HE2B ESI

SECG4 - Security

Project of secure client/server file system storage

Authors:
Billal Zidi (54637)
Bryan Gregoire (53735)

Teacher:
Pierre Hauweele

Group:
D121

Academic Year:
2021 - 2022

Table of contents

Introduction

The goal is to create a file storage and sharing system where users can upload, edit and delete their files and where the information is secure. This project includes registration, user login and file management.

Additional Information

Numerous software exists to create online code, in this project the choice fell on an extremely well-known framework: "Laravel".

The latter enjoys a very large community full of developers. Therefore, several optimizations are already present and facilitate the integration of security. Moreover, if a flaw is found, it is very quickly corrected, and it will be enough to keep up to date to have a solid site base.

The latter is based on PHP and therefore requires a server and a database for data persistence. Moreover, in order to correctly modify a Laravel project, it is necessary to have Composer and NodeJS installed.

Safety information

- Use of an account specific to the system in the DB. This allows you to protect yourself if there is a flaw in the system, in fact the modifications/losses of data made with this user will not affect the rest of the application.
- The account associated with the chat only has access to SELECT, UPDATE, DELETE and INSERT queries so that it cannot cause major damage.
- The debug version is disabled in the ".env" file, which protects the vulnerabilities more if errors occur (in line with the improvements mentioned below at the server level).
- Passwords are encrypted using the "password_hash()" function (bcrypt compatible). Now this is defined as the encryption standard for passwords on the web. Each password is encrypted differently even if the words are identical. There is a salt and with the option
- There is a salt and with the option "rounds" to 12, there is a hash loop of 12 times on purpose. This is an algorithm that is deliberately slower to strongly impact the "rainbow attack" among others.

- The messages are encrypted before insertion in the server and decrypted at the client using several HASH directly in SQL, this allows a prevention in case of leak of the database and no user has access to this encryption / decryption key (everything happens on the server side).

- Given the choice of a recent framework with recent updates, all the flaws discovered at the moment are corrected.

- It is necessary to keep up to date with the advances of Laravel in order to be always up to date from a security point of view.

- The list of the most common vulnerabilities is verified and secured (OWASP): Injection, broken authentication, exposure of sensitive data, external entitie.

- XML, broken access control, incorrect security configuration, XSS cross-site scripting, insecure deserialization, use of components with known vulnerabilities, insufficient logging and monitoring.

- After 5 login/registration attempts, the user is blocked for 1 minute. This prevents too fast requests and brute force.

- User IDs are UUIDs, this prevents a user from modifying the URL to try to find out the ID of another recipient (even if checks are made).

- All forms have a specific Laravel tag: @CSRF to protect against CSRF flaws that allow a user to perform an action that was not intended by the base user.

- Passwords must be at least 8 characters long and respect the format (uppercase, lowercase, special character and number) to complicate password recovery attacks.

- User actions are recorded in the Laravel folders:

  "/storage/logs" and are configurable in the server (see below). It is important to important to be able to see all the actions of a user to detect an intrusion attempt or in case of a problem to understand where it comes from in order to fix the problem quickly.

- No risk of SQL injection because all the queries are prepared automatically when using the DB interface methods.

- No risk of XSS vulnerability since users cannot enter any code and all codes are automatically escaped on display (but see server configuration below).

- All forms are checked on the server side as well to block a data with information that he does not have access to create an unauthorized action.

- The APIs are secured and authenticity checks are performed against the connections. A user A will not be able to access the messages of a user B even if he knows the link to which the API points.

- The content of the code loaded by Javascript in the messages is inserted through the "text" method in order to prevent the insertion of HTML on the

client side.

Even if the configurations related to the code are completed, several configurations are to be done to greatly optimize the security of the website so that it is viable in production:

- After deployment on a domain name, an SSL certificate would be needed to secure the transfer of information between the client and the server.
- With the SSL certificate, all cookies could be HTTPS-ONLY and will be more secure than simple cookies.
- With a domain name, it would be possible to link Cloudflare to limit DdoS attacks, bot, captcha etc.
- Install Fail2ban on the server to handle network attacks.
- We should add iptables rules in the "**installation/firewall.sh**" file on the server to limit ping, DdoS attacks and temporarily ban those who make too many requests.
- We should block too many requests from the same user (avoid brute force or server overload) with a configuration on the PHP + MySQL + Firewall side.
- PHP should be configured not to display any errors to the client (displaying errors allows a better understanding of the architecture of the code) as well as to allow only cookies from the current server.
- You should activate the server logs (NGINX/APACHE), PHP and MySQL to see all the requests made and to detect if there is a problem or attempts at intrusion.
- It would be necessary to add in the configuration of the server (NGINX in this case) rules blocking the XSS flaws, securing the COOKIES and validating the authentication of the server (see file "installation/xss.txt").

General conclusion

The data is secured on the client side as well as on the server side, however to secure the whole system it is necessary to secure the server side as well.

The framework is a very important help, but it is necessary to regularly keep up to date with updates since a flaw discovered on a framework can very quickly be exported to other sites that use this same framework.

It is impossible to be totally secure from all vulnerabilities (DdoS for example) but it is possible to limit their impact and to increase considerably the difficulty for hackers (using an anti-brute force system with iptables or Cloudflare increases enormously the computation time needed to test all passwords).