Security: Bring CORS and CSP into core

Index

Basic Information

- Name: Anvesh Mishra
- Pronouns: He/Him
- Email: anveshgreat11@gmail.com
- Github: https://github.com/Anv3sh
- Twitter: https://twitter.com/anv3shh
- Time Zone: UTC+5:30 (IST - India)
- Working Hours: 6-7 hours per day, anytime between 3 pm till 2 am IST
- Location: Ghaziabad, Uttar Pradesh, India
- Degree: 3rd year, B.Tech (Computer Science)

About Me

I am an undergrad Computer Engineering student from India. Currently, I am in my pre final year of study. Most of my work has been done using django along with the django rest framework. I have also been an active contributor to the Django framework and open source in general. I am still working on some tickets and actively helping people and suggesting ideas in the community.

Previous Works

Contributions to Django

- PRs(merged)
    - Fixed #32672, Added a regex and check to detect Primary Key in SQLite3 while working with inspectdb.
    - Fixed #32234, Made inspectdb inform about composite primary keys.
    - Fixed #32969, Fixed pickling of HttpResponse and subclasses.
    - Fixed #29186, Fixed pickling of HttpRequest and subclasses.
- PRs(work in progress)
    - #27704 TypedMultipleChoiceField as base field class for contrib.postgres.ArrayField with choices.
- PRs(closed)
    - #12075 Added wsgiorg.routing args support.
- Valuable comments on tickets
    - #28616
    - #32263

Contributions to django-csp

Found out a issue in documentation :

The documentation says to use the variable {{nonce}}, but that doesn't work. Reading the code it should be {{CSP_NONCE}} but as I checked the nonce.rst the changes are already done but aren't reflected on ReadTheDocs.

- Documentation for context processor · Issue #194 · mozilla/django-csp (github.com)

Abstract and Objectives

This proposal aims at adding support for Cross-Origin-Resource-Sharing(CORS) and Content-Security-Policy(CSP) into Django's core more importantly in the SecurityMiddleware. As security is a major concern and with CORS and CSP in action Django can prevent unauthorized requests and attacks like Cross-Site Scripting(XSS) and data injection attacks. Adding these policies to the core would be a great addition for developers who previously had to rely on the use of third party packages like django-cors-headers and django-csp.

Django currently does not ship with inbuilt support for using CORS and CSP as security policies. Most users depend on third-party libraries like django-cors-headers and django-csp in order to use the Cross-Origin-Resource-Sharing(CORS) and Content-Security-Policy(CSP). Since preventing unauthorized requests and attacks such as XSS and data injection are pretty severe,hence this proposal aims at adding CORS and CSP to core.

While going through django's documentation and code for django's security middleware, django-cors-headers and django-csp, implementing CORS and CSP into core seems very straightforward. However, there are many challenges that come on while studying it in detail. Therefore, for initial research, I explored the existing packages which enable support for CORS and CSP , namely, `django-cors-headers` and `django-csp`.

Existing Packages

1. django-cors-headers

A Django App that adds Cross-Origin Resource Sharing (CORS) headers to responses. This allows in-browser requests to your Django application from other origins. Adding CORS headers allows your resources to be accessed on other domains.

For using django-cors-headers you need to configure it in your Django settings and set at least one of three following settings:

- CORS_ALLOWED_ORIGINS
- CORS_ALLOWED_ORIGIN_REGEXES
- CORS_ALLOW_ALL_ORIGINS

CORS_ALLOWED_ORIGINS: Sequence[str]

A list of origins that are authorized to make cross-site HTTP requests. The origins in this setting will be allowed, and the requesting origin will be echoed back to the client in the Access-Control-Allow-Origin header. Defaults to [ ].

```
CORS_ALLOWED_ORIGINS = [
    "https://example.com",
    "https://sub.example.com",
    "http://localhost:8080",
    "http://127.0.0.1:9000",
]
```

CORS_ALLOWED_ORIGIN_REGEXES: Sequence[str | Pattern[str]]

A list of strings representing regexes that match Origins that are authorized to make cross-site HTTP requests. Defaults to []. Useful when CORS_ALLOWED_ORIGINS is impractical, such as when you have a large number of subdomains.

```
CORS_ALLOWED_ORIGIN_REGEXES = [
    r"^https://\w+\.example\.com$",
]
```

CORS_ALLOW_ALL_ORIGINS: bool

If True, all origins will be allowed. Other settings restricting allowed origins will be ignored. Defaults to False.
Setting this to True can be dangerous, as it allows any website to make cross-origin requests to yours. Generally you'll want to restrict the list of allowed origins with `CORS_ALLOWED_ORIGINS` or `CORS_ALLOWED_ORIGIN_REGEXES`.

Other optional settings:

- CORS_URLS_REGEX
- CORS_ALLOW_METHODS
- CORS_ALLOW_HEADERS
- CORS_EXPOSE_HEADERS
- CORS_PREFLIGHT_MAX_AGE
- CORS_ALLOW_CREDENTIALS

django-cors-headers also provides signals to create custom cors handlers that can be used to check if a given request should be allowed for the CORS policy.

2. django-csp

django-csp adds Content-Security-Policy(CSP) headers to Django applications.
django-csp provides multiple settings for all the CSP directives to use in Django like **`CSP_DEFAULT_SRC`**, **`CSP_SCRIPT_SRC`**, **`CSP_IMG_SRC`** etc.

django-csp decorators

django-csp also provides support for changing the policy on a per-view (or even per-request) basis by using decorators. The decorators provided are:

- `@csp(**kwargs)`

- `@csp_exempt()`
- `@csp_update(**kwargs)`
- `@csp_replace(**kwargs)`Proposed Implementation Details

NOTE : The proposed implementation consists of some code snippets. These are added for demonstration purposes only, actual implementations / workflows might be different.

CORS

I propose that CORS will be implemented by introducing the new **`CORSMiddleware`**. The following steps will be taken to add CORSMiddleware into core:

1) Firstly, a new CORSMiddleware is to be added into **`.\django\middleware\cors.py`**

```
class CORSMiddleware(MiddlewareMixin):
      ..........
```

2) Secondly, we need to create a **`CORS_SETTINGS={}`** attribute in **`./django/conf/global_settings.py`**.
   It will be a dictionary with following settings:

   - **`CORS_ALLOWED_ORIGINS`**
     A sequence of the allowed origins.

   - **`CORS_ALLOWED_ORIGIN_REGEXES`**
     A sequence of the allowed origin regexs.

   - **`CORS_ALLOW_ALL_ORIGINS`**
     A boolean to set if all origins should be allowed or not, defaults to False.

   - **`CORS_URLS_REGEX`**
     A string to set a single regex for all the allowed origins.

   - **`CORS_ALLOW_METHODS`**
     A sequence of all the allowed methods.

   - **`CORS_ALLOW_HEADERS`**
     A sequence of all the allowed headers. Sets the `Access-Control-Allow-Headers` response header

   - **`CORS_EXPOSE_HEADERS`**
     A sequence of the extra headers to be exposed to the browser.

- **`CORS_PREFLIGHT_MAX_AGE`**
- **`CORS_ALLOW_CREDENTIALS`**

The **`CORS_SETTINGS`** attribute with default values will look something like this:

```
CORS_SETTINGS = {
    "CORS_ALLOWED_ORIGINS": [],
    "CORS_ALLOWED_ORIGIN_REGEXES": [],
    "CORS_ALLOW_ALL_ORIGINS": False,
    "CORS_URLS_REGEX": r'^.*$',
    "CORS_ALLOW_METHODS": [
        "DELETE",
        "GET",
        "OPTIONS",
        "PATCH",
        "POST",
        "PUT",
    ],
    "CORS_ALLOW_HEADERS": [
        "accept",
        "accept-encoding",
        "authorization",
        "content-type",
        "dnt",
        "origin",
        "user-agent",
        "x-csrftoken",
        "x-requested-with",
    ],
    "CORS_EXPOSE_HEADERS": [],
    "CORS_PREFLIGHT_MAX_AGE": 86400,
    "CORS_ALLOW_CREDENTIALS": False,
}
```

3) **Decorators**: There will be a single decorator to rule it all say **`@cors(**kwargs)`** to set the cors headers for per-view customisation, rather than creating multiple decorators

which will make it unnecessarily granular. The user can pass the respective cors settings in the ****kwargs*** to set the headers. The arguments will be as following:

- ***allow_origins*** defaults to [*]
- ***allow_methods*** defaults to ["DELETE","GET","OPTIONS","PATCH","POST","PUT",]
- ***allow_credentials*** defaults to False
- ***allow_header*** defaults to ["accept","accept-encoding","authorization","content-type","dnt","origin","user-agent","x-csrftoken","x-requested-with",]
- ***allow_all_origins*** defaults to False
- ***expose_headers*** defaults to []
- ***max_age*** defaults 86400(one day)

The list of cors decorators that will be introduced are:

- **`@cors(**kwargs)`**
  This decorator will be used to set the different cors headers on a per-view basis.

```
@cors(allow_origins = [*], allow_credentials=False)
def my_view(request):
return render(request, 'my_template.html')
```

- **`@cors_exempt`**
  This decorator will be used to exempt a particular view for the global cors settings.

```
@cors_exempt
def my_view(request):
      return render(request, 'my_template.html')
```

- **`@cors_update(**kwargs)`**
  This decorator will be used to update/append new values to one or more cors headers and also to replace the globally defined values of one or more cors headers for a particular view.

```
@cors_update(allow_origins=["https://example.com","http://localhost:8080",]
,allow_credentials=True)
def my_view(request):
      Return render(request,'my_template')
```

4) **System checks**: The cors system checks are to be implemented in ***.\django\core\checks\security\cors.py***
5) CORS and CSRF:

CSP

I propose CSP to be added to the SecurityMiddleware. Following the design of the SecurityMiddleware the following steps will be taken to integrate CSP into django core:

1. **Settings**: We will have the following settings that will be added to ***.\django\conf\global_settings.py***:

    - **`SECURE_CSP:Sequence[str]`**
      A dictionary of all the directives that need to be set.

```
SECURE_CSP = {
"default-src": "'self'",
}
```

    - **`SECURE_CSP_REPORT_ONLY: Sequence[str]`**
      A dictionary of all the csp directives with **report-to** as the necessary directive that needs to be set in report-only mode. A warning needs to be added for if the user does not set the **report-uri** directive the warning can be as "report-to not set in csp report only mode".

```
SECURE_CSP_REPORT_ONLY = {
"default-src": "'self'",
"report-to": "/csp-report-endpoint/",
}
```

    - **`SECURE_INCLUDE_NONCE_IN: str`**
      A boolean to set directive for nonce usage. Default value False

```
SECURE_INCLUDE_NONCE_IN = "default-src"
```

    - **`SECURE_CSP_EXCLUDE_URL_PREFIXES: Sequence[str]`**
      A sequence of url prefixes that need to be excluded from csp.

2. **Decorators**: This proposal also adds up a couple of csp decorators in ***.\django\views\decorators\csp.py*** for per-view customisation. The list of csp decorators to be introduced are:

    - **`@csp(**kwargs)`**
      This decorator will be used to set the different policies mentioned above on per view.

```
@csp("script-src" = ["'self'", "cdn.example.net"])
def my_view(request):
      return render(request,"my_template.html")
```

- **`@csp_exempt`**
  This decorator will be used to exempt a particular view from the global csp policy set by the user.

```
@csp_exempt
def my_view(request):
      return render(request, "my_template.html")
```

- **`@csp_update(**kwargs)`**
  This decorator will be used to update/replace the value of one or more csp policies.

```
@csp_update("img-src"= "imgsrv.com")
def my_view(request):
      return render(request,"my_template.html")
```

- **`@csp_report_only(**kwargs)`**
  This decorator will be used to add the Report-Only header to the response object for a particular view in order to use the Report-Only functionality i.e just reporting that the csp policy could have blocked a particular inline script if it was enabled.

```
@csp_report_only("default-src"= "'self'", "report-to"=
"/csp-violation-report-endpoint/")
def my_view(request):
      return render(request,"my_template.html")
```

3. **Context Processor**: After configuring CSP the inline scripts will stop executing to overcome this problem this proposal introduces nonce support which is implemented by the **`CSP_INCLUDE_NONCE_IN`** setting so as to enable the inline scripts.
   Also, for enabling a particular script to execute we will include a context processor that adds a variable **`CSP_NONCE`** that can be added to the scripts, style, image and various other tags to enable nonce in them.The context processor will be implemented in **`.\django\template\context_processors\nonce`**

```
def nonce(request):
    nonce = request.csp_nonce if hasattr(request, 'csp_nonce') else ''
```

```
    return {
        'CSP_NONCE': nonce
    }
```

django-csp has this feature already implemented so we just need to take references from its API, it'll be an easy to implement thing.
What is nonce?
nonce is a shorthand for a `number used only once`. This will be generated by the ***_make_nonce()*** method whose location still is to be thought of.

4. The csp system checks are to be implemented in **`.\django\core\checks\security\base`**

CSP Mock Implementation

SecurityMiddleware

```
class SecurityMiddleware(MiddlewareMixin):
    def __init__(self, get_response):
        super().__init__(get_response)
        ......................
        self.csp = settings.SECURE_CSP
        self.csp_report_only = settings.SECURE_CSP_REPORT_ONLY
        self.csp_nonce = settings.SECURE_CSP_INCLUDE_NONCE_IN

    def _make_nonce(self,request):
        if not getattr(request, '_csp_nonce', None):
            request._csp_nonce = (
                base64
                .b64encode(os.urandom(16))
                .decode("ascii")
            )
        return request._csp_nonce

    def process_request(self, request):
        path = request.path.lstrip("/")
        nonce = partial(self._make_nonce, request)
        request.csp_nonce = SimpleLazyObject(nonce)
        if (
            self.redirect
            and not request.is_secure()
```

```
            and not any(pattern.search(path) for pattern in
self.redirect_exempt)
        ):
            host = self.redirect_host or request.get_host()
            return HttpResponsePermanentRedirect(
                "https://%s%s" % (host, request.get_full_path())
            )

    def process_response(self, request, response):
       ......................
        if self.csp:
            csp_header = ";".join(f"{k}{v}" for k,v in self.csp.items())
            if self.csp_nonce:
                nonce = getattr(request, '_csp_nonce', None)
                csp_header += "; 'nonce-%s'" % nonce
                response.headers["Content-Security-Policy"] = csp_header
        return response
```

Context Processor

```
def nonce(request):
    nonce = request.csp_nonce if hasattr(request, 'csp_nonce') else ''

    return {
        'CSP_NONCE': nonce
    }
```

The tests for CORS and CSP will be written in **`tests.check_framework.test_security`** **and** **`tests.middleware.test_security`**.
The docs for this proposed implementation will be written in ***docs/ref/checks.txt***, **`docs/ref/settings.txt`**, **`docs/topics/settings.txt`**, **`docs/refs/middleware.txt`**.

Stretch Goals

Timeline

4th May 2023 to 28th May 2023 (Community Bonding Period)

I am currently working on a few tickets and I would like to continue to work on more tickets. This would help me build a better understanding of django and its various components. In addition, I would also be well versed with the contribution practices followed by the django software foundation and the coding style. I would further like to utilize this period by discussing my proposed implementation and refine it. For this, I would interact with the django-developers-mailing-list as well as take feedback from my mentors.

29th May 2023 - 31st June 2023

- Add all CSP settings to `django.conf.global_settings`
- Document all CSP settings in docs/ref/settings.txt

1st June 2023 - 21st June 2023

- Implement `CSP` in `django.middleware.security.`
- Add `nonce` context processor.
- Write tests for `CSP`.
- Write tests for context processors.
- Document `CSP` in docs/topics/security.txt
- Create docs/howto/csp.txt and write a short informative guide on best practices for `CSP` and how to prevent `CSP` pitfalls and common mistakes.

22nd June 2023 to 2nd July 2023

- Implement all `CSP` decorators.
- Write tests for all `CSP` decorators.
- Document all `CSP` decorators in docs/ref/security.txt

3rd July 2023 to 13th July 2023

- Add system checks to ensure `CSP` settings are configured correctly.
- Add system checks to ensure `CSP` settings are assigned the correct data type and format.
- Add security checks to make sure CSP policies don't have trivial bypasses and vulnerabilities
- Add tests for all new system checks.

- Document all the new system checks in `docs/ref/checks.txt.`

14th July 2023 to 16th July 2023

- Ask for final reviews on **`CSP`** PR
- Merge **`CSP`** PR into core

17th July 2023 to 23th July 2023

- Add all **`CORS`** settings to `django.conf.global_settings`
- Document all **`CORS`** settings in docs/ref/settings.txt

24th July 2023 to 13th August 2023

- Implement `CORSMiddleware` in `django.middleware.cors`
- Write tests for `CORSMiddleware.`
- Write tests for context processor.
- Document `CORSMiddleware` in docs/ref/middleware.txt.
- Create docs/ref/cors.txt and document Cross Origin Request Sharing.
- Create docs/howto/cors.txt and write a short guide on best practices for `CORS`.

13th August 2023 to 20th August 2023

- Implement all CORS decorators
- Write tests for all CORS decorators
- Document all CORS decorators in docs/ref/cors.txt

21st August 2023 to 28th August 2023

- Implement system checks to make sure CORS settings are configured correctly and have no errors.
- Add system checks to ensure `CORS` settings are assigned the correct data type and format.
- Add tests for all new system checks.
- Document new system checks in docs/ref/checks.txt

29th August 2023 to 31st August 2023

- Ask for final reviews of CORS PR.
- Merge CORS PR into core.

References

https://github.com/django/django/pull/5776 - PR on django-csp to contrib
https://code.djangoproject.com/ticket/15727 - Add support for Content-Security-Policy (CSP) to core
https://github.com/django/django/pull/3550 - PR on Add support for Content-Security-Policy header support for SecurityMiddleware
https://csper.io/blog/multiple-policies - UsingCSP with multiple policies
https://github.com/adamchainz/django-cors-headers/pull/536 - PR to Remove CORS_REPLACE_HTTPS_REFERER and CorsPostCsrfMiddleware from django-cors-headers