moz_inapp_pay Documentation

Release 1.0

Kumar McMillan and contributors

October 20, 2015

Contents

A Python module to make web payments with Mozilla's navigator.mozPay().

You can read all about how web payments work in the developer docs.

Mozilla's web payments allow you to operate an app (or website) that accepts payments for digital goods. As payments are completed, the Firefox Marketplace needs to communicate the transaction ID to your app. You can use this library to validate the signature of that communication. All communication is done via signed JWT (JSON Web Token).

This also includes some generic ways to validate JWT objects. Hmm, maybe that should be extracted for more general use.

CHAPTER 1

Installation

With pip or easy_install, run:

```
pip install mozpay
```

Or install it from source:

```
pip install git+git://github.com/mozilla/mozpay-py.git
```

CHAPTER 2

Verify a postback

```
import logging
from mozpay import InvalidJWT, process_postback
try:
    data = process_postback(signed_request,
                            app_key,
                            app_secret)
    print data['response']['transactionID']
except InvalidJWT:
    loggging.exception('in postback')
```

CHAPTER 3

Verify a chargeback

```
import logging
from mozpay import InvalidJWT, process_chargeback
try:
    data = process_chargeback(signed_request,
                              app_key,
                              app_secret)
    print data['response']['transactionID']
    print data['response']['reason']
except InvalidJWT:
    logging.exception('in chargeback')
```

CHAPTER 4

Use It With Django

If you use the Django framework, there's an app you can plug right into your urls.py.

Add the app in your settings.py file:

```
INSTALLED_APPS = [
    # ...
    'mozpay.djangoapp',
]
```

Add your key and secret that was granted by the Firefox Marketplace to your **local** settings.py file:

```
MOZ_APP_KEY = '<from marketplace.mozilla.org>'
MOZ_APP_SECRET = '<from marketplace.mozilla.org>'
```

Note: Do not commit your secret to a public repo. **Always keep it secure on your server**. Never expose it to the client in JavaScript or anywhere else.

Add the postback / chargeback URLs to your urls.py file:

```
from django.conf.urls.defaults import patterns, include

urlpatterns = patterns('',
    ('^moz/', include('mozpay.djangoapp.urls')),
)
```

This will add `/moz/postback` and `/moz/chargeback` to your URLs. You'll enter these callback URLs into the in-app payment config screen on the Firefox Marketplace.

If you want to do further processing on the postbacks, you can connect to a few signals. Here is an example of code to go in your app (probably in models.py):

```
import logging
from django.dispatch import receiver

from mozpay.djangoapp.signals import (moz_inapp_postback,
                                      moz_inapp_chargeback)


@receiver(moz_inapp_postback)
def mozmarket_postback(request, jwt_data, **kwargs):
    logging.info('transaction ID %s processed ok'
                 % jwt_data['response']['transactionID'])
```

```
@receiver(moz_inapp_chargeback)
def mozmarket_chargeback(request, jwt_data, **kwargs):
    logging.info('transaction ID %s charged back; reason: %r'
                 % (jwt_data['response']['transactionID'],
                    jwt_data['response']['reason']))
```

Exceptions are logged to the channel `mozpay.djangoapp.views` so be sure to add the appropriate handlers to that.

When an InvalidJWT exception occurs, a 400 Bad Request is returned.

CHAPTER 5

JWT Verification API

Helper functions to verify JWT (JSON Web Token) objects. Some are specific to Firefox Marketplace payments, others are more generic.

mozpay.verify.**verify_jwt**(*signed_request*, *expected_aud*, *secret*, *validators=[]*, *required_keys=('request.pricePoint', 'request.name', 'request.description', 'response.transactionID')*, *algorithms=None*)

Verifies a postback/chargeback JWT.

Returns the trusted JSON data from the original request. When there's an error, an exception derived from *mozpay.exc.InvalidJWT* will be raised.

This is an all-in-one function that does all verification you'd need. There are some low-level functions you can use to just verify certain parts of a JWT.

Arguments:

signed_request JWT byte string.

expected_aud The expected value for the aud (audience) of the JWT. See mozpay.verify.verify_audience().

secret A shared secret to validate the JWT with. See *mozpay.verify.verify_sig()*.

validators A list of extra callables. Each one is passed a JSON Python dict representing the JWT after it has passed all other checks.

required_keys A list of JWT keys to validate. See *mozpay.verify.verify_keys()*.

algorithms A list of valid JWT algorithms to accept. By default this will only include HS256 because that's what the Firefox Marketplace uses.

mozpay.verify.**verify_sig**(*signed_request*, *secret*, *issuer=None*, *algorithms=None*, *expected_aud=None*)

Verify the JWT signature.

Given a raw JWT, this verifies it was signed with *secret*, decodes it, and returns the JSON dict.

mozpay.verify.**verify_claims**(*app_req*, *issuer=None*)

Verify JWT claims.

All times must be UTC unix timestamps.

These claims will be verified:

- iat: issued at time. If JWT was issued more than an hour ago it is rejected.
- exp: expiration time.

All exceptions are derived from *mozpay.exc.InvalidJWT*. For expirations a *mozpay.exc.RequestExpired* exception will be raised.

mozpay.verify.**verify_keys**(*app_req*, *required_keys*, *issuer=None*)

Verify all JWT object keys listed in required_keys.

Each required key is specified as a dot-separated path. The key values are returned as a list ordered by how you specified them.

Take this JWT for example:

```
{
    "iss": "...",
    "aud": "...",
    "request": {
        "pricePoint": 1,
    }
}
```

You could verify the presence of all keys and retrieve their values like this:

```
iss, aud, price = verify_keys(jwt_dict,
                              ('iss',
                               'aud',
                               'request.pricePoint'))
```

Do you see how the comma separated assigned variables match the keys that were extracted? The order is important.

CHAPTER 6

Exceptions

Exceptions that might be raised during JWT processing.

exception mozpay.exc.**InvalidJWT**(*msg*, *issuer=None*)
The JWT received by an issuer is invalid.

exception mozpay.exc.**RequestExpired**(*msg*, *issuer=None*)
The JWT request expired.

CHAPTER 7

Source Code and Bug Tracker

The source code is hosted on https://github.com/mozilla/mozpay-py and you can submit pull requests and bugs over there.

CHAPTER 8

Developers

Hello! To work on this module, check out the source from git and be sure you have the tox tool. To run the test suite, cd into the root and type:

```
tox
```

This will run all tests in a virtualenv using the supported versions of Python.

To build the documentation, create a virtualenv then run:

```
pip install -r docs/requirements.txt
```

Build the docs from the root like this:

```
make -C docs/ html
```

Et voila:

```
open docs/_build/html/index.html
```

CHAPTER 9

Changelog

- 2.1.0
  - Added `algorithms` list to verification functions to adjust what JWT algorithms are accepted. **By default only HS256 is accepted now**.
  - Upgraded PyJWT to the latest version.
  - Removed M2Crypto as a dependency because that is no longer needed and it wasn't actually used for our signing purposes anyway.
- 2.0.0
  - Changed postback/chargeback from reading a JWT in the post body to reading it from the `notice` parameter. See https://bugzilla.mozilla.org/show_bug.cgi?id=838066 for details.
- 1.0.4
  - First working release.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

m

I

M

R

V