Django IPRestrict Documentation

Release 1.7.0

Tamas Szabo

Oct 05, 2018

Contents

Django IPRestrict is an app that can be used to restrict access to your Django project by IP addresses.

The restriction can be for the whole site or sections of it.

IP addresses can be specific IP addresses, ranges of IP addresses or IP addresses from given countries.

The code is released under BSD license and it is hosted on github at https://github.com/muccg/django-iprestrict/.

CHAPTER 1

Table of Contents

1.1 Requirements and Installation

1.1.1 Requirements

- `Django 1.8+`

Additionally, if you would like to use country based restrictions you will need:

- `pycountry`
- MaxMind `geoip` or `geoip2` libraries as described in the *Django* documentation. Links below.

In case you are on at least *Django 1.9* or newer, you should configure geoip2, if you are on *Django 1.8* you have to use and configure geoip.

1.1.2 Installation

You can pip install from PyPI:

```
pip install django-iprestrict
```

The country based lookups are optional, if you need it you can install them with:

```
pip install django-iprestrict[geoip]
```

Note: if you're not using the country based lookups you will have to set the `IPRESTRICT_GEOIP_ENABLED` setting to `False` in your `settings.py`. See: *IPRESTRICT_GEOIP_ENABLED*.

Development

For development create a `virtualenv`, activate it and then:

```
pip install -e .[geoip,dev]
```

To run the tests against the *python* and *Django* in your virtualenv:

```
./runtests.sh
```

To run the tests against all combinations of *python 2*, *python 3*, and supported *Django* versions:

```
tox
```

This will also run *flake8*.

1.2 Configuration

1.2.1 Setting up the app

Add `iprestrict` to `INSTALLED_APPS` in your settings file:

```
INSTALLED_APPS = (
  ...
  'iprestrict',
)
```

Run the migrations for the `iprestrict` application:

```
$ ./manage.py migrate iprestrict
```

Enable Django Admin for at least the iprestrict application.

Add the urls of iprestrict to your project. Ex in your root urls.py:

```
from django.conf.urls import url, include

urlpatterns = [
    # ... snip ...
    url(r'^iprestrict/', include('iprestrict.urls', namespace='iprestrict')),
```

This configuration will allow you to configure and test your restriction rules.

1.2.2 Configuring the restriction rules

Go to your admin page and open IPRestrict Rules to get a list of your current rules.

The rules consist of:

- a **URL pattern** - this is *regex* that will be matched against the *URL* requested by the client. The value `"ALL"` is special and it will always match (a tad nicer than `.*`)
- **Reverse ip group** - reverses the *IP Group*. Ex. *“Australian IPs”* will match all IPs from Australia. However if *Reverse ip group* is set it will match all IPs that aren’t in Australia.
- an **IP Group** - a named group of IP addresses. Examples would be *“localnet”*, *“Trusted net”*, *“Sam’s Home IP”* etc. or the provided *“ALL”* and *“localhost”*
- an **Action** (labeled *“Is allowed?”*) - what to do if both the client url and the ip match the *URL Pattern* and the *IP Group*. Possible values are `"ALLOW"` and `"DENY"` to allow or deny the request.

The rules are checked in the order you see them from top to bottom. The url the client requested is matched against the *URL Pattern* and the client IP address is checked if it is in the *IP Group*. If both the URL and IP address of a rule match the processing of the rules stops, and the request will be denied or allowed based on the rule's *Action*.

If no rules match the request the request will just fall through (ie. it will be allowed).

As you can see after installation 2 rules are provided by default.

- The first one allows all request from localhost
- The second denies all requests

This is a deny by default strategy, when you will have to create rules for all the IP addresses that can access the application explicitly.

To allow everything by default and specify the IP addresses you would like to deny access simply delete the *ALL, ALL, DENY* rule.

The order of the rules can be changed by clicking the *"Move Up"* and *"Move Down"* links.

Example config:

```
/admin/.* localnet ALLOW
/admin/.* ALL DENY
```

These rules would restrict access to admin only from localnet, but allow access to the rest of the application.

Example config 2:

```
ALL "Fishy IPs" DENY
ALL "Trusted Nets" ALLOW
ALL ALL DENY
```

App can be used only from *"Trusted Nets"*, but even inside the *Trusted Nets* there are some IPs you would like to Deny access to (defined in *"Fishy IPs"*).

Example config 3:

```
ALL "localnet" ALLOW
ALL "Fishy IPs" DENY
ALL "Australian IPs" "Reverse ip group" DENY
ALL ALL ALLOW
```

Allow full access from localnet, deny some *"Fishy IPs"*, deny access from IPs that are NOT from Australia, then allow access for everyone else. The last 2 rules combined is a way to allow access only to IPs from Australia.

Sooner or later you will probably have to define some *IP Groups* (ex. like Trusted Nets above).

There are 2 types of IP Groups. The first type allows you to define specific IP addresses or ranges/subnets of IP addresses. The second type allows you to define rules based on the IP's country of origin.

Both IP Groups have a name and an optional longer description.

Range based IP Groups

The range based IP Groups are defined by a a list of IP Ranges.

IP Ranges can be:

- a single IP address (complete just the *First ip* field)
- a subnet (complete the *First ip* field and the *CIDR prefix length*)

- a range of ip addressess (complete the *First ip* and the *Last ip* in the range and leave the *Cidr prefix length* empty)

Ex.

Value	First ip	Cidr prefix length	Last ip
single ip 192.168.1.1	192.168.1.1		
subnet 192.168.1.1/24	192.168.1.1	24	
ip range 192.168.1.1 - 192.168.1.10	192.168.1.1		192.168.1.10

Location based IP Groups

The usage of location based IP Group is optional. It is possible to opt out of using them by setting `IPRESTRICT_GEOIP_ENABLED` to `False` in your `settings.py` in which case you don't have to install the dependencies needed by `geoip`.

The location based IP Groups are defined by a list of 2 digit country codes. You can add one or more country codes to the group. If you have more than one country code, the group will match the IP address if its country matches any of the country codes in the group.

Ex.

Value	Country codes
All IPs from Australia	AU
All IPs from Australia and New Zealand	AU, NZ

1.2.3 Testing the rules

When you are happy with the rules you set up, you might want to test them.

Go to *YOUR_URL/iprestrict/* page. You can use the page to enter any *URL* and *IP Address* and *Test* them against the rules in your database.

1.2.4 Enabling the middleware

Add `iprestrict.middleware.IPRestrictMiddleware` to your `MIDDLEWARE` in your settings file (or `MIDDLEWARE_CLASSES` for old versions of Django). Generally, you will want this middleware to run early, before your session, auth etc. middlewares (the `superuser_required` decorator may also not function correctly if placed out of order):

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'iprestrict.middleware.IPRestrictMiddleware',
    ...
)
```

Your Django project is now restricted based on the rules defined.

1.2.5 Settings

Django IP Restrict has settings to adapt to the environment in which your app is hosted in.

IPRESTRICT_GEOIP_ENABLED

Default: `True`

Set to `False` if you don't require restriction by location. In this case the dependencies needed by location based IP Groups, don't have to be installed.

IPRESTRICT_RELOAD_RULES

Default: `True`

When set to `False` rules will be reloaded only after restarting the server. See *Rules are cached*.

IPRESTRICT_IGNORE_PROXY_HEADER

Default: `False`

When this setting is `True`, Django IP Restrict will completely disregard the `X-Forwarded-For` HTTP header. Normally, the middleware would block requests with a suspect value for `X-Forwarded-For`.

IPRESTRICT_TRUSTED_PROXIES

Default: `[]` (Empty List)

Use this setting when your app is hosted behind a reverse proxy. When values are provided, they will be checked against the HTTP `X-Forwarded-For` header to determine the true client IP address.

IPRESTRICT_TRUST_ALL_PROXIES

Default: `False`

Use this setting when using a managed proxy with a dynamic IP (like when behind an AWS Load Balancer, or other cloud equivalent). When this setting is `True`, Django IP Restrict will always check the HTTP `X-Forwarded-For` header to determine the true client IP address.

1.3 Changing and reloading rules

1.3.1 Rules are cached

In order to avoid the reload of rules on each request the rules are cached on the first load from the middleware. This also means that if the rule are changed they have to be re-loaded somehow into the middleware.

One possibility is to just restart your server after you change the rules. In case this is acceptable for you just set the following variable into your settings file:

```
IPRESTRICT_RELOAD_RULES = False
```

The second possibility (which is the default behaviour) is to request a rule reload. The next time the middleware will receive a request the rules will be reloaded. There is a custom management command for reloading rules:

```
$ ./manage.py reload_rules
```

The advantage of this approach is that you don't have to restart your server every time you change your rules. The disadvantage is that on each request a query will be executed that selects the first row from the `reloadrulesrequest` DB table.

1.3.2 Changing the rules on a production server

In case you are using the default caching described above, remember that every time you change your rules you will have to follow up with running the `reload_rules` command.

However, the recommended way of changing your restriction rules is to make the changes using admin on a staging server and test them there, export them and then import them on the production server instead of changing rules directly on the production server.

After you changed the rules and are happy with them you can export them using:

```
$ ./manage.py dumpdata iprestrict --indent=4 --exclude iprestrict.ReloadRulesRequest >
↪ new_rules.json
```

Then you can copy the new_rules.json file to your production server and import them with the custom management command `import_rules`. For example if you've copied your rules file to `/tmp` you would use:

```
$ ./manage.py import_rules /tmp/new_rules.json
```

You would also have to reload the rules or restart the server (depending on what caching strategy you are using):

```
$ ./manage.py reload_rules
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search