django-firefence Documentation

Release 0.1.0

Rehan Dalal

Jul 16, 2017

Contents

CHAPTER 1

Overview

django-firefence is a project developed to provide firewall style request filtering to a Django project at the application level or at the view level.

The library is compatible with Django >= 1.8.

Installation

Installing django-firefence is easily done using pip. Assuming it is installed just run the following from the command line:

```
$ pip install django-firefence
```

This command will download the latest version of django-firefence from the Python Package Index and install it to your system. More information about `pip` and pypi can be found here:

- install pip
- pypi

Alternatively you can install from the distribution using the *setup.py* script:

```
$ python setup.py install
```

You could also install the development version by running the following:

```
$ pip install django-firefence==dev
```

Or simply install from a clone of the git repo (recommended for contributors to the project):

```
$ git clone https://github.com/rehandalal/django-firefence.git
$ mkvirtualenv django-firefence
$ pip install -r requirements.txt
$ pip install --editable .
```

Concepts

Rules

A Rule is the basic building block of django-firefence. They are objects that define what characterists of a request to match on and what action to take if they match.

You must define an action for all rules. This action must be one of either 'ALLOW' or 'DENY'.

You may also define a host for a rule. This host will match against the hostname or the IP address of the incoming request. This can be a simple hostname of the remote machine(eg: 'localhost'), an IPv4 address (eg: '192.168.1.1'), an IPv6 address (eg: '2001:0db8:85a3:0000:0000:8a2e:0370:7334'), an IPv4 subnet in CIDR notation (eg: '192.168.1.0/24') or and IPv6 subnet in CIDR notation (eg: '2001:0db8::/32').

Finally, you may define a port for a rule. This will match the server port that the request is made to. Typically requests are made to port 80 for HTTP and port 443 for HTTPS but if you have some kind of non-standard setup you can use this to filter accordingly. The port can be an integer or a string (eg: 80 or '80'), a string representing a range of ports (eg: '80:90'), a string with a comma-separated list of ports (eg: '80, 443') or a list or tuple of integers or strings (eg: ('80', '443') or [80, 443]).

If the host is not defined the rule will match all IPs or hostnames. Similarly, if the port is not defined the rule will match all ports. If both are defined, both must match.

RuleSets

``RuleSet``s are an ordered, iterable collection of ``Rule``s. They provide the list of rules for a request to be matched against. Rules are applied to requests in order. When a request matches a rule, that rule's action is applied and all subsequent rules are ignored.

If there are no rules in a RuleSet there is no action taken. If for some reason you wanted to block all requests you would need to add a rule with the action set to 'DENY' and no host or port specified.

If a RuleSet only has rules with 'DENY' as the action, it will allow all requests except the ones that match one of the rules. However, if there are any rules in a RuleSet that have 'ALLOW' as an action, then requests are denied by default unless they match an allow-rule.

Fences

A Fence is a backend object that takes a RuleSet and defines what to do if a denial-rule is matched. The default backend provided by django-firefence simply raises a PermissionDenied error when a denial occurs.

Settings

All the settings are optional and can be set in your Django settings file as follows:

```
FIREFENCE = {
    'RULES': [
        {
            'action': 'ALLOW',
            'host': '192.168.1.1',
            'port': '80, 443',
        }
    ],
    'DEFAULT_BACKEND': 'firefence.backends.Fence',
}
```

These are the available settings:

RULES A list or tuple of default rules. These will be used by the middleware or the decorator (if not specified).

Each rule may be a `dict` or a `Rule` object.

DEFAULT: `()`

DEFAULT_BACKEND An import path for the backend class to use. This backend will be used by the middleware and the decorator (if not specified).

DEFAULT: `'firefence.backends.Fence'`

CHAPTER 2

Recipes

There are many different ways in which you may choose to use django-firefence. Here are some of the basic patterns that you can use:

Middleware

The easiest, but least flexible way to use django-firence is to simply install the `FirefenceMiddleware` middleware and define some default rules:

```
MIDDLEWARE += ['firefence.middleware.FirefenceMiddleware']

FIREFENCE= {
    'RULES': [
        {
            'action': 'ALLOW',
            'host': '192.168.1.1',
            'port': '80, 443',
        }
    ],
}
```

When using the middleware, *ALL* requests are filtered through the default rules.

By default the middleware uses the provided `Fence` backend, however you may change the `DEFAULT_BACKEND` setting to use a custom backend.

Decorator

django-firefence comes with a view decorator that you can use to protect individual views.

This decorator allows you to specify a set of rules to use as well what backend class to use. If either is not provided the defaults specified in the settings will be used.

Here are some examples of how to use the decorator

```
from firefence.decorators import fence_protected
from firefence.rules import Rule

from my_project.firefence_backends import CustomFence


# Use the default rules and backend
@fence_protected()
def my_view(request):
    return render(request, 'template.html')


# Use a custom set of rules
@fence_protected(rules=[
    Rule(action=Rule.ALLOW, host='192.168.1.1')
])
def another_view(request):
    return render(request, 'template.html')


# Use a custom backend
@fence_protected(backend_class=CustomFence)
def third_view(request):
    return render(request, 'template.html')
```

Fence

Sometimes you may have a common set of rules you wish to apply to a number of views. One way that you could do this is to create an instance of the `Fence` backend with those rules and use it to decorate the views:

```
from firefence.backends import Fence
from firefence.rules import Rule


fence = Fence([
    Rule(action=Rule.DENY, host='192.168.1.1', port=80),
    Rule(action=Rule.ALLOW, port=[80, 443]),
])


@fence.protect
def my_view(request):
    return render(request, 'template.html')


@fence.protect
def another_view(request):
    return render(request, 'template.html')
```

Custom Backend

The provided `Fence` backend raises a `PermissionDenied` error when a denial occurs. If this is not the desired behaviour, you must use a custom backend.

To make the process easy we provide a `AbstractFence` class that you can extend to easily create new backends. All you have to do is implement a `reject` method on the new backend. This method must either raise an exception that Django can handle or return an `HttpResponse` object.

```
from firefence.backends import AbstractFence


class CustomFence(AbstractFence):
    def reject(self, request):
        return render(request, 'denied.html')
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search