falcon-auth Documentation

Release 0.0.1

Ritesh Kadmawala

Aug 06, 2018

Table of Contents

Contents:

CHAPTER 1

falcon-auth

A falcon middleware + authentication backends that adds authentication layer to you app/api service.

1.1 Installation

Install the extension with pip, or easy_install.

```
$ pip install -U falcon-auth
```

1.2 Usage

This package exposes a falcon middleware which takes an authentication backend as an input and use it to authenticate requests. You can specify some routes and methods which are exempted from authentication. Once the middleware authenticates the request using the specified authentication backend, it add the authenticated user to the `request context`

```
import falcon
from falcon_auth import FalconAuthMiddleware, BasicAuthBackend

user_loader = lambda username, password: { 'username': username }
auth_backend = BasicAuthBackend(user_loader)
auth_middleware = FalconAuthMiddleware(auth_backend,
                    exempt_routes=['/exempt'], exempt_methods=['HEAD'])
api = falcon.API(middleware=[auth_middleware])

class ApiResource:

    def on_post(self, req, resp):
```

(continues on next page)

(continued from previous page)

```
        user = req.context['user']
        resp.body = "User Found: {}".format(user['username'])
```

1.3 Override Authentication for a specific resource

Its possible to customize the exempt routes, exempt methods and authentication backend on a per resource basis as well

```
import falcon
from falcon_auth import FalconAuthMiddleware, BasicAuthBackend, TokenAuthBackend

# a loader function to fetch user from username, password
user_loader = lambda username, password: { 'username': username }

# basic auth backend
basic_auth = BasicAuthBackend(user_loader)

# Auth Middleware that uses basic_auth for authentication
auth_middleware = FalconAuthMiddleware(basic_auth)
api = falcon.API(middleware=[auth_middleware])


class ApiResource:

    auth = {
        'backend': TokenAuthBackend(user_loader=lambda token: { 'id': 5 }),
        'exempt_methods': ['GET']
    }

    # token auth backend

    def on_post(self, req, resp):
        resp.body = "This resource uses token authentication"

    def on_get(self, req, resp):
        resp.body = "This resource doesn't need authentication"


api.add_route("/api", ApiResource())
```

1.4 Disable Authentication for a specific resource

```
class ApiResource:
    auth = {
        'auth_disabled': True
    }
```

1.5 Accessing Authenticated User

Once the middleware authenticates the request using the specified authentication backend, it add the authenticated user to the *request context*

```
class ApiResource:

    def on_post(self, req, resp):
        user = req.context['user']
        resp.body = "User Found: {}".format(user['username'])
```

1.6 Authentication Backends

- **Basic Authentication**

Implements HTTP Basic Authentication wherein the HTTP `Authorization` header contains the user credentials(username and password) encoded using `base64` and a prefix (typically Basic)

- **Token Authentication**

Implements a Simple Token Based Authentication Scheme where HTTP `Authorization` header contains a prefix (typically Token) followed by an *API Token*

- **JWT Authentication**

Token based authentication using the JSON Web Token standard

- **Dummy Authentication**

Backend which does not perform any authentication checks

- **Multi Backend Authentication**

A Backend which comprises of multiple backends and requires any of them to authenticate the request successfully

1.7 Tests

This library comes with a good set of tests which are included in `tests/`. To run install `pytest` and simply invoke `py.test` or `python setup.py test` to exercise the tests. You can check the test coverage by running `py.test --cov falcon_auth`

1.8 API

class falcon_auth.**FalconAuthMiddleware**(*backend*, *exempt_routes=None*, *exempt_methods=None*)

Creates a falcon auth middleware that uses given authentication backend, and some optinal configuration to authenticate requests. After initializing the authentication backend globally you can override the backend as well as other configuration for a particular resource by setting the *auth* attribute on it to an instance of this class.

The authentication backend must return an authenticated user which is then set as *request.context.user* to be used further down by resources othewise an *falcon.HTTPUnauthorized* exception is raised.

Args:

backend(`falcon_auth.backends.AuthBackend`, required): Specifies the auth backend to be used to authenticate requests

exempt_routes(list, optional): A list of paths to be excluded while performing authentication. Default is `None`

exempt_methods(list, optional): A list of paths to be excluded while performing authentication. Default is `['OPTIONS']`

class falcon_auth.**BasicAuthBackend**(*user_loader*, *auth_header_prefix='Basic'*)

Implements HTTP Basic Authentication Clients should authenticate by passing the *base64* encoded credentials *username:password* in the *Authorization* HTTP header, prepended with the string specified in the setting *auth_header_prefix*. For example:

Authorization: BASIC ZGZkZmY6ZGZkZ2RkZg==

Args:

user_loader(function, required): A callback function that is called with the user credentials (username and password) extracted from the *Authorization* header. Returns an *authenticated user* if user exists matching the credentials or return *None* to indicate if no user found or credentials mismatch.

auth_header_prefix(string, optional): A prefix that is used with the bases64 encoded credentials in the *Authorization* header. Default is `basic`

authenticate(*req*, *resp*, *resource*)

Extract basic auth token from request *authorization* header, deocode the token, verifies the username/password and return either a `user` object if successful else raise an *falcon.HTTPUnauthoried exception*

get_auth_token(*user_payload*)

Extracts username, password from the *user_payload* and encode the credentials *username:password* in *base64* form

class falcon_auth.**TokenAuthBackend**(*user_loader*, *auth_header_prefix='Token'*)

Implements Simple Token Based Authentication. Clients should authenticate by passing the token key in the "Authorizati

HTTP header, prepended with the string "Token ". For example:

Authorization: Token 401f7ac837da42b97f613d789819ff93537bee6a

Args:

user_loader(function, required): A callback function that is called with the token extracted from the *Authorization* header. Returns an *authenticated user* if user exists matching the credentials or return *None* to indicate if no user found or credentials mismatch.

auth_header_prefix(string, optional): A prefix that is used with the token in the *Authorization* header. Default is `basic`

authenticate(*req*, *resp*, *resource*)

Extract basic auth token from request *authorization* header, deocode the token, verifies the username/password and return either a `user` object if successful else raise an *falcon.HTTPUnauthoried exception*

get_auth_token(*user_payload*)

Extracts token from the *user_payload*

class falcon_auth.**JWTAuthBackend**(*user_loader*, *secret_key*, *algorithm='HS256'*, *auth_header_prefix='jwt'*, *leeway=0*, *expiration_delta=86400*, *audience=None*, *issuer=None*, *verify_claims=None*, *required_claims=None*)

Token based authentication using the JSON Web Token standard Clients should authenticate by passing the token key in the *Authorization* HTTP header, prepended with the string specified in the setting *auth_header_prefix*. For example:

Authorization: JWT eyJhbGciOiAiSFMyNTYiLCAidHlwIj

Args:

user_loader(function, required): A callback function that is called with the decoded *jwt payload* extracted from the *Authorization* header. Returns an *authenticated user* if user exists matching the credentials or return *None* to indicate if no user found or credentials mismatch.

secrey_key(string, required): A secure key that was used to encode and create the *jwt token* from a dictionary payload

algorithm(string, optional): Specifies the algorithm that was used to for cryptographic signing. Default is `HS256` which stands for HMAC using SHA-256 hash algorithm. Other supported algorithms can be found here

auth_header_prefix(string, optional): A prefix that is used with the bases64 encoded credentials in the *Authorization* header. Default is `jwt`

leeway(int, optional): Specifies the timedelta in seconds that is allowed as leeway while validating *expiration time / nbf(not before) claim /iat (issued at) claim* which is in past but not very far. For example, if you have a JWT payload with an expiration time set to 30 seconds after creation but you know that sometimes you will process it after 30 seconds, you can set a leeway of 10 seconds in order to have some margin. Default is `0 seconds`

expiration_delta(int, optional): Specifies the timedelta in seconds that will be added to current time to set the expiration for the token. Default is `1 day(24 * 60 * 60 seconds)`

audience(string, optional): Specifies the string that will be specified as value of `aud` field in the jwt payload. It will also be checked agains the `aud` field while decoding.

issuer(string, optional): Specifies the string that will be specified as value of `iss` field in the jwt payload. It will also be checked agains the `iss` field while decoding.

authenticate(*req*, *resp*, *resource*)

Extract auth token from request *authorization* header, deocode jwt token, verify configured claims and return either a `user` object if successful else raise an *falcon.HTTPUnauthoried exception*

get_auth_token(*user_payload*)

Create a JWT authentication token from `user_payload`

Args:

user_payload(dict, required): A *dict* containing required information to create authentication token

class falcon_auth.**NoneAuthBackend**(*user_loader*)

Dummy authentication backend.

This backend does not perform any authentication check. It can be used with the MultiAuthBackend in order to provide a fallback for an unauthenticated user.

Args:

user_loader(function, required): A callback function that is called without any arguments and returns an *unauthenticated user*.

authenticate(*req*, *resp*, *resource*)
Authenticate the request and return the authenticated user. Must return *None* if authentication fails, or raise an exception

class falcon_auth.**MultiAuthBackend**(**backends*)
A backend which takes two or more `AuthBackend` as inputs and successfully authenticates if either of them succeeds else raises *falcon.HTTPUnauthoried exception*

Args:

backends(AuthBackend, required): A list of *AuthBackend* to be used in order to authenticate the user.

authenticate(*req*, *resp*, *resource*)
Authenticate the request and return the authenticated user. Must return *None* if authentication fails, or raise an exception

get_auth_token(*user_payload*)
Returns a authentication token created using the provided user details

Args:

user_payload(dict, required): A *dict* containing required information to create authentication token

CHAPTER 2

Contributing

2.1 Getting Started

Fork the repository to your own account.

Clone the repository to a suitable location on your local machine.

```
$git clone https://github.com/loanzen/falcon-auth.git
```

To update the project from within the project's folder you can run the following command:

```
$git pull --rebase
```

2.2 Building

Install the project's dependencies.

```
$pip install -r requirements.txt
$pip install -r requirements-dev.txt
```

2.3 Feature Requests

I'm always looking for suggestions to improve this project. If you have a suggestion for improving an existing feature, or would like to suggest a completely new feature, please file an issue with my Github repository

2.4 Bug Reports

You may file bug reports on Github Issues

2.5 Pull Requests

Along with my desire to hear your feedback and suggestions, I'm also interested in accepting direct assistance in the form of new code or documentation. Please feel free to file pull requests against my Github repository

CHAPTER 3

LICENSE

```
The MIT License (MIT)

Copyright (c) 2017 Ritesh Kadmawala <ritesh@loanzen.in>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Index