apiauth Documentation

Release 0.1

Author

December 19, 2016

Contents

A library to create valid API requests by a client and to verify API requests on the server. This library just handles the nonce and MAC generation, so

- The client is responsible for acquiring the shared secret (the *key*) and the key ID.
- The server will need to prevent replay attacks.

CHAPTER 1

Rationale

I was looking for a library to handle the signing and validating of API requests and found macauthlib. It's good; you should probably use it. It supports a 'variety of common object types' and mutates those objects by adding the header for you. I'm not using it because

- I've got a bug up my butt about mutable objects even though the macauthlib functions are idempotent.
- I haven't contributed much code to the world and macauthlib links to a spec that was simple enough that I thought I could write code against it! Later I noticed that it was the last version of the OAuth 2.0 spec that Eran Hammer authored. He withdrew from that development and his reasons are really interesting.

CHAPTER 2

Client Usage

A client will have received a key ID and a key by some method outside the scope of this library. So, the client only needs to do one thing: sign the requests it makes.

The output of *apiauth.client.sign_request()* should be placed in the **Authorization** header.

2.1 API

apiauth.client.**sign_request**(*key_id*, *key*, *method*, *host*, *port*, *request_uri*)

Generates an 'Authentication' header with some sane defaults.

```
>>> sign_request('keyid', 'mykey', 'POST', 'api.example.com', 443, '/foo/bar?baz=buzz')
'MAC id="keyid", ts="1374996296", nonce="e4493430d227af804ef7fbac9c40d4564a133c03", mac="ntu+vZt
```

Args: method: A valid HTTP method, like 'GET' or 'POST'. host: The server name. request_uri: The URI of the request, including parameters. key_id: The key ID you negotiated with the server. key: The secret key you negotiated with the server. port: The remote port; defaults to 443.

Returns: A value for the Authorization header. This will generate the timestamp, nonce, and mac for you.

CHAPTER 3

Server Usage

The server will need to prevent replay attacks (which is outside the scope of this library).

It will also be responsible for dispensing key IDs and secrect keys. We have provided *apiauth.server.key_pair()* to help with that.

The server should expect an **Authorization** header. If it's not provided, the response should be a 401 and contain the WWW-Authenticate header.

The *apiauth.server.unpack_header()* function can be used to extract the values in the header and the either of the following functions used to validate it: *apiauth.server.valid_request()* or *apiauth.server.valid_request_header()*.

3.1 API

apiauth.server.**key_pair**()

Returns a tuple of a key ID and a secret key.

Both use os.urandom. The key is a md5sum, all upper case. The key is simply base64 encoded.

Flask has a very simple suggestion for creating secret keys: http://flask.pocoo.org/docs/quickstart/#sessions

```
>>> [len(x) for x in key_pair()]
[32, 40]
```

apiauth.server.**unpack_header**(*header*)

Returns a dictionary with the values in the header.

```
>>> sorted(unpack_header('MAC id="keyid", ts="1234567890", nonce="asdfghjkl", mac="qwertyuiop",
['ext', 'id', 'mac', 'nonce', 'ts']
>>> sorted(unpack_header('MAC id="keyid", ts="1234567890", nonce="asdfghjkl", mac="qwertyuiop"')
['id', 'mac', 'nonce', 'ts']
```

Returns: A dictionary with the following keys: id, ts, nonce, mac, and optionally ext.

apiauth.server.**valid_request**(*given_timestamp*, *given_nonce*, *given_mac*, *key*, *method*, *host*, *port*, *request_uri*)

Whether or not the given timestamp, nonce, and mac can be recalculated.

```
>>> valid_request(1234567890, 'nonce', 'aDBHxns5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey',
True
>>> valid_request(1987654321, 'nonce', 'aDBHxns5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey',
False
```

```
>>> valid_request(1234567890, 'badnonce', 'aDBHxns5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey
False
>>> valid_request(1234567890, 'nonce', 'BADMACs5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey',
False
>>> valid_request(1234567890, 'nonce', 'aDBHxns5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey',
False
>>> valid_request(1234567890, 'nonce', 'aDBHxns5jtbW2kQPD3wlyvIdyOJPlkAaY2l4oBA9Vk8=', 'mykey',
False
```

apiauth.server.**valid_request_header**(*header*, *key*, *method*, *host*, *port*, *request_uri*)

Whether or not the given header can be recalculated.

```
>>> valid_request_header('MAC id="keyid", ts="1374996296", nonce="e4493430d227af804ef7fbac9c40d4
True
```

a

A

K

S

U

V