Pasee

Apr 16, 2021

Contents:

The name “Pasee”, inspired from “Kisee” (the IdM), spoken as the french phrase “Passez !”, `[ps()]`.

Pasee is a simple layer over multiple IdMs, typically a Kisee and other identity providers like Twitter, Github, ...

CHAPTER 1

Quickstart

For a `Pasee` to run easily you should start by running a local Kisee.

Once a local `Kisee` is started follow those steps:

Install pasee in a venv:

```
pip install pasee
```

And start it using:

```
pasee --settings-file example-settings.toml
```

CHAPTER 2

Features

2.1 Groups

Pasee could just gather multiple identity providers, but it add one optional feature: handling groups.

Groups are typically not given by identity providers: from your point of view, twitter says "It's this person" not "He's root".

2.2 Self-service registration and password reset

`Pasee` exposes an API for users to register, but it's not its role to handle identities, so it only forwards blindly those requests to the *main configured identity provider*, typically a `Kisee` instance.

2.3 Tokens with authorization scope

`Pasee` delivers JWT to your users, a pair of JWT:

- An access token: prooving the identity (and groups) of the user.
- A refresh token: a special token meant only to be used for requesting a new access token.

The access token has short TTL while the refresh token has a longer one, so your clients can use the refresh token to get new access tokens as needed. The users should never use the refresh token for something else than asking for a new access token, that's why we're not giving groups in the refresh token.

2.4 Users

A user, from a `Pasee` point of view can be seen as a tuple of `(identity_provider, name)`.

Obviously two distinct users can share the same name on two distinct identity providers, they still are two distinct identities, whence the "tuple".

So, if a user "John" identifies against `Pasee`, and `Pasee` uses the `twitter` backend to proove who he is, he'll be identified as `"twitter-John"`, so there's no name clash between `"kisee-John"`, and `"twitter-John"`.

In a convoluted setup where `Pasee` delegates to a `Kisee` named "externals", which delegates to another `Pasee`, which delegates to another `Kisee` named "internals", you'll receive users identified as "externals-internals-ada". As you see `Pasee` is adding the prefixes, `Kisee` does not.

This way you can use a deep and wide setup of `Kisee`s and `Pasee`s, you'll never hit a name-clash, and you'll always be able to visually spot where a user come from.

2.5 Groups

Groups naming is hierarchical, separated by dots, like "foo.bar.baz".

Staff members can create any group. One creating a group becomes staff of its own group, (member of `my_very_personal_group.staff`).

Staff of a group can:

- Create subgroups, like `my_very_personal_group.friends`.
- Manage staff of the group.

This applies recursively, let's play it from scratch:

A `staff` member (you) creates an `admin` group, you become automatically member of `admin.staff`.

As a staff member of admin, you create `admin.developers`, you become automatically member of `admin.developers.staff`.

You add a coworker John to `admin.developers.staff`.

John add three coworkers, Alan, Ada, and Donald to `admin.developers`.

At this point, only you can create subgroups or manage users / staff of `admin`, and only John can create subgroups of `admin.developers`, add members to it, or add staff to it.

Hint: You should create a root group per service using `Pasee`, and use subgroups to handle authorizations, and root groups for global authorizations, like:

- articles - writers - reviewers
- comments - moderators - ...
- superusers
- ...

CHAPTER 3

API

The API is limited to a few endpoints that tends to self-describe themselves.

You can get a JSON-Home on `/` describing the following resources:

- `users` used to manage users
- `groups` used to manage groups
- `tokens` used by users to create tokens

To test the API you can first create a `staff` account:

```
python -m pasee –append –groups staff your_username
```

CHAPTER 4

Configuration

Pasee uses a toml configuration file like:

```
host = "0.0.0.0"
port = 8150

# 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-----
..."""

# Generated using:
# openssl ec -in secp256k1.pem -pubout > secp256k1.pub
public_key = """-----BEGIN PUBLIC KEY-----
MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEEVgsgM7Aliru0XU+OggGC5jxRoZUI4/C
fsNJ8ZUlTKxjn8VzO4Db2ITFvUdyRCQjGRuq5QRJt7a46ZyfrDb+6w==
-----END PUBLIC KEY-----"""

[jwt]
iss = "pasee.meltylab.fr"

[[identity_providers]]
name = "kisee"
host = "127.0.0.1"
port = 8140
protocol = "kisee"
[identity_providers.settings]
public_keys = ["""-----BEGIN PUBLIC KEY-----MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEE/
↪WCqajmhfppNUB2uekSxX976fcWA3bbdew8NkUtCoBigl9lWkqfnkF18H9fgG0gafPhGtub23+8Ldulvmf1lg==-
↪----END PUBLIC KEY-----"""]
```

(continues on next page)

(continued from previous page)

```
[[identity_providers]]
name = "twitter"
host = "twitter.com"
port = 443
protocol = "oauth2"
[identity_providers.settings]
app_id = "..."
app_secret = "..."
```

CHAPTER 5

Contributing

5.1 Quickstart

To install dev dependencies, create a venv and run:

```
pip install flit
flit install --symlink
```

And run kisee using:

```
pasee  # or python -m pasee
```

5.2 Releasing

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

Then to release we're using flit:

```
flit publish
```

CHAPTER 6

FAQ

6.1 How to configure Pasee to use my LDAP server?

Setup an instance of `Kisee` to use it (not implemented yet), and add this `Kisee` instance in the `identity_providers` of your `Pasee` instance.

In your *Kisee* backend you could even expose groups or any meta-informations stored in your LDAP server as JWT claims. Those claims have to be whitelisted in `Pasee` configuration to be kept in `Pasee`-signed tokens (By default, we only trust identities, from identities backends).

6.2 Why a `Kisee` identity backend settings uses an array of public keys?

To help you rotate a `Kisee` private key by allowing both during the transition.

6.3 Can Pasee expose an OAuth2 or OpenID endpoint?

Yes, feel free to implement it, see current `Twitter` and `Facebook` implementations.

6.4 Can Pasee use multiple instances of `Kisee` to hit different identity sources?

Yes, but a single one can handle registrations from Pasee. If you want to let your user choose on which `Kisee` instance they're registering, use the `Kisee` API directly for registration instead of passing registrations thrue `Pasee`.

6.5 I don't get it, why do I need a private key on Kisee and another on Pasee?

Pasee can use multiple identity providers (OAuth2, OpenID connect, Pasee instances), and will even work without a Kisee backend. As a Kisee have to sign tokens, and a Pasee have to sign tokens too, they both need a private key. You could use the same private key on every Kisee and Pasee instances, it won't break the implementation. You can obviously use different ones too.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search