Flask-Principal Documentation

Release 0.3.5

Matt Wright

May 22, 2017

Contents

1 Introduction **3**

2 Links **5**

3 Protecting access to resources **7**

4 Authentication providers **9**

5 User Information providers **11**

6 Granular Resource Protection **13**

7 API **15**
- 7.1 Starting the extension . . . 15
- 7.2 Main Types . . . 15
- 7.3 Predefined Need Types . . . 15
- 7.4 Signals . . . 15

8 Changelog **17**
- 8.1 Flask-Principal Changelog . . . 17

9 Indices and tables **19**

“I am that I am”

CHAPTER 1

Introduction

Flask-Principal provides a very loose framework to tie in providers of two types of service, often located in different parts of a web application:

1. Authentication providers
2. User information providers

For example, an authentication provider may be oauth, using Flask-OAuth and the user information may be stored in a relational database. Looseness of the framework is provided by using signals as the interface.

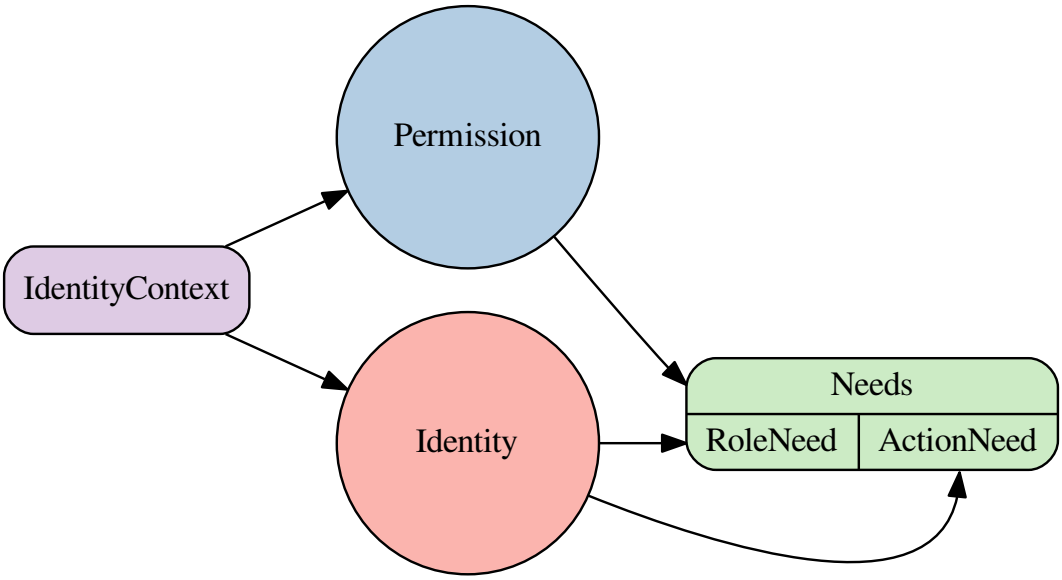
The major components are the Identity, Needs, Permission, and the IdentityContext.

1. The Identity represents the user, and is stored/loaded from various locations (eg session) for each request. The Identity is the user's avatar to the system. It contains the access rights that the user has.
2. A Need is the smallest grain of access control, and represents a specific parameter for the situation. For example "has the admin role", "can edit blog posts".

   Needs are any tuple, or probably could be object you like, but a tuple fits perfectly. The predesigned Need types (for saving your typing) are either pairs of (method, value) where method is used to specify common things such as *"role"*, *"user"*, etc. And the value is the value. An example of such is *('role', 'admin')*. Which would be a Need for a admin role. Or Triples for use-cases such as "The permission to edit a particular instance of an object or row", which might be represented as the triple *('article', 'edit', 46)*, where 46 is the key/ID for that row/object.

   Essentially, how and what Needs are is very much down to the user, and is designed loosely so that any effect can be achieved by using custom instances as Needs.

   Whilst a Need is a permission to access a resource, an Identity should provide a set of Needs that it has access to.
3. A Permission is a set of requirements, any of which should be present for access to a resource.
4. An IdentityContext is the context of a certain identity against a certain Permission. It can be used as a context manager, or a decorator.


Permission
IdentityContext
Identity
Needs
RoleNeed
ActionNeed

CHAPTER 2

Links

- documentation
- source
- *changelog*

CHAPTER 3

Protecting access to resources

For users of Flask-Principal (not authentication providers), access restriction is easy to define as both a decorator and a context manager. A simple quickstart example is presented with commenting:

```
from flask import Flask, Response
from flask.ext.principal import Principal, Permission, RoleNeed

app = Flask(__name__)

# load the extension
principals = Principal(app)

# Create a permission with a single Need, in this case a RoleNeed.
admin_permission = Permission(RoleNeed('admin'))

# protect a view with a principal for that need
@app.route('/admin')
@admin_permission.require()
def do_admin_index():
    return Response('Only if you are an admin')

# this time protect with a context manager
@app.route('/articles')
def do_articles():
    with admin_permission.require():
        return Response('Only if you are admin')
```

CHAPTER 4

Authentication providers

Authentication providers should use the *identity-changed* signal to indicate that a request has been authenticated. For example, the following code is a hypothetical example of how one might combine the popular Flask-Login extension with Flask-Principal:

```
from flask import Flask, current_app, request, session
from flask.ext.login import LoginManager, login_user, logout_user, \
     login_required, current_user
from flask.ext.wtf import Form, TextField, PasswordField, Required, Email
from flask.ext.principal import Principal, Identity, AnonymousIdentity, \
     identity_changed

app = Flask(__name__)

Principal(app)

login_manager = LoginManager(app)

@login_manager.user_loader
def load_user(userid):
    # Return an instance of the User model
    return datastore.find_user(id=userid)

class LoginForm(Form):
    email = TextField()
    password = PasswordField()

@app.route('/login', methods=['GET', 'POST'])
def login():
    # A hypothetical login form that uses Flask-WTF
    form = LoginForm()

    # Validate form input
    if form.validate_on_submit():
        # Retrieve the user from the hypothetical datastore
        user = datastore.find_user(email=form.email.data)
```

```
        # Compare passwords (use password hashing production)
        if form.password.data == user.password:
            # Keep the user info in the session using Flask-Login
            login_user(user)

            # Tell Flask-Principal the identity changed
            identity_changed.send(current_app._get_current_object(),
                                  identity=Identity(user.id))

            return redirect(request.args.get('next') or '/')

    return render_template('login.html', form=form)

@app.route('/logout')
@login_required
def logout():
    # Remove the user information from the session
    logout_user()

    # Remove session keys set by Flask-Principal
    for key in ('identity.name', 'identity.auth_type'):
        session.pop(key, None)

    # Tell Flask-Principal the user is anonymous
    identity_changed.send(current_app._get_current_object(),
                          identity=AnonymousIdentity())

    return redirect(request.args.get('next') or '/')
```

CHAPTER 5

User Information providers

User information providers should connect to the *identity-loaded* signal to add any additional information to the Identity instance such as roles. The following is another hypothetical example using Flask-Login and could be combined with the previous example. It shows how one might use a role based permission scheme:

```
from flask.ext.login import current_user
from flask.ext.principal import identity_loaded, RoleNeed, UserNeed

@identity_loaded.connect_via(app)
def on_identity_loaded(sender, identity):
    # Set the identity user object
    identity.user = current_user

    # Add the UserNeed to the identity
    if hasattr(current_user, 'id'):
        identity.provides.add(UserNeed(current_user.id))

    # Assuming the User model has a list of roles, update the
    # identity with the roles that the user provides
    if hasattr(current_user, 'roles'):
        for role in current_user.roles:
            identity.provides.add(RoleNeed(role.name))
```

CHAPTER 6

Granular Resource Protection

Now lets say, for example, you only want the author of a blog post to be able to edit said article. This can be achieved by creating the necessary *Need* and *Permission* objects, and adding more logic into the *identity_loaded* signal handler. For example:

```
from collections import namedtuple
from functools import partial

from flask.ext.login import current_user
from flask.ext.principal import identity_loaded, Permission, RoleNeed, \
     UserNeed

BlogPostNeed = namedtuple('blog_post', ['method', 'value'])
EditBlogPostNeed = partial(BlogPostNeed, 'edit')

class EditBlogPostPermission(Permission):
    def __init__(self, post_id):
        need = EditBlogPostNeed(unicode(post_id))
        super(EditBlogPostPermission, self).__init__(need)

@identity_loaded.connect_via(app)
def on_identity_loaded(sender, identity):
    # Set the identity user object
    identity.user = current_user

    # Add the UserNeed to the identity
    if hasattr(current_user, 'id'):
        identity.provides.add(UserNeed(current_user.id))

    # Assuming the User model has a list of roles, update the
    # identity with the roles that the user provides
    if hasattr(current_user, 'roles'):
        for role in current_user.roles:
            identity.provides.add(RoleNeed(role.name))

    # Assuming the User model has a list of posts the user
```

```
    # has authored, add the needs to the identity
    if hasattr(current_user, 'posts'):
        for post in current_user.posts:
            identity.provides.add(EditBlogPostNeed(unicode(post.id)))
```

The next step will be to protect the endpoint that allows a user to edit an article. This is done by creating a permission object on the fly using the ID of the resource, in this case the blog post:

```
@app.route('/posts/<post_id>', methods=['PUT', 'PATCH'])
def edit_post(post_id):
    permission = EditBlogPostPermission(post_id)

    if permission.can():
        # Save the edits ...
        return render_template('edit_post.html')

    abort(403)  # HTTP Forbidden
```

CHAPTER 7

API

Starting the extension

Main Types

Predefined Need Types

Signals

identity_changed
: Signal sent when the identity for a request has been changed.

identity_loaded
: Signal sent when the identity has been initialised for a request.

CHAPTER 8

Changelog

Flask-Principal Changelog

Here you can see the full list of changes between each Flask-Principal release.

Version 0.4.0

Not released yet

- Added Python 3 support
- Dropped support for Python 2.5

Version 0.3.5

Released April 3rd 2013

- Fixed possible bug with `AnonymousIdentity` supplying "anon" as the username
- Changed Indentity `name` property to `id` to be more generic

Version 0.3.4

Released February 1st 2013

- Add `__repr__` method to Identity and Permission classes
- Optimized `_is_static_resource` method

Version 0.3.3

Released September 4th 2012

- Add `init_app` method to accomodate usage with a factory pattern.

Version 0.3.2

Released August 25th 2012

- Update to check for `static_url_path` in Flask 0.9

Version 0.3.1

Released August 16th 2012

- Fixed bug with re-raising exceptions/tracebacks

Version 0.3

Released June 20th 2012

- Python 2.5/GAE support
- New extension structure
- Added ignore_static option
- Updated docs

Version 0.2

Initial development by Ali Asfshar. Original repository

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

I

identity_changed (built-in variable), 15
identity_loaded (built-in variable), 15