WebChatApp Document

Repository(public): https://github.com/Miqqou/SecPro-webchat

Contents

1. Description

A web application made with Django web framework for python. The Application is made for sending a message to another user without anyone else being able to read it (e.g on the server side). It also includes login, logout, registration, and admin panel as features.

The site has admin, home, profile, chat, inbox, login, logout and registration pages. The admin panel is Django default. And it can be accessed by 'site/admin/'. From there it is possible to delete users and update users to superuser.

To send a message user must first go to the chat page ('host/webchatapp/chat'). Then they can input the recipient's username and message content. After that message is sent by clicking the send button. To read a message user must first go to the inbox page ('host/webchatapp/inbox'). Then they must input their password to read the messages.

Home, profile, login and logout should be self-explanatory. After 5 failed logins user will be put on a 10 min cooldown. This applies to the inbox messages also. User registration is also count as a fail to prevent user account creation spam. Cooldown is user account and ip based.

Registration has some limitations. Username must be 4-20 characters long containing only allowed symbols (@ . + - _), letters and numbers. Password has some validation, which requires it to be at least 12 characters long, contain 1 symbol and capital letter, be distinct from username and not too common. Password cannot be reset.

2. Structure of the program

Noteworthy files for the project.

- webchat/settings.py
  - All settings for Django project.
- webchatapp/templates
  - All the html files
- webchatapp/static
  - CSS styling and pictures
- webchatapp/.py files
  - form.py
    - has the customized registration form of Django.
  - models.py
    - has customized database models.
  - tests.py
    - has all the implemented unit tests.
  - urls.py
    - has all the URLs of the website structure.
  - validators.py
    - has customized password validators to demand user to use stronger password for the registration.
  - views.py
    - has all the functionality of the web pages.

3. Secure programming solutions

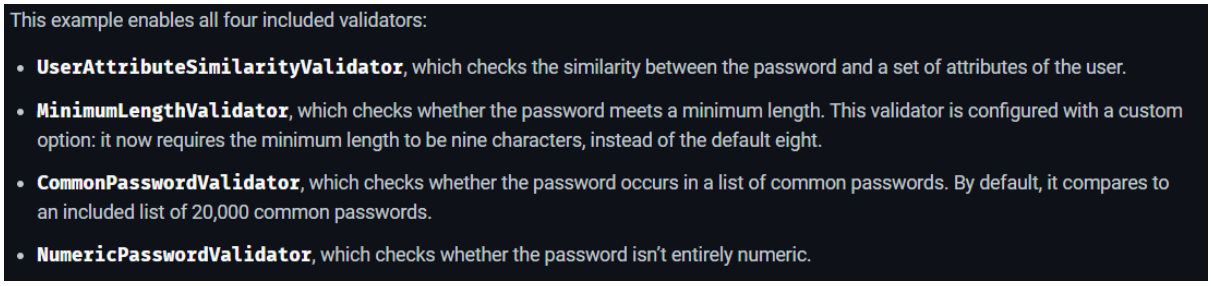
The application's biggest issue was, how to encrypt the messages so that only the recipient can decrypt then. I concluded that asymmetric encryption was needed. But then the problem was, where to store the user's private key. Cookies aren't a secure place and because it was a web application storing the key on the user's device wouldn't be accessible or possible. The solution was to store the private key on the database but symmetrically encrypted with the user's password.

When users registrates their account, a private-public-key pair is generated for them. A key derivation function (PBKDF2HMAC with SHA256 algo) is used to derive key from the user password. The key is used to encrypt the private key. Encrypted private key is stored in the database with the public key and salt used for the encryption key. Private key's PEM format serialization is also encrypted with the user's password using the serialization library's bestAvailableEncryption() function, which is curated best available encryption at the time that is supported by the system. (In the future this can be also changed to NoEncryption() if this is decided to be useless or too resource heavy to encrypt the private key two times.)

When user sends a message to another user, the message is encrypted asymmetrically with the public key of the recipient. Messages are encrypted with padding (OAEP with SHA256 mask generation function), ensuring that message content isn't predictable, or the length is known. Messages are stored encrypted in the database.

When the recipient wants to read the message, they need to decrypt the message with their private key, which is stored in the database encrypted. The recipient only needs to give their password and the system derives the encryption key which is then used to decrypt the private key. Then the password is used to decrypt the serialization level encryption. Then the private key is used to decrypt the message.

User password is hashed by argon2 and stored in the database. Argon2id variety version 19 is used with the Django's default settings (memory cost: 102400, time cost: 2, parallelism: 8). Password validators are used to ensure that user creates a stronger password. Django's default validators are shown in picture 3.1. Also I have created custom ones: SymbolValidator and CapitalValidator, for ensuring that at least one symbol and capital letter are included. Password minimum length is set to 12 characters.

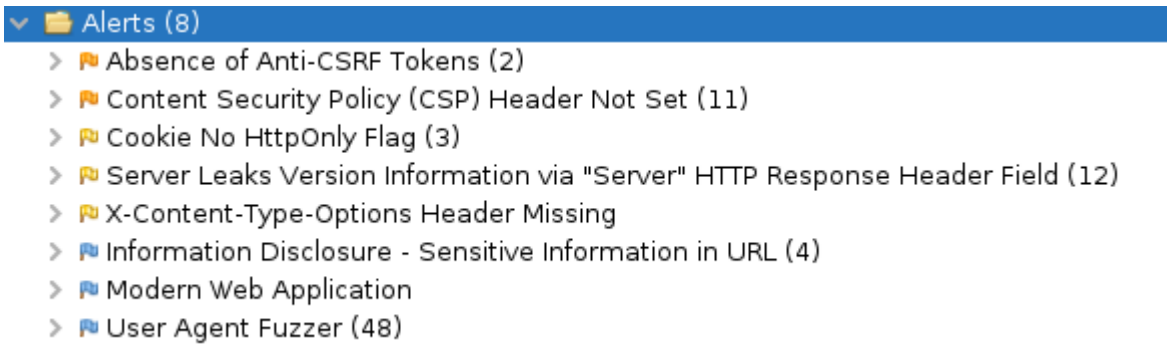


Picture 3.1. Django's password validators. https://docs.djangoproject.com/en/3.2/topics/auth/passwords/#module-django.contrib.auth.password_validation

4. Security testing

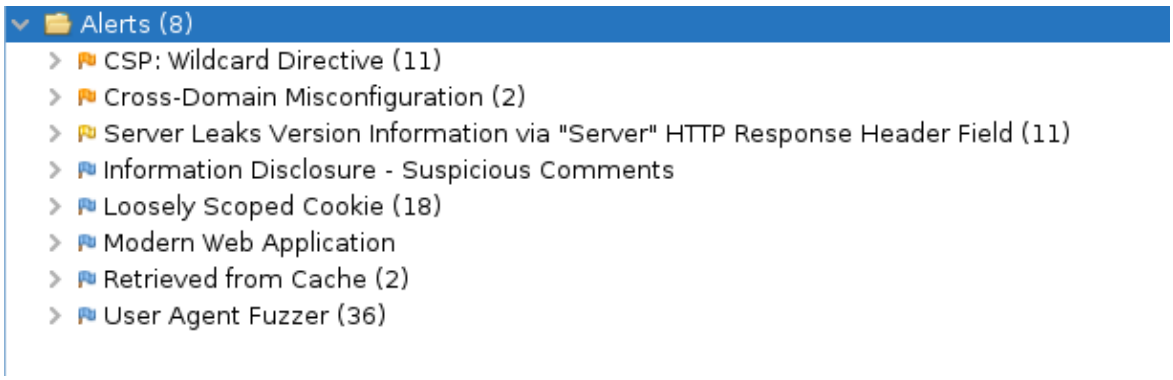
Some unit tests were implemented for testing the functionality of axes login/password failure cooldown. These should be updated in the future to cover more parts of the application.

For automated security testing OWASP's ZAP was used, which is a web application security scanner. As shown in picture 4.1 some problems were found.



Picture 4.1. The first OWASP ZAP testing results.

Login page form had an error which was flagged as an anti-CSRF token. This has been fixed now. Then the application lacked any content security policies. These were added. This didn't solve the flagging completely, because bootstrap elements are allowed, and they are marked as 'wildcard'. Cookies were set HttpOnly, which makes them more secure. Alerts that persisted are shown in picture 4.2.



Picture 4.2. OWASP ZAP testing results after applying fixes.

The web/application server is leaking version information via the "Server" HTTP response header. Django settings should be configured to suppress the "Server" header or provide generic details. This could be done using some custom middleware.

Manual security testing was done by following OWASP top 10 (https://owasp.org/www-project-top-ten/) and Mozilla security cheat sheet (https://infosec.mozilla.org/guidelines/web_security). Because time was limited, most of the items were just checked that they have implementation, but the implementation wasn't actually manually tested. A10:2021-Server-Side Request Forgery, A02:2021-Cryptographic Failures, A07:2021 – Identification and Authentication Failures were tested. The results are from manual testing done after the automatic testing.

Results:

- CSRF token is implemented. Logout of user can be exploited if user clicks link 'host/webchatapp/logout'. But this doesn't count as 'destructive change'.
- Database doesn't show sensitive information.
- Login has failure cooldown. Registration validates password to be stronger. URL doesn't contain sensitive information. Logout invalidates session. Most pages need login so anything sensitive isn't shown.

Other Results:

- Database doesn't have logging on server side.
- HTTPS must be implemented. (was should be previously)
- Some configurations could be more strict.

5. Acknowledgments and improvements

MUSTs:

- Site must be configured to use HTTPS. And should be tested with at least Google's nogotofail -network security testing tool (https://github.com/google/nogotofail/blob/master/docs/getting_started.md).
- Password validation cooldown must be fine-tuned. The current way the cooldown works Dos attack is possible to make some one's account on cooldown for ever. Configure it to use cache based cooldown and ip based, remove the user account based cooldown.
- PBKDF2HMAC iterations amount must be increased every year. Instead of using this, scrypt would be better to implement in the future or some more modern key derivation function, for example argon2.

SHOULDs:

- Inbox messages filtering doesn't have functionality implemented. Message decryption slows down if many messages are in inbox. There should be filtering or only the newest messages would be decrypted.
- Base.html and navbar.html are mostly bootstrap (https://getbootstrap.com/). See the licence file in templates. Bootstrap HTML resources should be replaced and csp should be adjusted accordingly. Now all boostrap are allowed content which may poses security risks.
- Static files should be excluded in the csp. At the moment the admin panel doesn't have styling because of this, which makes it harder to use.
- Password minimum length should be 14 characters to ensure strong password.

COULDs:

- Message max length is only 500 characters (RSA key 512 bytes – 11 bytes long padding), this could be increased with dividing the message in 500 characters long parts and then encrypting and decrypting the parts separately.
- More unit tests could be implemented so that most of the testing would be automated.
- Password could be reset in the future. When reset the public-private-key pair would be regenerated accordingly. Downside is that the old messages wouldn't be accessible anymore. Also, for resetting password, users should be able to set email address for their account.