Password manager website:
Repository: https://github.com/Robomarti/cyberSecurity23Project1/
Installation guide:

Clone the repository. Navigate to the directory and use the command “python manage.py runserver”. if this fails, you might have to use the command “python3 manage.py runserver”. Then you can access the website at http://127.0.0.1:8000/ while the server is running. MAKE SURE THAT THE URL STARTS WITH HTTP AND NOT HTTPS.
In this project, I am using the 2021 version of the OWASP list, and Django in a class-based view style.

FLAW 1:
Link to the problem / fix:
https://github.com/Robomarti/cyberSecurity23Project1/blob/master/password_manager/views.py#L54 , also lines 79 and 93 of the same file.
A01:2021-Broken Access Control
This problem comes from users being able to access other users' information just by pampering the URL. A user can see other users' passwords by changing the website address to http://127.0.0.1:8000/password/x/ where x is the number of the password that they are trying to access. Users can also delete (http://127.0.0.1:8000/password-deletion/x/), or change (http://127.0.0.1:8000/password-updating/x/) other users’ passwords.

We can fix this by checking whether the user who is accessing the password is the user associated with the password. We do this by overwriting a part of the get function on the PasswordDetails, PasswordUpdating, and the PasswordDeletion class. If the user is the user who should have access to view the password, then we return to the process of routing them to the correct page. Otherwise, we redirect the user trying to access the page to their own password list. In addition to this, we could randomize the URLs using a Python package called hashids so that guessing other users’ passwords’ ids would be harder, but I decided to not include the package in my project to make it easier to review. Implementing this on my own would probably also be too time-consuming.

FLAW 2:
Link to the problem / fix:
https://github.com/Robomarti/cyberSecurity23Project1/blob/master/mysite/settings.py#L76
A02:2021-Cryptographic Failures
This problem comes when the server transfers data using the HTTP protocol. This compromises the security of the users’ passwords and usernames by transferring critical information in clear text. Note that this guide only works if the website has SSL, so testing it would probably be impossible for the reviewer. Please do not uncomment the marked parts as that would only lock you out of the website until you reverted the changes. In case this happens, after reverting the changes, when accessing the website your URL starts with http and not https. With that out of the way, here is the fix.

We can fix this problem by forcing Django to use the HTTPS protocol by writing “SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')” to the settings.py file. We also have to write the line “SECURE_SSL_REDIRECT = True” so that our website redirects requests made with HTTP to HTTPS. We can also ensure that our cookies are safe by assigning SESSION_COOKIE_SECURE and CSRF_COOKIE_SECURE as True. To be extra safe, we can set SECURE_HSTS_SECONDS = 30 and set it eventually to 31536000 when we have made sure everything works. This allows browsers to refuse to communicate with the website non-securely, reducing the risk of a man-in-the-middle attack. We will also set SECURE_HSTS_INCLUDE_SUBDOMAINS as True to refuse insecure connection to subdomains.

FLAW 3:
A05:2021-Security Misconfiguration
This problem comes from leaving the superuser in the database with an obvious username “admin”, and an obvious password “admin”.

This problem can be solved by changing the superuser's username and password, or deleting the superuser and creating a new one. The details can be changed in the admin menu at http://127.0.0.1:8000/password/admin/ . A new superuser can be created by running the command “python manage.py createsuperuser”, and if you get an error on Windows you can use the command “winpty python manage.py createsuperuser”.

FLAW 4:
Link to the problem / fix:
https://github.com/Robomarti/cyberSecurity23Project1/blob/master/mysite/settings.py#L121
A07:2021-Identification and Authentication Failures
This problem comes from the web application accepting weak passwords when creating a user and having no protection against brute force attacks.

Django automatically enforces strong passwords when importing the template, but some other frameworks might not, so it can be useful to check how Django does it and recreate the system in your own project. However, in production, it can be useful to turn this enforcement off. For example, if you want to test new users and you don’t want to memorize different passwords. The option can be turned on or off from the settings.py file by adding or removing the values inside the AUTH_PASSWORD_VALIDATORS list. The risk of brute force attacks can be easily mitigated by installing and using a library like django-defender, which protects the server from brute force attacks.

FLAW 5:
Link to the problem / fix:
https://github.com/Robomarti/cyberSecurity23Project1/blob/master/mysite/settings.py#L26
as well line 47 of the same file.
A09:2021-Security Logging and Monitoring Failures
This problem comes from Django not saving the logs to the backend, so in case of a bug or security breach, developers won’t know anything about them. Also, by default, when an error

happens django gives a lot of details and metadata to the user. This is a security risk since the user should not be given information about the inner workings of the backend.

This can be fixed by adding a few lines to the settings.py file. First we create a logs directory where we wanted to save the logs, in my case inside the password_manager directory. Then we add a certain value for LOGGING in the settings.py file as seen in my project. To fix the giving too many details during errors problem, we just need to assign the DEBUG as False, instead of its default value of True.