django-request-signer Documentation

Release 0.0.2

imtapps

January 27, 2012

CONTENTS

Contents:

CHAPTER

ONE

ABOUT DJANGO REQUEST SIGNER

1.1 About

Django Request Signer provides both a client and a server component to assist in verifying that both the sending and receiving ends of a web service call can trust one another. This trust is established by allowing clients to register with the server and recieve a unique public client id and a private key.

1.2 How it Works

1. A client will have an id and a private key which is issued by the server.
2. The server will store all client ids and corresponding private keys.
3. **When a client needs to request something from the server the following will happen:**
    (a) **The request URL, querystring, post data, and client id will be combined with** the private key to create a unique signature.
    (b) **The url, post data (if any exists), querystring, plus the client id and** signature will be passed to the server in an http request.
    (c) **The server will receive the request, and use the client id to look up the** corresponding private key.
    (d) **The server will then use the request (minus the signature) along with the** private key to try to recreate the exact same signature as the one passed from the client.
    (e) **If the server is able to calculate the same signature that was provided by the** client, the server knows it can trust the request, if not the server will respond with a Bad Request (400).

CHAPTER

TWO

REQUEST SIGNER SERVER

2.1 Creating a Client Id and Private Key

1. Add 'request_signer' to INSTALLED_APPS in your Django settings file.
2. Log into the Django Admin.
3. Navigate to "/request_signer/authorizedclient/add/".
4. Choose a Client Id and save the generated Private Key.
5. That's it! Now you have everything you need to be able to talk to your server.

2.2 Requiring a valid signature for views

To require a valid signature for a view use the **signature_required** decorator

Function based views

```
from request_signer import signature_required

@signature_required
def myview(request):
    pass
```

Class based views

```
from request_signer import signature_required

url(r'sample/',  signature_required(views.MyView.as_view())),
```

CHAPTER

THREE

REQUEST SIGNER CLIENT

3.1 Creating a signed request

You can create a signed URL using the SignedRequestFactory's build_request_url method:

```
from request_signer.client.generic import SignedRequestFactory

factory = SignedRequestFactory('http_method', 'client_id', 'private_key')
signed_request_url = factory.build_request_url(post_data_dict_or_none, 'request_url')
```

Alternatively, you can create a client class to encapsulate dealing with external services:

```
from request_signer.client.generic import Client

class OurClient(Client):
    domain_settings_name = 'DJANGO_SETTINGS_SERVICE_DOMAIN'
    client_id_settings_name = 'DJANGO_SETTINGS_CLIENT_ID'
    private_key_settings_name = 'DJANGO_SETTINGS_PRIVATE_KEY'

    def do_some_remote_action(self, whatever, args, you, want):
        response = self._get_response('POST', '/service/endpoint/',
            dict(arg1=whatever, arg2=you, arg3=want))
        return response.is_successful

    def get_some_remote_data(self, key):
        response = self._get_response('GET', '/service/endpoint/{key}'.format(key=key))
        return response.json
```

To use this client class:

```
client = OurClient()
if client.do_some_remote_action("this", "thing", "rocks", "hard"):
    server_json = client.get_some_remote_data(123)
    print server_json['secret']
else:
    print "fail!"
```

CHAPTER

FOUR

ACCEPTANCE TESTS

These are some acceptance tests written in lettuce to demonstrate how the client/server interaction works.

```
Feature: Server rejects all requests that do not have a valid signature

    Scenario: Server accepts valid signature from client
        Given a client with the client id "me" and the private key "bWU="
            And that client is registered with the server
        When the client makes a request to the server at "/sample/" with the correct signature
        Then the server should reply with a "200"

    Scenario: Server rejects incorrect signature from client
        Given a client with the client id "me" and the private key "bWU="
            And that client is registered with the server
        When the client makes a request to the server at "/sample/" with an invalid signature
        Then the server should reply with a "400"

    Scenario: Server rejects unsigned request
        Given a client with the client id "me" and the private key "bWU="
            And that client is registered with the server
        When the client makes a request to the server at "/sample/" with no signature
        Then the server should reply with a "400"

    Scenario: Server accepts valid signature from client with post data
        Given a client with the client id "me" and the private key "bWU="
            And that client is registered with the server
        When the client posts a request to the server at "/sample/" with the correct signature and th
            | arg1 | arg2 | arg3 | arg4 |
            | val1 | val2 | val3 | val4 |
        Then the server should reply with a "200"

    Scenario: Server accepts valid signature from client that posts with no data
        Given a client with the client id "me" and the private key "bWU="
            And that client is registered with the server
        When the client posts a request to the server at "/sample/" with the correct signature and no
        Then the server should reply with a "200"
```

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*