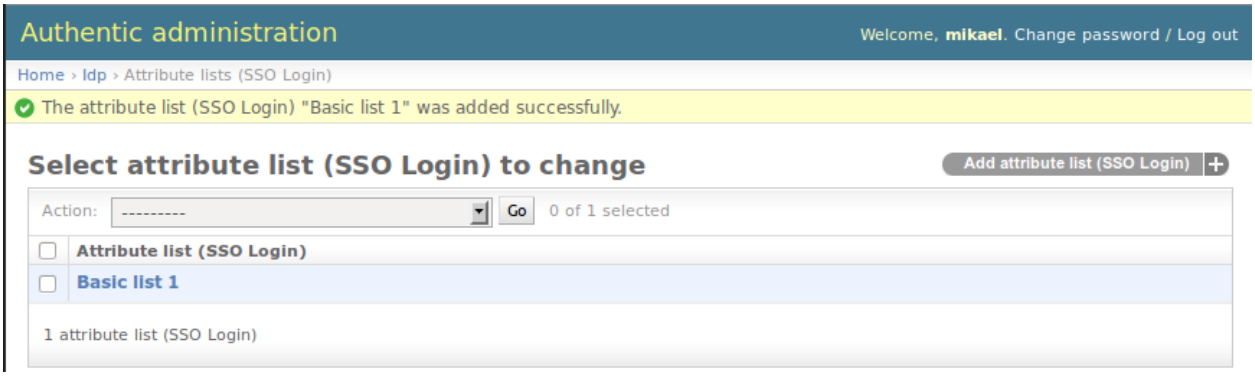


Create a named list of attribute items

1. Go to http[s]://your.domain.com/admin/idp/attributelist/add/
2. Name the list and add items to list



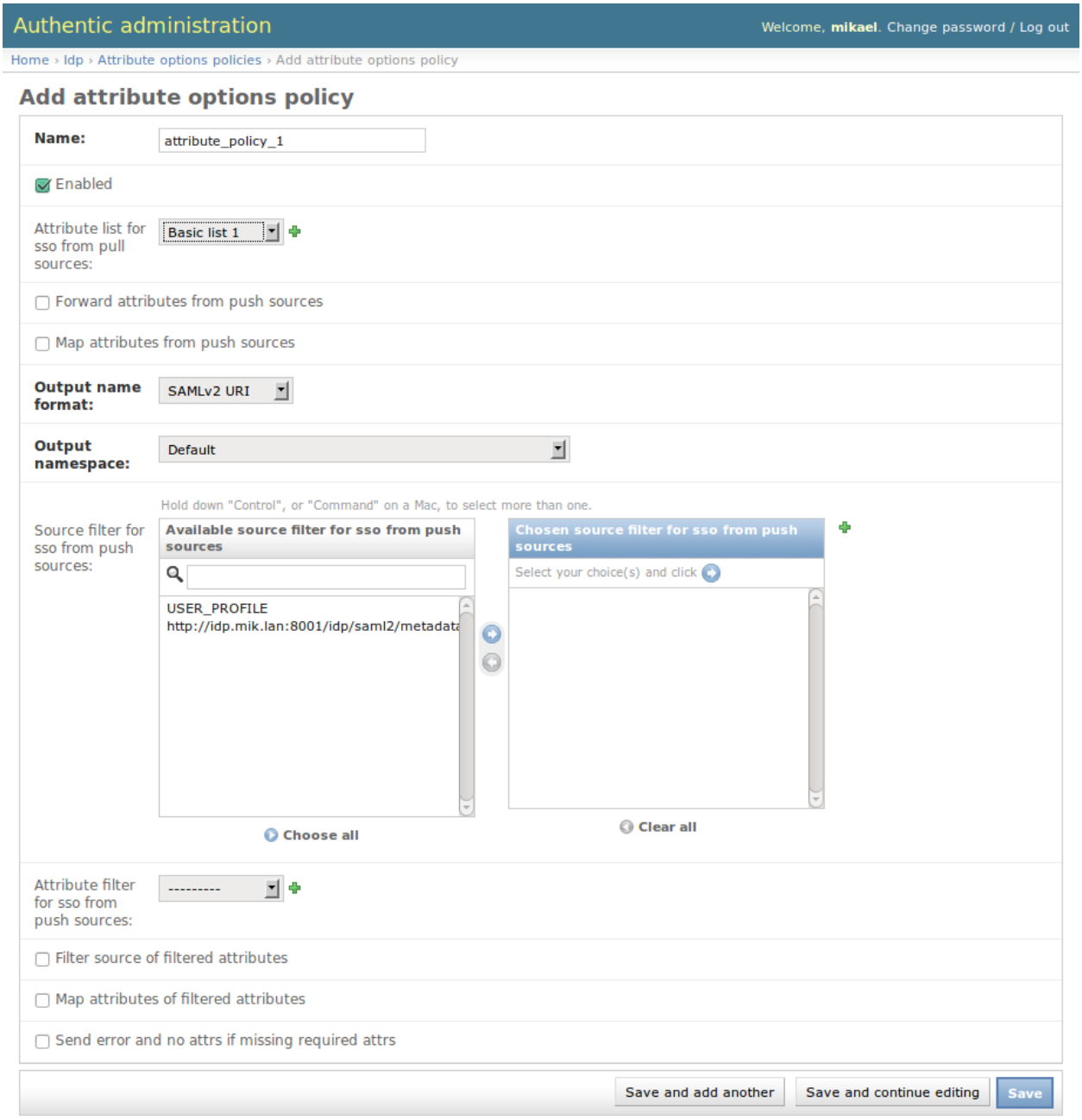
3. Save



Create or modify an attribute policy

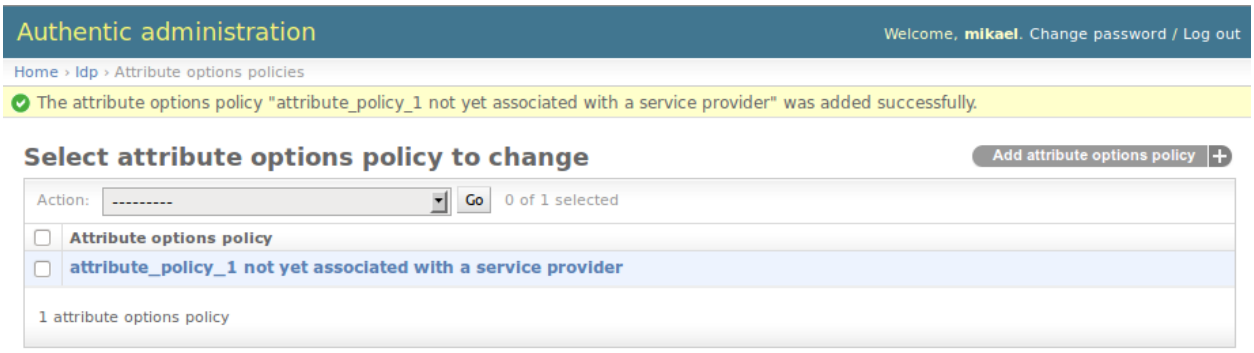
You can create a global policy 'All' or 'Default'. For details, see *How global policies are used in Authentic 2 administration*. Or you can create a regular policy and associate it to a service provider.

1. Go to http[s]://your.domain.com/admin/idp/attributepolicy/add/
2. Add list to the policy


Authentic administration
Welcome, mikael. Change password / Log out
Home › Idp › Attribute options policies › Add attribute options policy
Add attribute options policy
Name:
attribute_policy_1
Enabled
Attribute list for sso from pull sources:
Basic list 1
Forward attributes from push sources
Map attributes from push sources
Output name format:
SAMLv2 URI
Output namespace:
Default
Hold down "Control", or "Command" on a Mac, to select more than one.
Source filter for sso from push sources:
Available source filter for sso from push sources
USER_PROFILE
http://idp.mik.lan:8001/idp/saml2/metadata
Choose all
Chosen source filter for sso from push sources
Select your choice(s) and click
Clear all
Attribute filter for sso from push sources:

Filter source of filtered attributes
Map attributes of filtered attributes
Send error and no attrs if missing required attrs
Save and add another
Save and continue editing
Save

3. Save

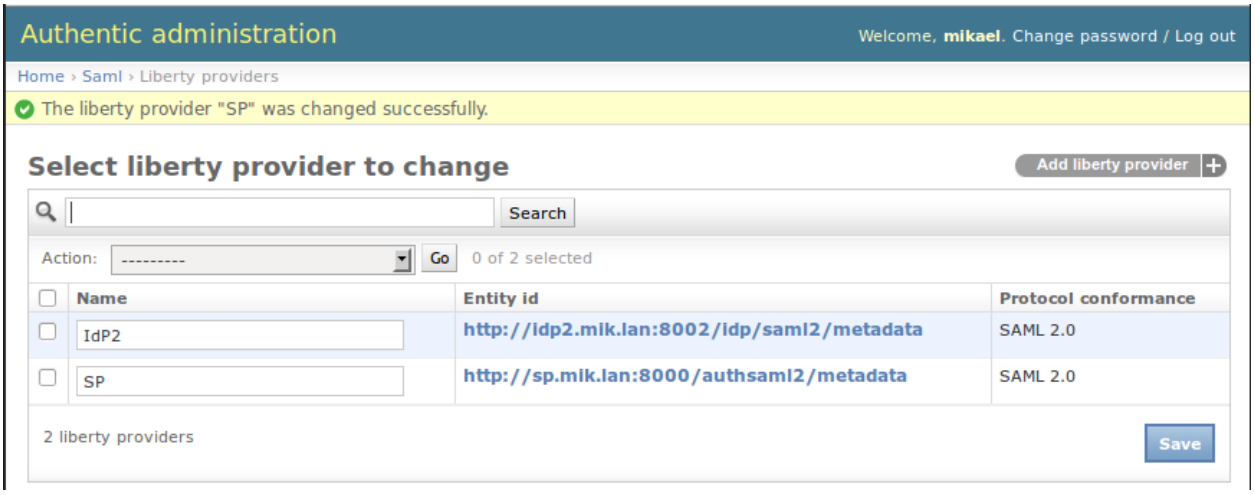

Authentic administration
Welcome, mikael. Change password / Log out
Home › Idp › Attribute options policies
The attribute options policy "attribute_policy_1 not yet associated with a service provider" was added successfully.
Select attribute options policy to change
Add attribute options policy
Action:

Go
0 of 1 selected
Attribute options policy
attribute_policy_1 not yet associated with a service provider
1 attribute options policy

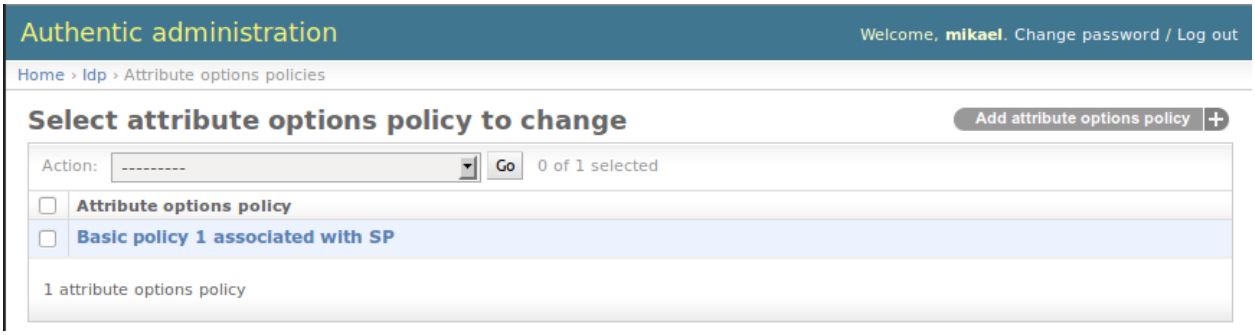
Associate the policy to a service provider

1. Go to http[s]://your.domain.com/admin/saml/libertyprovider/1/
2. Associate the policy to the service provider and **enable it**

3. Save



4. The display name of the policy has changed



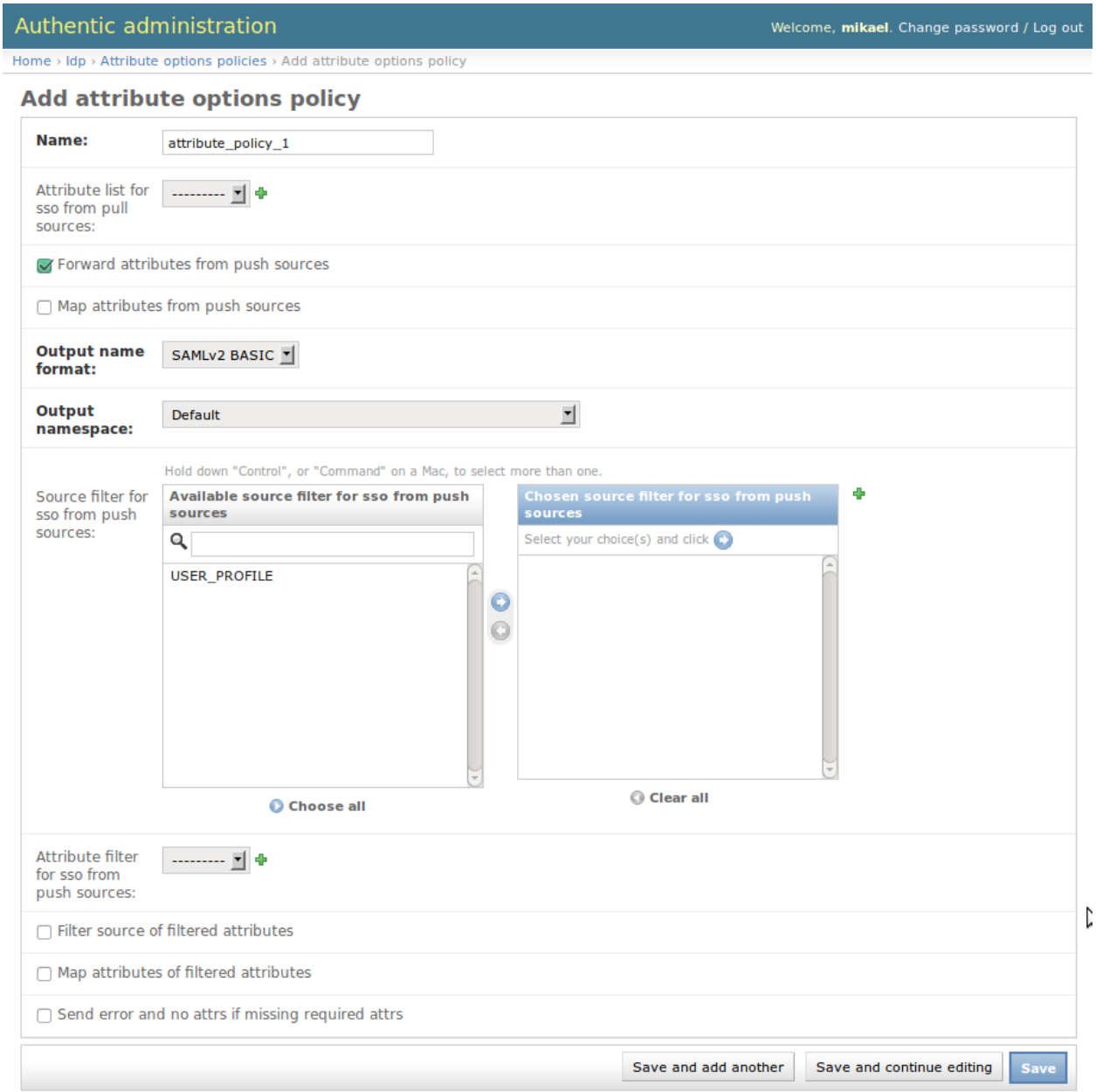
Handle attributes provided by other Identity providers and pushed to SAML2 service providers in SSO response (proxy attributes)

To have these kind of attributes to forward, authentic must be configured as a SAML2 service provider, see the corresponding administration page *Configure Authentic 2 as a SAML2 service provider or a SAML2 proxy*.

Forward all attributes in session without any modification

Create or modify an attribute policy activating the option 'Forward attributes from push sources' and save.

No other option below must be used.



Attach policy to the service provider if it is not yet the case.

No need to deal with namespace here.

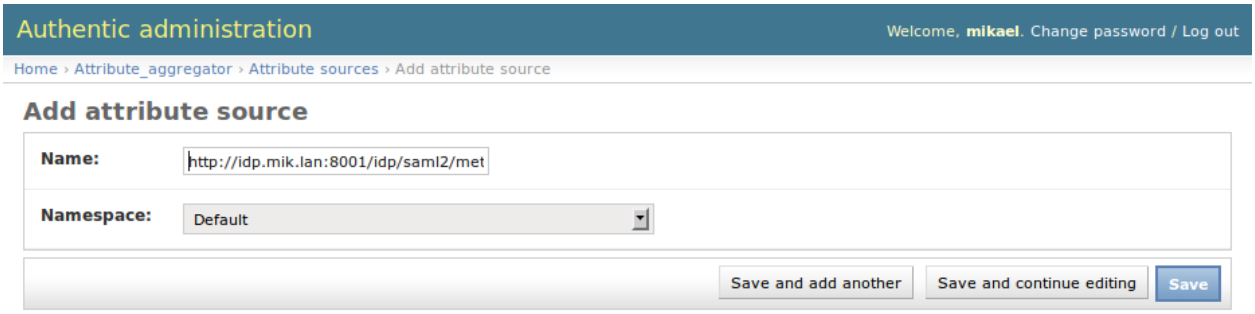
Filter attributes from source only

Here, you want to forward **all** attributes of selected source of attributes.

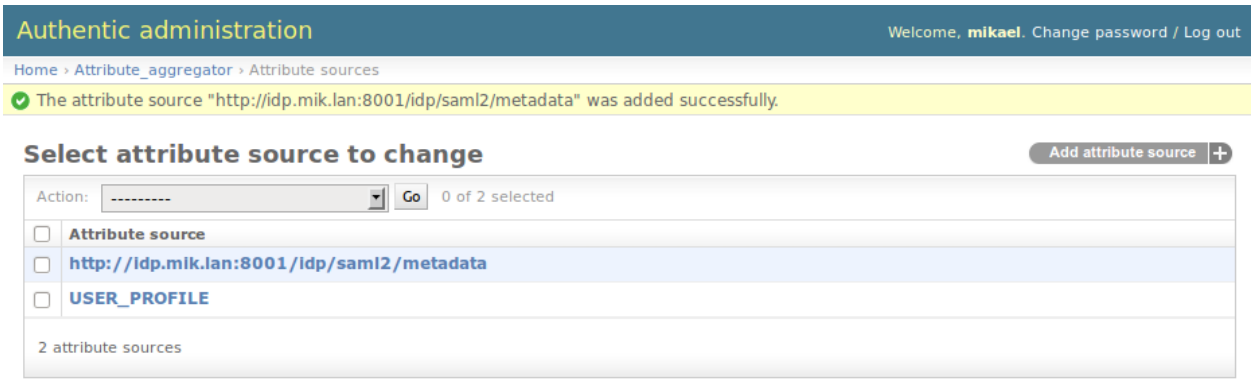
First of all you need to create objects corresponding to the sources of attributes.

The name of the source object must be the entity ID of the SAML2 identity provider.

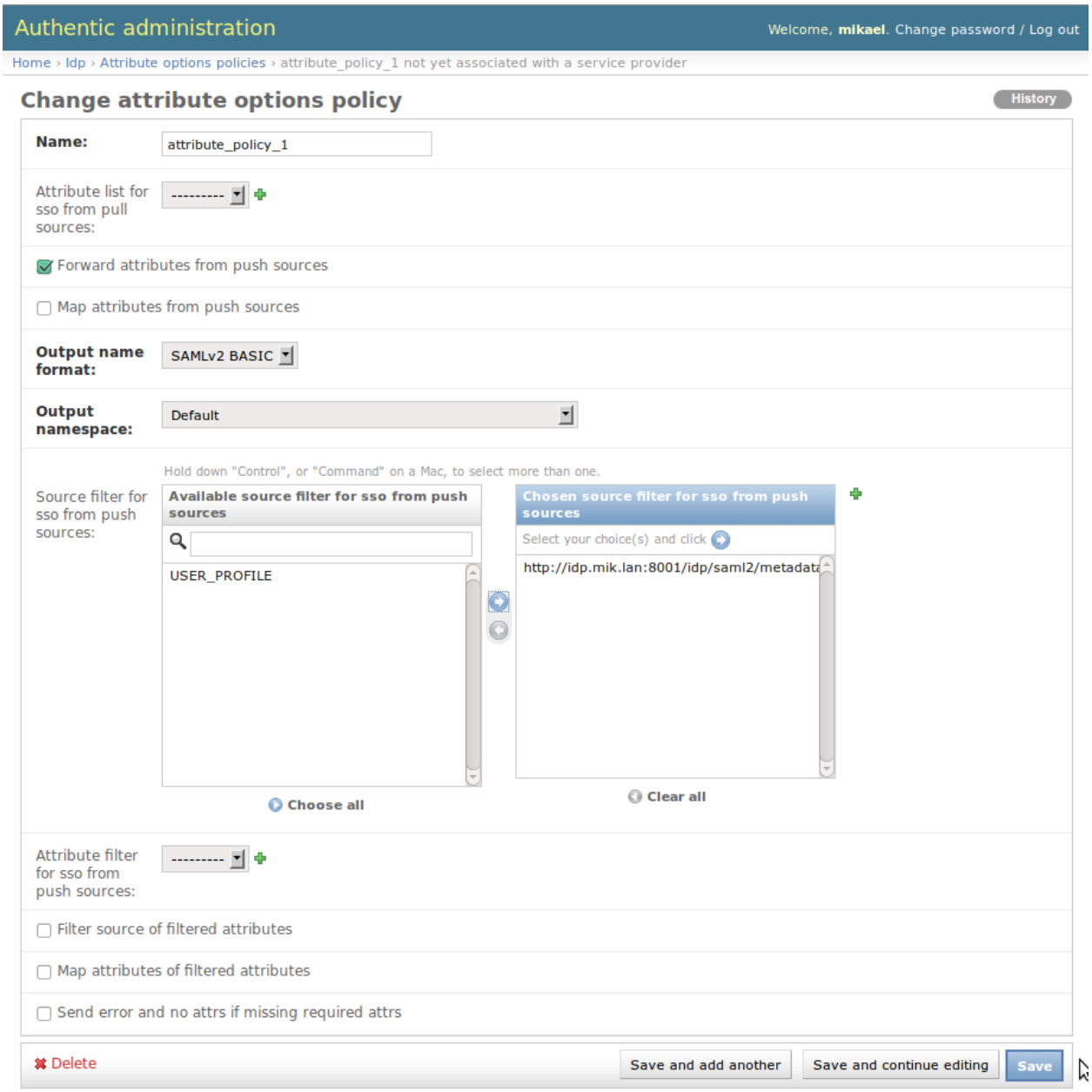
1. Go to http[s]://your.domain.com/admin/attribute_aggregator/attributesource/add/
2. Set the name (No need to change the namespace)



3. Save



Then create or modify an attribute policy activating the option **'Forward attributes from push sources'**. You then select the source you want to forward attributes through the selection box and you save.



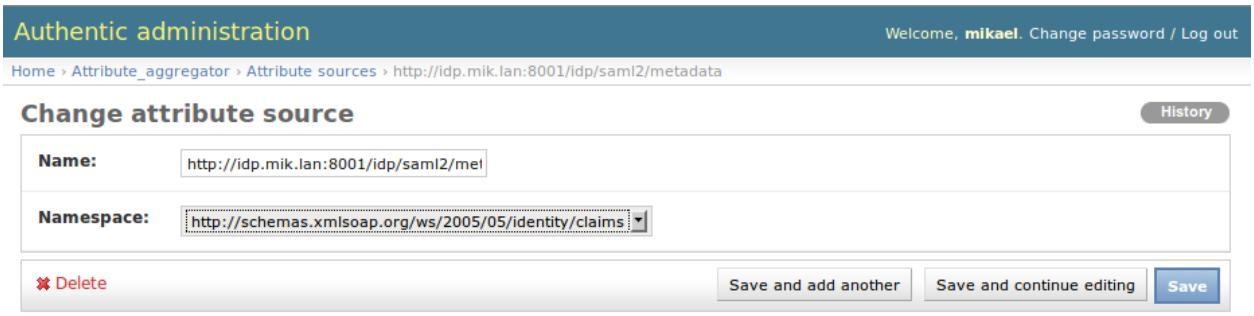
Attach policy to the service provider if it is not yet the case.

No need to deal with namespace here.

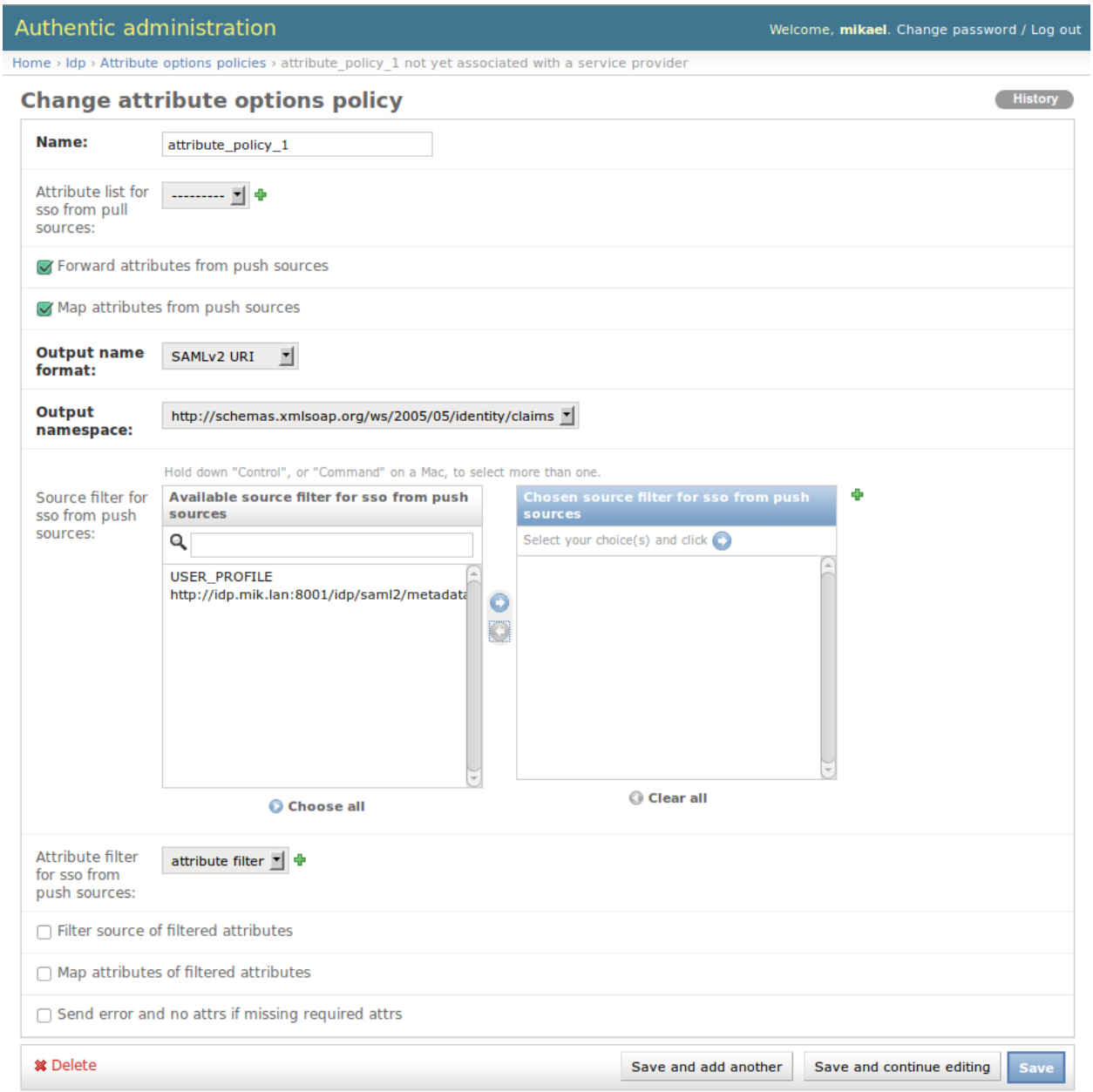
Modify namespace of attributes forwarded when attributes forwarded are not filtered or when filtered according to the source

The system needs to 'recognise the attributes' to perform the mapping. For this, you need to indicate the namespace of attributes received per source if the namespace is not the one of Authentic 2 (X500/LDAP and extensions edu* and supann).

In other words if the source provides attributes in a different namespace, you need to create objects corresponding to the sources of attributes and indicate there the right namespace. By default, the only other supported namespace is http://schemas.xmlsoap.org/ws/2005/05/identity/claims.



Then create or modify an attribute policy activating the options 'Forward attributes from push sources', **'Map attributes from push sources'**. You also choose the output namespace expected with the parameters **'Output name format'** and **'Output namespace'**.



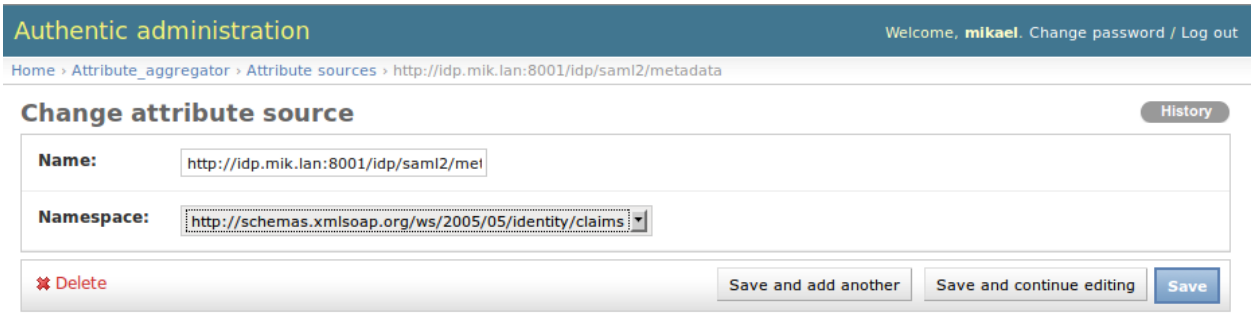
Remind that the default namespace is X500/LDAP + edu* + supann and the only other supported namespace is http://schemas.xmlsoap.org/ws/2005/05/identity/claims.

Attach policy to the service provider if it is not yet the case.

Filter attributes with a list of attributes, with or without choosing the source

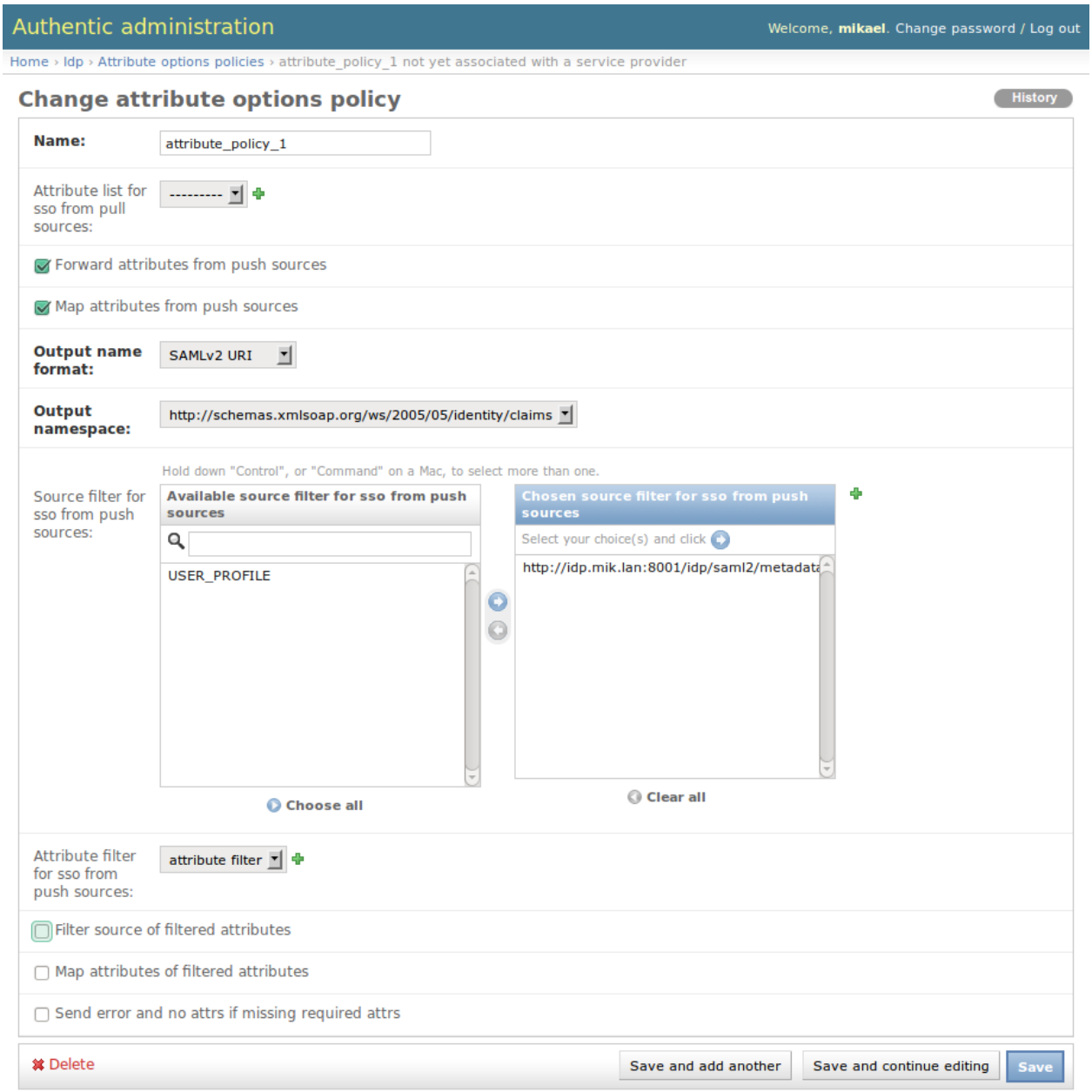
The system needs to 'recognise the attributes' to filter the attributes according to a list of attributes. For this, you need to indicate the namespace of attributes received per source if the namespace is not the one of Authentic 2 (X500/LDAP and extensions edu* and supann).

In other words if the source provides attributes in a different namespace, you need to create objects corresponding to the sources of attributes and indicate there the right namespace. By default, the only other supported namespace is http://schemas.xmlsoap.org/ws/2005/05/identity/claims.

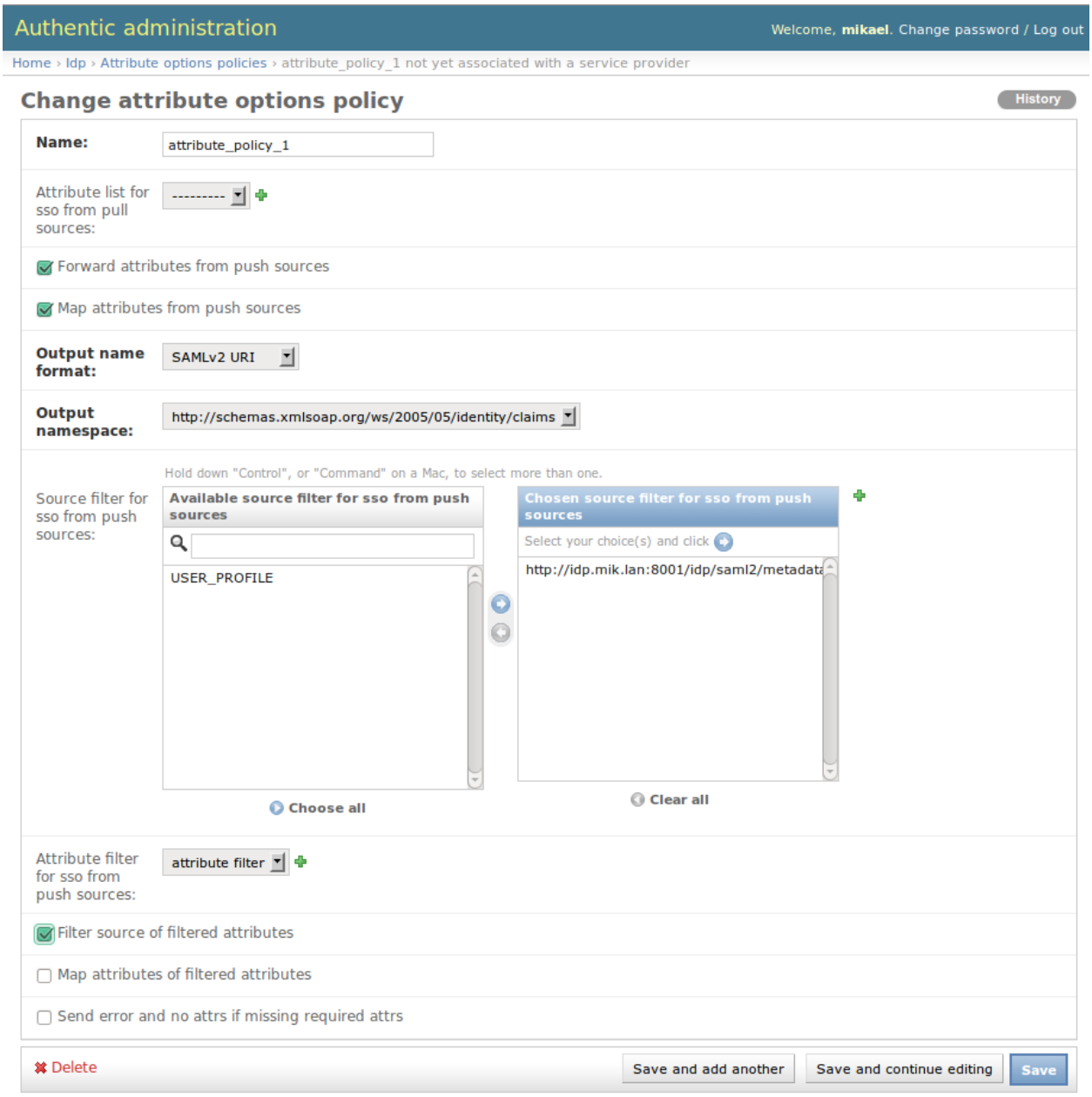


You then create an attribute list as described in section *'Create a named list of attribute items'*.

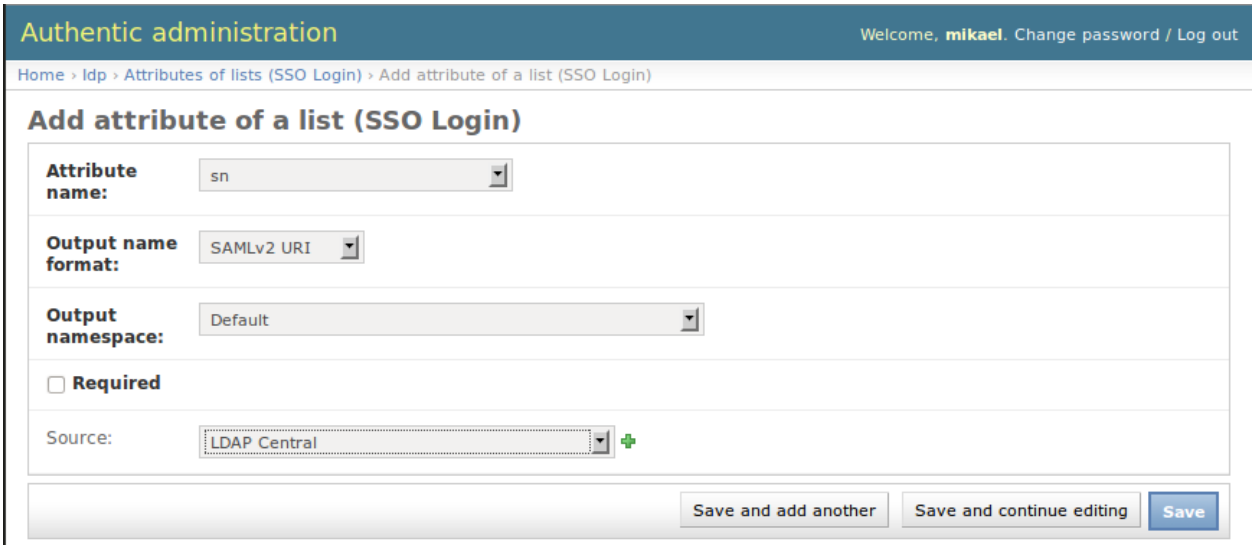
Then create or modify an attribute policy activating the option **'Forward attributes from push sources'**. You then associate the list of attributes.



If you want to also filter according to the source you can configure it as defined in section *'Filter attributes from source only'*. You can also choose to filter with the source indicate per attribute item of the list. For this select the option **'Filter source of filtered attributes'**.

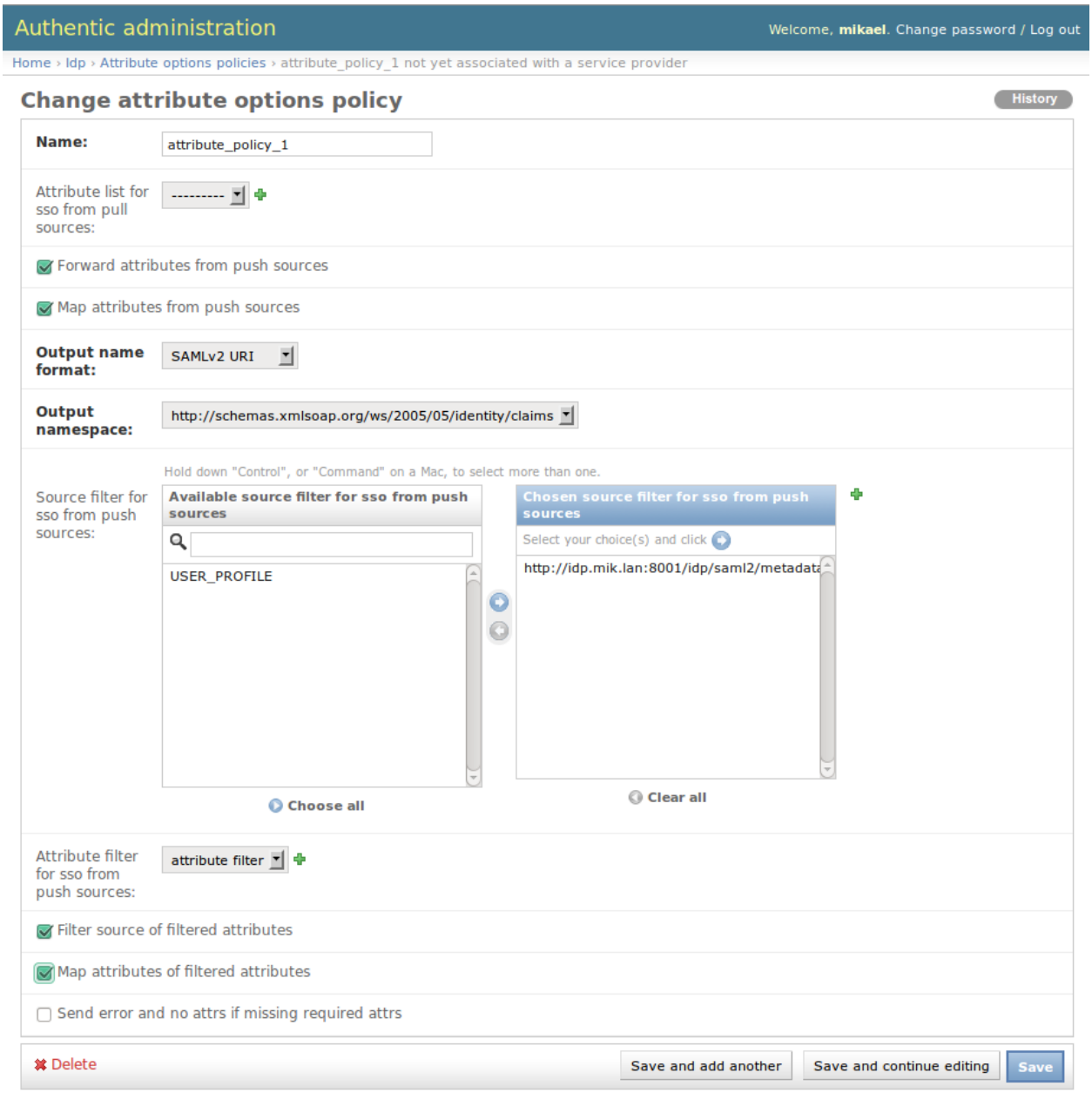

Authentic administration
Welcome, mikael. Change password / Log out
Home › Idp › Attribute options policies › attribute_policy_1 not yet associated with a service provider
Change attribute options policy
History
Name:
attribute_policy_1
Attribute list for sso from pull sources:

Forward attributes from push sources
Map attributes from push sources
Output name format:
SAMLv2 URI
Output namespace:
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
Hold down "Control", or "Command" on a Mac, to select more than one.
Source filter for sso from push sources:
Available source filter for sso from push sources
USER_PROFILE
Choose all
Chosen source filter for sso from push sources
Select your choice(s) and click
http://idp.mik.lan:8001/idp/saml2/metadata
Clear all
Attribute filter for sso from push sources:
attribute filter
Filter source of filtered attributes
Map attributes of filtered attributes
Send error and no attrs if missing required attrs
Delete
Save and add another
Save and continue editing
Save

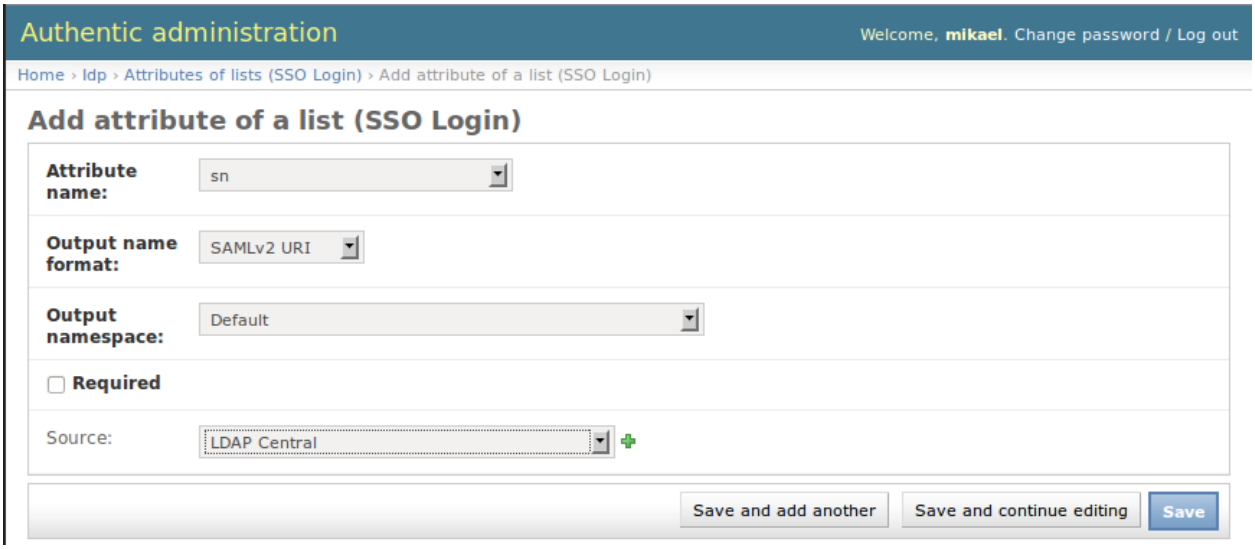


The default name format is URI. You can however change the name format and namespace with the option **'Map attributes from push sources'** and the parameters **'Output name format'** and **'Output namespace'**.

Using the option **'Map attributes of filtered attributes'** the output name format and namespace are the ones indicated per attribute item of the list.


Authentic administration
Welcome, mikael. Change password / Log out
Home › Idp › Attribute options policies › attribute_policy_1 not yet associated with a service provider
Change attribute options policy
History
Name:
attribute_policy_1
Attribute list for sso from pull sources:

Forward attributes from push sources
Map attributes from push sources
Output name format:
SAMLv2 URI
Output namespace:
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
Hold down "Control", or "Command" on a Mac, to select more than one.
Source filter for sso from push sources:
Available source filter for sso from push sources
USER_PROFILE
Choose all
Chosen source filter for sso from push sources
Select your choice(s) and click
http://idp.mik.lan:8001/idp/saml2/metadata
Clear all
Attribute filter for sso from push sources:
attribute filter
Filter source of filtered attributes
Map attributes of filtered attributes
Send error and no attrs if missing required attrs
Delete
Save and add another
Save and continue editing
Save



Push manually (writing bits of code) attributes to SAML2 service providers in SSO response

In idp/signals.py connect to the add_attributes_to_response signal:

```
add_attributes_to_response.connect(your_function)
```

Your function must return an attribute dictionnary as follows:

```
dic = {}
attributes = {}
attributes[name] = (value1, value2, )
attributes[(name, format)] = (value1, value2, )
attributes[(name, format, nickname)] = (value1, value2, )
dic['attributes'] = attributes
return dic
```

format must be in (lasso.SAML2_ATTRIBUTE_NAME_FORMAT_URI, lasso.SAML2_ATTRIBUTE_NAME_FORMAT_BASIC)

You can use the attributes form the local source and the attributes in the session that are pushed by other identity providers.

Attributes in the session are in:

```
request.session['multisource_attributes']
```

See the page *Attributes in session pushed by third SAML2 identity providers*.

If you want to use local source of attributes and use mapping capabilities of the UserAttributeProfile see the page *Attribute management machinery explained (attribute_aggregator module)*. Use the file idp/attributes.py as an exemple.

1.16.3 Modifying supported namespaces and attribute name mappings

The mapping is defined in the file attribute_aggregatore/mapping.py

The manual modification of this file is necessary to extend the default schema and mappings.

Add new namespaces in ATTRIBUTE_NAMESPACES.

To extend the default schema add key/value in ATTRIBUTE_MAPPING, for instance:

```
"displayName": {
    "oid": "2.16.840.1.113730.3.1.241",
    "display_name": _("displayName"),
    "type": "http://www.w3.org/2001/XMLSchema#string",
    "syntax": "1.3.6.1.4.1.1466.115.121.1.15",
},
```

Add mapping of attribute name extending attribute entries in ATTRIBUTE_MAPPING, for instance:

```
"sn": {
    "oid": "2.5.4.4",
    "display_name": _("sn surname"),
    "alias": ['surname'],
    "profile_field_name": 'last_name',
    "type": "http://www.w3.org/2001/XMLSchema#string",
    "namespaces": {
        "http://schemas.xmlsoap.org/ws/2005/05/identity/claims": {
            "identifiers":
                [
            "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname",
                ],
            "friendly_names":
                [
            "Last Name",
                ],
        }
    }
},
```

1.17 Attribute management machinery explained (attribute_aggregator module)

1.17.1 Attribute aggegrator module

The core attribute management is based on the attribute aggregator module.

Intro

Attribute aggregator provides a main Model class called UserAttributeProfile, functions to load attributes and extract attributes.

The mapping between attribute namespaces is built-in and depends on a unique file (mapping.py).

A main schema is defined and is based on LDAP/X500 for naming. The support of http://schemas.xmlsoap.org/ws/2005/05/identity/claims is partly complete.

Source of attributes are connected with attribute loading functions using signals.

FAQ

Why not use the Django User profile?

The django user profile needs to define attributes as class attributes and then support form filling or mapping with LDAP.

That is useful and may be used, especially because the profile can be used as a source of attribute to load in the attribute_aggregator profile.

The attribute_aggregator profile allow to load multivalued attributes from any source supported (LDAP, Django profile and attributes in Django session for now) from any namespace defined in mapping.py (LDAP/X500 and claims for now).

The profile can be loaded giving a source or a list of attribute, or can be from any known source, or with a dictionnary.

Attributes can be extracted with many functions in any namespace supported.

Quick explanation

The schema is defined in mapping.py and is made of definitions like this:

```
"sn": {
    "oid": "2.5.4.4",
    "display_name": _("sn surname"),
    "alias": ['surname'],
    "profile_field_name": 'last_name',
    "type": "http://www.w3.org/2001/XMLSchema#string",
    "namespaces": {
        "http://schemas.xmlsoap.org/ws/2005/05/identity/claims": {
            "identifiers":
                [
        "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname",
                ],
            "friendly_names":
                [
            "Last Name",
                ],
        }
    }
},
```

The profile store all the data in a text field taht contains a cPickle list of instances of the class AttributeData.

The profile is attached to a user and then can be created or loaded with:

```
profile = load_or_create_user_profile(user=user)
```

User may be None to create a temporary profile for an anonymous user. But that need a DB cleaning function not implemented.

The model *UserAttributeProfile*

The model 'UserAttributeProfile' can be attached to a user and then persist (as a Model).

When the profile is loaded, all data stored are removed expect if the the data has an expiration date later.

The profile provide several methods to store and extract attributes.

All the methods to add attributes are based on a main one accepting a dictionnary of attribute is parameters 'load_by_dic()'. The other methods ('load_listed_attributes()', 'load_greedy()') send a signal with a list of attributes (listed_attributes_call) or not (any_attributes_call) to grab a dictionnary. The list is given with the definition name, oid or friendly name of the attribute in the system namespace.

Into the dictionnary, attributes are given with their name, oid or friendly name in the default namespace or with their name in a namepsace. An expiration date can also be given (ISO8601 format), if none, attribute will be deleted at next profile loading. The dictionnary format is as follows:

```
attributes = dict()
data_from_source = list()
a1 = dict()
    a1['oid'] = definition_name
Or
    a1['definition'] = definition_name
        definition may be the definition name like 'gn'
        or an alias like 'givenName'
Or
    a1['name'] = attribute_name_in_ns
    a1['namespace'] = ns_name
a1['expiration_date'] = date
a1['values'] = list_of_values
data_from_source.append(a1)
...
data_from_source.append(a2)
attributes[source_name] = data_from_source
```

Getters are defined to extract data from a profile. Only AttributeData instances are extracted that assume that any attribute namespace can be used.

- get_data_of_definition(definition)

Return a list of AttributeData instances corresponding to the definition given.

- get_freshest_data_of_definition(definition)

Return the freshest AttributeData instance. If multiple with no or same exp date, random. Should use the creation date soon.

- get_data_of_source

Return a list of AttributeData instances corresponding to the source given.

- get_data_of_source_by_name

Idem but source name is given, not a Source instance.

- get_data_of_definition_and_source

Return a list of AttributeData instances corresponding to the definition and source given.

- get_data_of_definition_and_source_by_name

Idem but source name is given, not a Source instance.

1.17.2 SAML2 attribute representation in assertions

SAML2 attribute profile (saml-profiles-2.0-os - Section 8) defines two kind of attribute element syntax in the attribute statement of assertions, also called *name format*:

- BASIC:

  ```
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
  ```

- URI:

```
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
```

URI should be used when attributes have "universally" known unique names like OID.

Example:

```
<saml:Attribute NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
    Name="FirstName">
    <saml:AttributeValue xsi:type="xs:string">By-Tor</saml:AttributeValue>
</saml:Attribute>

<saml:Attribute
    xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    Name="urn:oid:2.5.4.42" FriendlyName="givenName">
    <saml:AttributeValue xsi:type="xs:string"
        x500:Encoding="LDAP">Steven</saml:AttributeValue>
</saml:Attribute>
```

BASIC

Two <Attribute> elements refer to the same SAML attribute if and only if the values of their Name XML attributes are equal in the sense of Section 3.3.6 of [Schema2].

No additional XML attributes are defined for use with the <Attribute> element.

The schema type of the contents of the <AttributeValue> element MUST be drawn from one of the types defined in Section 3.3 of [Schema2]. The xsi:type attribute MUST be present and be given the appropriate value.

X.500/LDAP Attribute Profile (URI)

Extracted from the SAML2 core specifications

Two <Attribute> elements refer to the same SAML attribute if and only if their Name XML attribute values are equal in the sense of [RFC3061]. The FriendlyName attribute plays no role in the comparison.

Directory attribute type definitions for use in native X.500 directories specify the syntax of the attribute using ASN.1 [ASN.1]. For use in LDAP, directory attribute definitions additionally include an LDAP syntax which specifies how attribute or assertion values conforming to the syntax are to be represented when transferred in the LDAP protocol (known as an LDAP-specific encoding). The LDAP-specific encoding commonly produces Unicode characters in UTF-8 form. This SAML attribute profile specifies the form of SAML attribute values only for those directory attributes which have LDAP syntaxes. Future extensions to this profile may define attribute value formats for directory attributes whose syntaxes specify other encodings.

To represent the encoding rules in use for a particular attribute value, the <AttributeValue> element MUST contain an XML attribute named Encoding defined in the XML namespace urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500.

For any directory attribute with a syntax whose LDAP-specific encoding exclusively produces UTF-8 character strings as values, the SAML attribute value is encoded as simply the UTF-8 string itself, as the content of the <AttributeValue> element, with no additional whitespace. In such cases, the xsi:type XML attribute MUST be set to xs:string. The profile-specific Encoding XML attribute is provided, with a value of LDAP.

The AttributeData instances have a field expiration_data. It the profile exists, obsolete data are removed at loading.

1.17.3 When authentic 2 deals with attributes and needs mapping?

Authentic 2 behaves as an attribute provider: * At the SSO login * When an attribute request is received

Authentic requests (e.g. by soap) are not yet supported.

When Authentic 2 behaves as an attribute provider at SSO login

At a SSO request, just before responding to the service provider, the saml2 idp module sends the signal 'add_attributes_to_response' giving the SP entity ID.

The signal is connected to the function 'provide_attributes_at_sso()' in charge of providing the attributes at the SSO for this SP.

Attributes sources are of two kinds. The first ones are the sources that can be requested by the IdP with a syncrhonous binding without user intercations. These sources are called pull sources. They are for now limited to LDAP sources. The other ones are sources are asyncrhonous bindings, usually requiring user interactions. These sources are called push sources. They are now limited to the attributes provided at SSO requests when the IdP acts as a SAML2 SP. There attributes are put/found in the Django session.

Each source in the system is declared with an instance of the AttributeSource model. We'll see later that to forward attributes of push sources it is not necessary that a source is declared in some circumstances.

To manage these sources an attribute policy is attached to services providers. Then the service provider model must be extended with a attribute attributes_at_sso_policy. The service provider must send the signal 'add_attributes_to_response'.

The implementation is actually done for SAML2 providers.

In such a policy attributes from pull and push sources are treated differently.

For pull sources, a list of attributes is indicated. Either an attribute is searched in all the pull sources and whatever attribute value found is returned. Or each attribute is indicated with a source. With each attribute is indicated the output format and namespace.

The policy may also indicate that all the attributes in the Django session must be forwarded. Then, no AttributeSource instance is required. All the attributes are then forwarded without treating input namespace considerations. When an AttributeSource instance is found, the input namespace of this source is considered. An option can then be set to tell that the output format and namespace must be taken. A list of attribute can also be given. This list can be use to filter attributes to forward without or without taking care of the source. The output namespace and format can also be trated per attribute.

If the namespace is default, the attribute names will be taken from the system namespace. In BASIC the name will be the definition name. In URI, the Name will be the OID in urn format and the friendly name will be the definition name. If a namespace is given, the first identifier of this attribute is taken as Name in BASIC. In URI, the same and the first friendly name is taken.

```
class LibertyServiceProvider(models.Model):
    ...
    attribute_policy = models.ForeignKey(AttributePolicy,
            verbose_name=_("Attribute policy"), null=True, blank=True)

class AttributePolicy(models.Model):
    # List of attributes to provide from pull sources at SSO Login.
    # If an attribute is indicate without a source, from any source.
    # The output format and namespace is given by each attribute.
    attribute_list_for_sso_from_pull_sources = \
        models.ForeignKey(LibertyAttributeMap,
        related_name = "attributes of pull sources",
        blank = True, null = True)

    # Set to true for proxying attributes from pull sources at SSO Login.
    # Attributes are in session.
```

```
    # All attributes are forwarded as is except if the parameter
    # 'attribute_list_for_sso_from_push_sources' is initialized
    forward_attributes_from_pull_sources = models.BooleanField(default=False)

    # Map attributes in session
    # forward_attributes_in_session must be true
    # At False, all attributes are forwarded as is
    # At true, look for the namespace of the source for input, If not found,
    # system namespace. Look for the options attribute_name_format and
    # output_namespace of the attribute policy for output.
    map_attributes_from_pull_sources = models.BooleanField(default=False)

    # ATTRIBUTE_VALUE_FORMATS[0] =>
    #    (lasso.SAML2_ATTRIBUTE_NAME_FORMAT_BASIC, 'SAMLv2 BASIC')
    output_name_format = models.CharField(max_length = 100,
        choices = ATTRIBUTE_VALUE_FORMATS,
        default = ATTRIBUTE_VALUE_FORMATS[0])

    #ATTRIBUTES_NS[0] => ('Default', 'Default')
    output_namespace = models.CharField(max_length = 100,
        choices = ATTRIBUTES_NS, default = ATTRIBUTES_NS[0])

    # Filter attributes pushed from source.
    source_filter_for_sso_from_push_sources = \
        models.ManyToManyField(AttributeSource,
        related_name = "attributes of pull sources",
        blank = True, null = True)

    # List of attributes to filter from pull sources at SSO Login.
    attribute_filter_for_sso_from_push_sources = \
        models.ForeignKey(LibertyAttributeMap,
        related_name = "attributes of pull sources",
        blank = True, null = True)

    # The sources of attributes of the previous list are considered.
    # May be used conjointly with 'source_filter_for_sso_from_push_sources'
    filter_source_of_filtered_attributes = models.BooleanField(default=False)

    # To map the attributes of forwarded attributes with the defaut output
    # format and namespace, use 'map_attributes_from_pull_sources'
    # Use the following option to use the output format and namespace
    # indicated for each attribute.
    map_attributes_of_filtered_attributes = models.BooleanField(default=False)


    # Set to true to take in account missing required attributes
    send_error_and_no_attrs_if_missing_required_attrs = \
        models.BooleanField(default=False)

    class Meta:
        verbose_name = _('attribute options policy')
        verbose_name_plural = _('attribute options policies')


class AttributeList(models.Model):
    name = models.CharField(max_length = 40, unique = True)
    attributes = models.ManyToManyField(AttributeItem,
        related_name = "attributes of the list",
```

```
        blank = True, null = True)


class AttributeItem(models.Model):
    attribute_name = models.CharField(max_length = 100, choices = ATTRIBUTES,
        default = ATTRIBUTES[0])
    # ATTRIBUTE_VALUE_FORMATS[0] =>
    #    (lasso.SAML2_ATTRIBUTE_NAME_FORMAT_BASIC, 'SAMLv2 BASIC')
    output_attribute_name_format = models.CharField(max_length = 100,
        choices = ATTRIBUTE_VALUE_FORMATS,
        default = ATTRIBUTE_VALUE_FORMATS[0])
    #ATTRIBUTES_NS[0] => ('Default', 'Default')
    output_namespace = models.CharField(max_length = 100,
        choices = ATTRIBUTES_NS, default = ATTRIBUTES_NS[0])
    required = models.BooleanField(default=False)
    source = models.ForeignKey(AttributeSource, blank = True, null = True)
```

A list of attributes can also be taken from the service provider metadata and added to 'attribute_list_for_sso_from_pull_sources'. The namespace may be extracted from the metadata. This namespace is then used to look for the corresponding definition and then to provide the attribute in the right namespace. Read attributes from metadata is not yet supported.

For the attributes of pull sources, once the list of attributes is defined, They are loaded in the user profile.

As explained before the attribute_aggregator loading function send signals to grab dictionnary of attributes. Up to know, only the ldap loading function are connected to these signals. The namespace of LDAP sources is assumed to be the same as the system namespace. There is here then no mapping needed. Other kind of sources than LDAP can be defined in attribute aggregator.

To grab attributes from a LDAP the user dn in the LDAP or at least a local identifier in the LDAP is required. For this purpose, each user has alias associated with LDAP source. These aliases must their DN in the LDAP. When the authentication LDAP backend will be taken in account, the dn will be taken direclty from the user Model instance.

Each LDAP sources are declared with the binding parameters. The LDAP namespace is always 'Default'.

If an attribute to load is not found and is required the answer should report an error (Not yet implemented).

Attributes in response can also be provided with other means than from an LDAP source. Attributes can be put in the user Django session and then loaded in the profile. An option of the service provier indicate if attributes in the session must be provided to the service provider.

To have the attribute loaded from the session, they must be provided in the session as follows: request.session['attributes'][source_name] = list()

The source_name must be the name of an existing instance of an 'AttributeSource'. Such an instance contains a field namespace indicating the namespace of attributes.

This is currently implemented only for the SAML2 service provider module of authentic2. Authsaml2, the SP module, parse the assertion and put the attributes in the session.

Then, Authentic 2 can be used as a SAML2 proxy forwarding attributes in assertion, eventually doing a namespace mapping. For this, the option forward attributes in sesion must be set (by default False).

1.18 Attributes in session pushed by third SAML2 identity providers

When an assertion is received, assertion data, including attributes, are pushed in the Django session dictionnary.

It leads to the creation of the following dictionnary:

```
request.session['multisource_attributes']
```

The keys of the dictionnary are the source names, i.e. the entity Id for SAML2 identity providers.

The values are list of data extracted from assertions. Indeed, this is done to store multiple assertion received from a same source in a same Django session:

```
request.session['multisource_attributes'] \
    [source_name] = list()
```

The items of this list are dictionnaries with the keys 'certificate_type' and 'attributes'.

For a saml2 assertion, all the keys are:

```
a8n['certificate_type'] = 'SAML2_assertion'
a8n['nameid'] = ...
a8n['subject_confirmation_method'] = ...
a8n['not_before'] = ...
a8n['not_on_or_after'] = ...
a8n['authn_context'] = ...
a8n['authn_instant'] = ...
a8n['attributes'] = attrs
```

a8n['attributes'] has the following structure:

```
attributes = {}
attributes[name] = (value1, value2, )
attributes[(name, format)] = (value1, value2, )
attributes[(name, format, nickname)] = (value1, value2, )
a8n['attributes'] = attributes
```

1.19 Consent Management in Authentic 2

1.19.1 What is the SAML2 federation consent aka account linking consent?

At the first single sign on process on the identity provider side, the user may be asked if she agrees to federation its local account with the remote account on the service provider side.

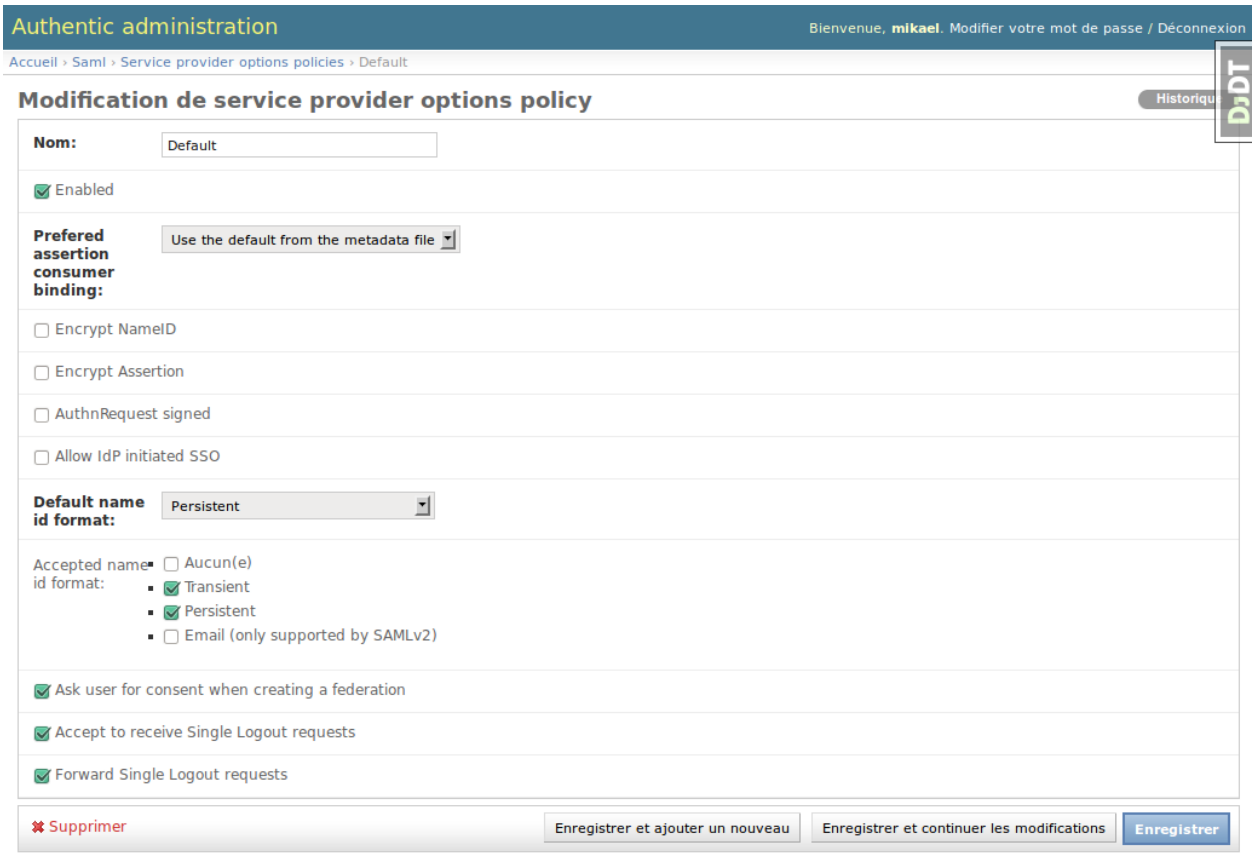
The account linking also called a federation means that the nameID is persistent and will link the two accounts. This signed identifier allows to the service provider to login the user without reauthentication during the following single sign on process.

How the consent is collected is determined by the identity provider. The service provider receives in the authnRequest the consent attribute indicating how the user consent was managed.

1.19.2 Account linking consent management on the identity provider side

The consent is managed per service provider according to the options policy that applies to the service provider.

The parameter 'Ask user for consent when creating a federation' determine if the user consent must be asked to the user.



Take care that is the identity provider provides the service provider with a transient nameID, there is no account linking, so there is no need for a consent.

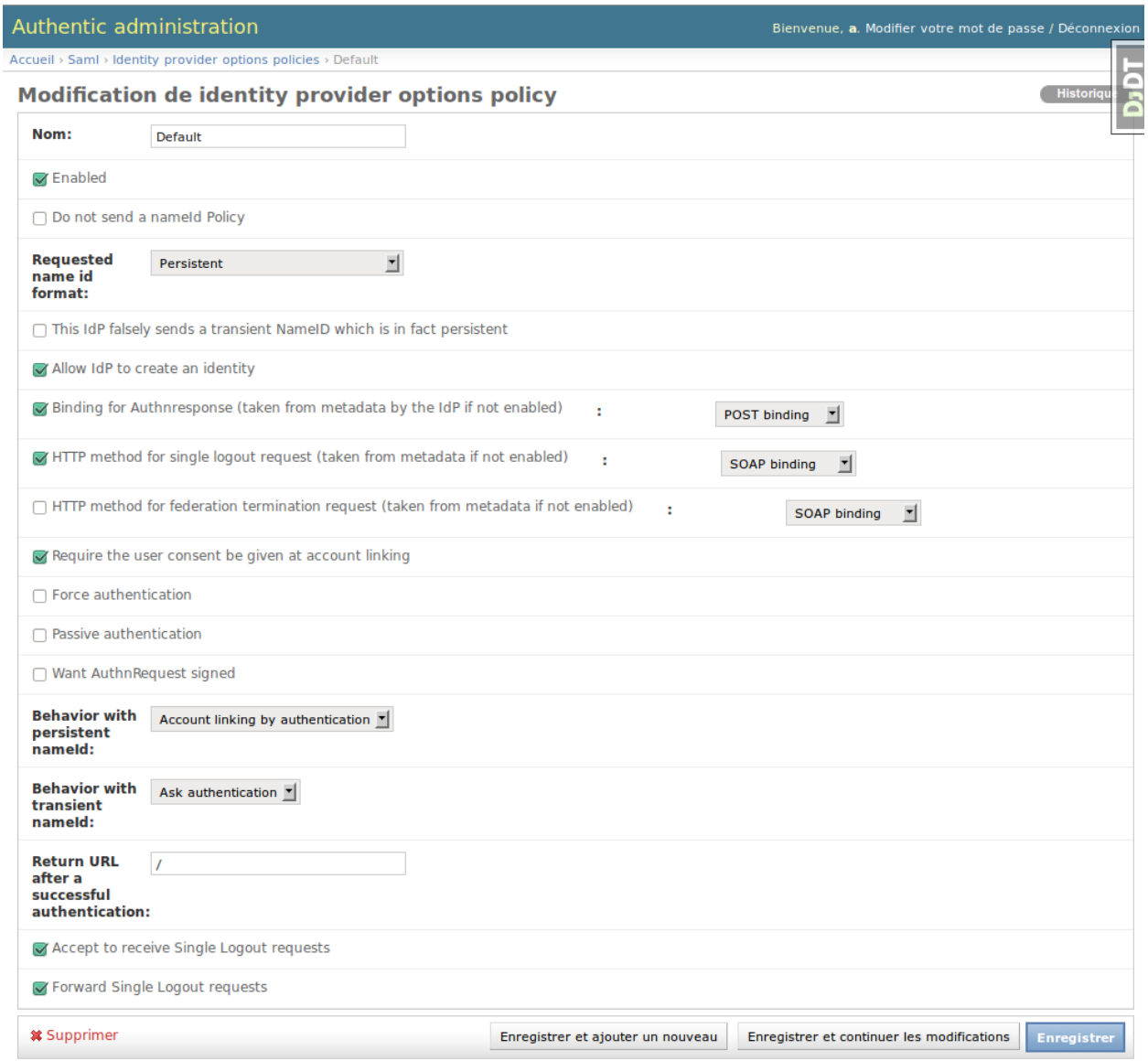
The user consent is only asked once. In other words, if the user already has a federation, the consent won't be asked anymore.

If the policy requires the user consent, this can be bypassed using the signal 'avoid_consent'.

1.19.3 Account linking consent management on the service provider side

The service provider may refuse a valid single sign on if the user consent was not asked.

The parameter 'Require the user consent be given at account linking' of the identity provider options policy determine the service provider behavior.



1.19.4 How manage attribute forwarding consent?

If there is no attribute policy associate with a service provider, no attribute is forwarded.

When an attribute policy applies you can configure the consent rules per service provider.

The choices are:

- Don't ask the user consent
- Ask the consent in all-or-nothing mode
- Allow attribute selection

To ask the user consent, tick the parameter 'Ask the user consent before forwarding attributes' of the attribute policy that applies to the service provider.

To allow the attribute selection on the attribute consent page, tick the parameter 'Allow the user to select the forwarding attributes'.


Authentic administration
Bienvenue, mikael. Modifier votre mot de passe / Déconnexion
Accueil › Idp › Attribute options policies › attribute_policy_1
Modification de attribute options policy
Historiqu
Name:
attribute_policy_1
Enabled
Ask the user consent before forwarding attributes
Allow the user to select the forwarding attributes
Attribute list for sso from pull sources:
list1
Forward attributes from push sources
Map attributes from push sources
Output name format:
SAMLv2 BASIC
Output namespace:
Default
Maintenez appuyé « Ctrl », ou « Commande (touche pomme) » sur un Mac, pour en sélectionner plusieurs.
Source filter for sso from push sources:
source filter for sso from push sources disponible(s)
LDAP1
source filter for sso from push sources choisi(es)
Sélectionnez un ou plusieurs choix et cliquez
Tout choisir
Tout enlever
Attribute filter for sso from push sources:

Filter source of filtered attributes
Map attributes of filtered attributes
Send error and no attrs if missing required attrs
Supprimer
Enregistrer et ajouter un nouveau
Enregistrer et continuer les modifications
Enregistrer

CHAPTER

TWO

COPYRIGHT

Authentic and Authentic 2 are copyrighted by Entr'ouvert and are licensed through the GNU AFFERO GENERAL PUBLIC LICENSE, version 3 or later. A copy of the whole license text is available in the COPYING file.

The OpenID IdP originates in the project django_openid_provider by Roman Barczyski, which is under the Apache 2.0 licence. This imply that you must distribute authentic2 under the AGPL3 licence when distributing this part of the project which is the only AGPL licence version compatible with the Apache 2.0 licence.

The Documentation is under the licence Creative Commons CC BY-SA 2.0.