Flask-Kerberos Documentation

Release 1.0.4

Michael Komitee

May 05, 2015

Contents

Flask-Kerberos is an extension to Flask that allows you to trivially add Kerberos based authentication to your website. It depends on both Flask and python-kerberos 1.1.1+. You can install the requirements from PyPI with *easy_install* or *pip* or download them by hand.

Unfortunately, as is the case with most things kerberos, it requires a kerberos environment as well as a keytab. Setting that up is outside the scope of this document.

The official copy of this documentation is available at Read the Docs.

CHAPTER 1

Installation

Install the extension with one of the following commands:

```
$ easy_install Flask-Kerberos
```

or alternatively if you have *pip* installed:

```
$ pip install Flask-Kerberos
```

CHAPTER 2

How to Use

To integrate Flask-Kerberos into your application you'll need to generate your keytab set the environment variable *KRB5_KTNAME* in your shell to the location of the keytab file.

After that, it should be as easy as decorating any view functions you wish to require authentication, and changing them to accept the authenticated user principal as their first argument:

```
from flask_kerberos import requires_authentication

@app.route("/protected")
@requires_authentication
def protected_view(user):
    ...
```

Flask-Kerberos assumes that the service will be running using the hostname of the host on which the application is run. If this is not the case, you can override it by initializing the module:

```
from flask_kerberos import init_kerberos

init_kerberos(app, hostname='example.com')
```

CHAPTER 3

How it works

When a protected view is accessed by a client, it will check to see if the request includes authentication credentials in an *Authorization* header. If there are no such credentials, the application will respond immediately with a *401 Unauthorized* response which includes a *WWW-Authenticate* header field with a value of *Negotiate* indicating to the client that they are currently unauthorized, but that they can authenticate using Negotiate authentication.

If credentials are presented in the *Authorization* header, the credentials will be validated, the principal of the authenticating user will be extracted, and the protected view will be called with the extracted principal passed in as the first argument.

Once the protected view returns, a *WWW-Authenticate* header will be added to the response which can then be used by the client to authenticate the server. This is known as mutual authentication.

CHAPTER 4

Full Example

To see a simple example, you can download the code from github. It is in the example directory.

CHAPTER 5

Changes

5.1 1.0.4

- fixes an exception triggered when a client doesn't request mutual authentication.

5.2 1.0.3

- bug fixes

5.3 1.0.2

- initial implementation

CHAPTER 6

API References

The full API reference: