kisee

Sep 02, 2021

Contents:

1 Installing **1**
- 1.1 Installing kisee . . . 1
- 1.2 Configuring HTTPS using nginx with certbot . . . 2
- 1.3 Testing your instance . . . 3

2 Features **5**
- 2.1 Overview . . . 5
- 2.2 The backend interface . . . 5

3 Configuration File **7**
- 3.1 Example . . . 7
- 3.2 Sentry . . . 8

4 Kisee API **9**

5 Contributing **11**
- 5.1 Quickstart . . . 11
- 5.2 Building the API doc . . . 11
- 5.3 Internals . . . 12
- 5.4 Releasing . . . 12

6 FAQ **13**
- 6.1 Can I use Kisee to query an OAuth2 service? . . . 13
- 6.2 Does Kisee implement groups? . . . 13
- 6.3 Does Kisee implement impersonation? . . . 13
- 6.4 Does Kisee expose self-service registration? . . . 13
- 6.5 Does Kisee expose a password reset feature? . . . 14

7 Indices and tables **15**

CHAPTER 1

Installing

1.1 Installing kisee

First start by building a venv, let's say `/tmp/kisee` for the example, but please find it a better place:

```
python3 -m venv /tmp/kisee
```

and activate it:

```
/tmp/kisee/bin/activate
```

To install `kisee`, run:

```
pip install kisee
```

Quickstart a settings file:

```
kisee-quickstart
```

Run it once manually to test it:

```
kisee  # or python -m kisee
```

This will start a server on port 8140, you can kill it and now configure systemd to start it.

In a file like `/etc/systemd/system/kisee.service`, copy:

```
[Unit]
Description=Kisee
After=network.target

[Service]
Type=simple
ExecStart=/tmp/kisee/bin/python -m kisee
```

(continues on next page)

(continued from previous page)

```
WorkingDirectory=/home/kisee/kisee-19.07.0/

Restart=on-abort
User=kisee
Group=kisee

[Install]
WantedBy=multi-user.target
```

Then reload systemd config, enable it and start it:

```
systemctl daemon-reaload
systemctl enable kisee
systemctl start kisee
```

1.2 Configuring HTTPS using nginx with certbot

Using `nginx` as a front-end for `kisee` may be a good idea, typically at least for HTTPS decapsulation.

First install nginx and certbot:

```
apt install nginx certbot python3-certbot-nginx
```

First generate a nice dhparam if needed:

```
[ -f /etc/ssl/certs/dhparam.pem ] || openssl dhparam -out /etc/ssl/certs/dhparam.pem␣
↪4096
```

Make sure your domain resolves correcly to the machine, and generate the certificate (replace EXAMPLE.COM in the command, if nginx is running, replace –standalone with –nginx):

```
DOMAIN=EXAMPLE.COM; certbot certonly --cert-name $DOMAIN -n --agree-tos -d $DOMAIN \
  -m admin@$DOMAIN --standalone --rsa-key-size 4096
```

Create the nginx TLS snippet (replace `EXAMPLE.COM`) in `/etc/nginx/snippets/letsencrypt-EXAMPLE.COM.conf` like this

```
ssl_ciphers "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES256-GCM-
↪SHA384:DHE-RSA-AES256-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-
↪SHA256:AES256+EECDH:AES256+EDH";
ssl_protocols TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;
ssl_session_cache shared:ssl_session_cache:10m;
ssl_certificate /etc/letsencrypt/live/EXAMPLE.COM/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/EXAMPLE.COM/privkey.pem;
ssl_dhparam /etc/ssl/certs/dhparam.pem;
```

Make sure `installer` and `authenticator` are set to `nginx` in `/etc/letsencrypt/renewal/EXAMPLE.COM.conf`, in the `[renewalparams]` section. `installer` may not exist, if so create it near the `authenticator` one.

Finally configure nginx like this (again, replace EXAMPLE.COM):

```
server
{
  listen 80;
  server_name EXAMPLE.COM;

  return 301 https://$server_name$request_uri;
}

server
{
  listen 443 ssl;
  server_name EXAMPLE.COM;

  include snippets/letsencrypt-EXAMPLE.COM.conf;

  location /
  {
    proxy_pass http://127.0.0.1:8140;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Forwarded-Protocol $scheme;
  }
}
```

1.3 Testing your instance

To check if your instance is running, just curl on it, over HTTPS from the outside:

```
curl https://kisee.example.com
```

this should give you the json-home of kisee, like this:

```
{
    "api": {
    "title": "Identification Provider",
    "links": {
        "author": "mailto:julien@palard.fr",
        "describedBy": "https://kisee.readthedocs.io"
    }
},
[...]
```

CHAPTER 2

Features

2.1 Overview

Kisee is an API giving JWTs in exchange for valid usernames/password pairs. That's it.

Kisee is better used as a backend of the Pasee identity manager: Pasee handle groups and can handle multiple identity backends (one or many Kisee instances, twitter, facebook, . . .).

Kisee can use your existing database (or use a dedicated one) to query the username and passwords if you're willing to implement a simple Python class to query it, so Kisee can query anything: LDAP, a flat file, a PostgreSQL database with a strange schema, whatever.

2.2 The backend interface

The backend class used by Kisee must implement the *kisee.identity_provider.IdentityProvider* ABC, meaning the following methods like:

```
async def identify(self, username: str, password: str) -> Optional[User]:
    """Identifies the given username/password pair, returns a dict if found.
    """
```

By implementing the backend ABC, you can make your `kisee` instance use your own backend: your own database schema, or anything storing your usernames and passwords.

To use your backend, specify it in `settings.toml` like this:

```
[identity_backend]
  class = "impart.path.to.your.backend.Class"
  [identity_backend.options]
    no = "option required"
```

The `options` dictionary will be passed as a `options` parameter of your backend. This is were you store typically the hostname, username, and password of your database if any, or path of your backend file, whatever needed.

CHAPTER 3

Configuration File

3.1 Example

A typical `settings.toml` file looks like this:

```
[server]
host = "0.0.0.0"
hostname = "http://localhost:8140"
port = 8140
debug = true

[identity_backend]
  class = "kisee.providers.demo.DemoBackend"
  [identity_backend.options]
    no = "option required"

[email]
  host = "localhost"
  sender = "sender@example.com"

[jwt]
  iss = "example.com"

  # Generated using:
  #
  #    openssl ecparam -name secp256k1 -genkey -noout -out secp256k1.pem
  #
  # Yes we know P-256 is a bad one, but for compatibility with JS
  # clients for the moment we can't really do better.
  private_key = '''
-----BEGIN EC PRIVATE KEY-----
MHQCAQEEIJJaLOWE+5qg6LNjYKOijMelSLYnexzLmTMvwG/Dy0r4oAcGBSuBBAAK
oUQDQgAEE/WCqajmhfppNUB2uekSxX976fcWA3bbdew8NkUtCoBigl9lWkqfnkF1
8H9fgG0gafPhGtub23+8Ldulvmf1lg==
```

(continues on next page)

(continued from previous page)

```
-----END EC PRIVATE KEY-----'''

  # Generated using:
  # openssl ec -in secp256k1.pem -pubout > secp256k1.pub
  public_key = '''
-----BEGIN PUBLIC KEY-----
MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEE/WCqajmhfppNUB2uekSxX976fcWA3bb
dew8NkUtCoBigl9lWkqfnkF18H9fgG0gafPhGtub23+8Ldulvmf1lg==
-----END PUBLIC KEY-----'''
```

3.2 Sentry

Sentry is optional.

For Sentry to work you'll need the *SENTRY_DSN* environment variable, and *sentry-sdk* installed, that's it.

CHAPTER 4

Kisee API

The Kisee API exposes the following resources:

- A json-home on */*
- */jwt/* to manage tokens (mainly create a new one by POSTing)
- */password_recoveries/* to initiate a password lost procedure and manage it.
- *POST /users/* for self-service registration.

CHAPTER 5

Contributing

5.1 Quickstart

To install dev dependencies, create a venv and run:

```
pip install -r requirements-dev.txt
pip install -e .
cp example-settings.toml settings.toml
```

And run kisee in development mode using:

```
adev runserver kisee/kisee.py
```

5.2 Building the API doc

We have a description of the kisee API using OpenAPI v3, that can be checked using:

```
pip install openapi-spec-validator
openapi-spec-validator kisee.yaml
```

so we can generate some things from here, like a documentation, for example:

```
wget https://repo1.maven.org/maven2/org/openapitools/openapi-generator-cli/5.0.0-
↪beta2/openapi-generator-cli-5.0.0-beta2.jar -O openapi-generator-cli.jar
java -jar openapi-generator-cli.jar generate -g html2 -i kisee.yaml -o apidocs/
```

Or by using the plain old standalone HTML/CSS/JS swagger-ui.

5.3 Internals

The `Kisee` daemon does not store `(username, password)` tuples, but uses a Python class, a `backend` you can choose in `settings.toml` to handle the actual storage..

`Kisee` provides some `demo backends` and `test backends` so you can play with them. You can provide your own backend to hit your own database, your LDAP server, or another IdP as needed.

5.4 Releasing

Our version scheme is calver, specifically `YY.MM.MICRO`, so please update it in `kisee/__init__.py` (single place), git tag, commit, and push.

Then to release:

```
git clean -dfqx
python -m build --sdist --wheel .
twine upload dist/*
```

CHAPTER 6

FAQ

6.1 Can I use Kisee to query an OAuth2 service?

Kisee is an identity provider, like twitter or github, so they're side by side, not one on top of the other, they play the same role. You can however use Pasee on top of Kisee and Twitter for example.

6.2 Does Kisee implement groups?

No, Kisee doesn't care about your groups like Github don't care about your groups, they're both just here to say "yes, it's this user" or "no, it is not".

From the Pasee point of view you'll be able to tell:

- User foo from Kisee is in group staff
- User bar from Github is in group staff too

6.3 Does Kisee implement impersonation?

No, if we do implement this we'll probably do in Pasee to rely on Pasee groups to tell who can impersonate.

6.4 Does Kisee expose self-service registration?

Optionally, only if you implement it or use a backend class implementing it.

6.5 Does Kisee expose a password reset feature?

Yes, by sending an email that you can template in the settings.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search