Project on the OWASP 2017 list

This project depicts 5 flaws on the OWASP 2017 list security issues and supplies corresponding correction recommendations

Link to Project: https://github.com/melting8snowman/electricsite

Installation Instructions

Installation instructions as per readme.md

Clone source from github by:

```
git clone git@github.com:melting8snowman/electricsite.git
```

Migrate data by running from main directory:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

start virtual server run from main directory

```
python manage.py runserver
```

FLAW 1: Security Misconfiguration

Source link:
https://github.com/melting8snowman/electricsite/blob/main/electricsite/settings.py (Row 27)

Description:

Security misconfiguration is apparently one of the most common issues on the OWASP list. That can happen in multiple different ways. Our example is a very basic one which can happen very easily – after moving into production one has forgotten to disable the DEBUG option.

How to fix it:

In the settings.py file we would instead of

DEBUG = True

use

DEBUG = False

Django actually has quite a nice feature for checking the security configuration, the Security checklist can be accessed through the command

```
python manage.py check --deploy
```

In our case there would still be quite a few to fix

WARNINGS:

?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS setting. If your entire site is served only over SSL, you may want to consider setting a value and enabling HTTP Strict Transport Security. Be sure to read the documentation first; enabling HSTS carelessly can cause serious, irreversible problems.

?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless your site should be available over both SSL and non-SSL connections, you may want to either set this setting True or configure a load balancer or reverse-proxy server to redirect all connections to HTTPS.

?: (security.W009) Your SECRET_KEY has less than 50 characters, less than 5 unique characters, or it's prefixed with 'django-insecure-' indicating that it was generated automatically by Django. Please generate a long and random value, otherwise many of Django's security-critical features will be vulnerable to attack.

?: (security.W012) SESSION_COOKIE_SECURE is not set to True. Using a secure-only session cookie makes it more difficult for network traffic sniffers to hijack user sessions.

?: (security.W016) You have 'django.middleware.csrf.CsrfViewMiddleware' in your MIDDLEWARE, but you have not set CSRF_COOKIE_SECURE to True. Using a secure-only CSRF cookie makes it more difficult for network traffic sniffers to steal the CSRF token.

?: (security.W018) You should not have DEBUG set to True in deployment.

?: (security.W020) ALLOWED_HOSTS must not be empty in deployment.

...

FLAW 2: Using vulnerable and outdated components

Source link:

https://github.com/melting8snowman/electricsite/blob/main/polls/views.py (Row 17)

Description:

Using components in python code is a give nowadays as one does not want to rewrite the contire code every time but to use libraries and components which are ready-made to be utilized. There is a danger for cybersecurity here as the component might be outdated or even worse typosquatted and thus malicious. Typosquatting means creating a rogue model which resembles the correct one but is slightly different by written name.

Compare

```
pip install python-dateutil
```

to

```
pip install python3-dateutil
```

The first is the correct package, latter is malicious package. Then using `import dateutil` in your code will add the malicious component into your code.

Issue reported here in more detail:
https://github.com/dateutil/dateutil/issues/984

And here

https://news.sophos.com/en-us/2019/12/05/machine-raiding-python-libraries-squashed-by-community/

How to fix:

Detect which one you have installed by running

`pip freeze`

If all is well, you should see something like this

python-dateutil==2.8.2

but if you see

python3-dateutil==x.x.x

you have the malicious package installed. Please delete the package and install the correct one.

In short, use newest patched versions and be careful when spelling component names.

FLAW 3: Broken Authentication

Source link:

https://github.com/melting8snowman/electricsite/blob/main/electricsite/settings.py (Rows 91-104)

Description:

Authentication and session management functions in web applications are used to verify the identity of the user and if they are incorrectly implemented, they allow attackers to compromise passwords, keys or sessions tokens. Attackers can exploit broken authentication with credential stuffing, automated brute force and dictionary attacks to gain other users' indentities.

As per OWASP an application has broken authentication vulnerability if there are for example the following authentication weaknesses:

- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".

- Uses weak or ineffective credential recovery and forgot-password processes, such as “knowledge-based answers”, which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords

(source: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication)

Our application accepts short passwords, furthermore the admin user currently has the above mentioned combination admin/admin which is specifically mentioned as an authentication vulnerability.

How to fix it

Password management is something that should generally not be reinvented unnecessarily, and Django provides lots of configurable password management options and tools. Django uses hashed passwords by default using the PBKDF2 algorithm with a SHA256 hash.

```
#PASSWORD_HASHERS = [
#    "django.contrib.auth.hashers.PBKDF2PasswordHasher",
#]
```

In addition there are multiple options for password management which can help us with the other authentication vulnerabilities relating to passwords. In order to make longer passwords mandatory, we can utilize password validation with the option to set password length being minimum 12 digits. Furthermore we can disallow username / password combinations which are too close to each other using UserAttributeSimilarityValidator. These can be activated with uncommenting the below rows.

```
AUTH_PASSWORD_VALIDATORS = [
    #{"NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityValidator",},
    {"NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    #    "OPTIONS": {
    #        "min_length": 12,
    #    },
    },
```

FLAW 4: (SQL) Injection

https://github.com/melting8snowman/electricsite/blob/main/polls/views.py (Rows 95-130 in method ContestView)

Description: SQL injection, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. A malicious user can access this information by including code into user inputs and thus may include sensitive data not initially designed to be fetched by the application. This data could include any number of items, including sensitive company data, user lists or private customer details or basically any data in the database.

How to fix it:

There are multiple ways of preventing SQL injections, for example input validation and parameterized queries. In our example case, we are using SQLite, which can protect against SQL injections if instead of stringing together a SQL query, we specify user-supplied data as part of the params.

By using the placeholder ?, SQLite automatically treats the data as input data, so it cannot not interfere with the parsing of the actual SQL statement.

SO instead of

db.execute("INSERT INTO Contest VALUES(" + variable + ")");

we use

db.execute("INSERT INTO Contest VALUES (?)", variable);

FLAW 5: Insufficient Logging and monitoring

Description:

https://github.com/melting8snowman/electricsite/blob/main/electricsite/settings.py (Rows 42, 53, 106-126)

Currently there is no security logging in the application.

Application logging should always be included for security events. Application logs are invaluable data for:

- Identifying security incidents
- Monitoring policy violations
- Establishing baselines
- Assisting non-repudiation controls (note that the trait non-repudiation is hard to achieve for logs because their trustworthiness is often just based on the logging party being audited properly while mechanisms like digital signatures are hard to utilize here)
- Providing information about problems and unusual conditions
- Contributing additional application-specific data for incident investigation which is lacking in other log sources
- Helping defend against vulnerability identification and exploitation through attack detection

(source: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

How to fix it:

We can add it for example by using the Django automated security logging by installing the component:

```
pip install django-automated-logging
```

and then configuring the settings file to include the logging:

Uncomment the following lines in settings.py

Line 42: `#”automated_logging”`

Line 55: `#”automated_logging.middleware.AutomatedLoggingMiddleware”`

Lines 106-126

```
## Uncomment to enable direct logging with log level info
#LOGGING = {
#    "version": 1,
#    "disable_existing_loggers": False,
#    "handlers": {
```

```
#         "console": {
#             "class": "logging.StreamHandler",
#         },
#     },
#     "root": {
#         "handlers": ["console"],
#         "level": "WARNING",
#     },
#     "loggers": {
#         "django": {
#             "handlers": ["console"],
#             "level": os.getenv("DJANGO_LOG_LEVEL", "INFO"),
#             "propagate": False,
#         },
#     },
#}
```

Furthermore if logging to file would be desired, we could add as LOGGING section loggers the following (Console logging included in project settings, file logging is additional):

```
LOGGING = {
    "version": 1,
    "disable_existing_loggers": False,
    "handlers": {
        "file": {
            "level": "DEBUG",
            "class": "logging.FileHandler",
          "filename":  BASE_DIR / "logs/debug.log",
        },
    },
    "loggers": {
        "django": {
            "handlers": ["file"],
            "level": "DEBUG",
            "propagate": True,
```

```
            },
        },
    }
```

and finally running the code

```
python manage.py migrate automated_logging
```