Django Paranoi Documentation

Release 0.1

Andy McKay

October 03, 2014

Contents

Django Paranoia implements some of the OWASP Detection Points as outlined in this document:

https://www.owasp.org/index.php/AppSensor_DetectionPoints

Basically it's an attempt to find out when someone is trying to do something nasty to your site and log it.

Note: this is does not prevent any actions that might cause damage to your site. All the usual prevention measures must be taken to ensure you are not susceptible to XSS, SQL injection and so on.

Check out the source at:

https://github.com/andymckay/django-paranoia

CHAPTER 1

Overview

Django Paranoia is a library to try and detect nasty things going on your site. It does not prevent attacks, just logs them.

1.1 Setup

To install:

```
pip install django-paranoia
```

Add in the middleware:

```
MIDDLEWARE_CLASSES = (
        ...
        'django_paranoia.middleware.Middleware',
)
```

Hook up the reporters:

```
DJANGO_PARANOIA_REPORTERS = [
        'django_paranoia.reporters.log',
]
```

1.2 Using

The OWASP Detection points cover a large amount of the Django site, so each part is covered in seperately:

- *Sessions*
- *Forms*
- *Views*

1.3 Output

When you configure Django Paranoia, you can pass through a list of reporters. Current choices are:

- *django_paranoia.reporters.log*: send reports to a log file.
- *django_paranoia.reporters.cef_*: send reports to a CEF log (mostly Mozilla specific)

CHAPTER 2

Sessions

Change your session backend to use Django Paranoid sessions, which is a wrapper around the cache session. At this time other session backends are *not* supported.

To configure:

```
SESSION_ENGINE = 'django_paranoia.sessions'

MIDDLEWARE_CLASSES = (
        ...
        'django_paranoia.sessions.ParanoidSessionMiddleware',
)
```

When a session is created it will store the user agent and IP address of the session. If that changes at any time when the request is accessed, it will log. It will log each time the session is accessed while it's different.

It's assumed that IP address is allowed to change during a session. The user agent should not.

CHAPTER 3

Forms

Instead of using the builtin form libraries, use the *ParanoidForm* and *ParanoidModelForm*. If you do this, your forms will raise warnings if:

- more keys are submitted than are required. This may not be useful if you've got multiple forms in one request.
- less keys are submitted than are required.
- a key or value contains an ascii character less than 32 (but trn are ok).

For example:

```
from django_paranoia.forms import ParanoidForm

class Scary(ParanoidForm):
    ...
```

The log will contain the dodgy data.

CHAPTER 4

Views

Django Paranoia comes with the same decorators as Django: *require_http_methods*, *require_GET*, *require_POST*, *require_safe*. Use these the exact same way you would in Django. But instead import them from this library, for example:

```
from django_paranoia.decorators import require_POST

@require_POST
def something_sensitive(request, ...):
    ...
```

This will return a HTTP 405 Not Allowed response as usual. But it will also send a warning that an attempt to access with something other than a POST was made.

CHAPTER 5

Developers

To run the tests against multiple environments, install tox using `pip install tox`. You need at least Python 2.7 to run tox itself but you'll need 2.6 as well to run all environments. Run the tests like this:

```
tox
```

To run the tests against a single environment:

```
tox -e py27-django15
```

To debug something weird, run tests directly from the virtualenv like:

```
.tox/py27-django15/bin/nosetests
```

CHAPTER 6

Indices and tables

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