SECG4 : Scientific Report - Secure

Group

- 56459 - Antoine Ghigny
- 55019 - Ian Cotton

Contents

1 Answer to checklist questions

1.1 Do I properly ensure confidentiality?

- Are sensitive data transmitted and stored securely ?
  - Sensitive data is transmitted securely between the client and the server using the SSL/TLS protocol.

    SSL stands for Secure Sockets Layer. In short, it is the standard technology for securing an Internet connection and protecting sensitive data sent between two systems, thus preventing criminals from reading and modifying the transferred information, including potential personal data. The two systems can be server and client (e.g. a shopping website and a browser) or server to server (e.g. an application containing personally identifiable information or payroll information). It does this by ensuring that any data transferred between users and sites, or between two systems, remains unreadable. It uses encryption algorithms to scramble data in transit, preventing hackers from reading it as it is sent over the connection. This information can be sensitive or personal, including credit card numbers and other financial information, names and addresses.

    TLS (Transport Layer Security) is just an updated, more secure version of SSL. We always refer to our security certificates as SSL because it is a more commonly used term.

    Private channels have been set up with pusher to ensure end-to-end encryption. This means that no one except authorised subscribers can read the payload data, not even pusher. These channels provide end-to-end authenticity
- Our sensitive data is stored securely, encrypted when added to the database and decrypted only when displayed to the relevant users.

Are sensitive requests sent to the server transmitted securely ?

- Requests are transmitted securely using the SSL/TLS protocol, as explained above.

Does a system administrator have access to the sensible data of some arbitrary user ?

- System administrators cannot retrieve unencrypted data such as passwords. These are indeed hashed and salted in the database. The hash functions only work in one direction. Only account owners who have access to their passwords can log in, and the password they enter will be hashed and compared with the password in the database. If it matches, the user can log in.
- Messages are encrypted in the database with a public and private key. A method in laravel does this for us (Crypt::decryptString). All encrypted values are encrypted using OpenSSL and AES-256-CBC encryption. In addition, all Z-encrypted values are signed with a message authentication code (MAC). The built-in message authentication code will prevent the decryption of any value that has been altered by users

1.2 Do I properly guarantee the integrity of the stored data?

- Yes, see point 4.1.3

1.3 Do I properly ensure non-repudiation?

- Do I use signature, certificates, a proper authority ?
  - The transmission of messages is carried out securely through the exchange of asymmetric keys. It then uses this type of encryption via a certificate. He then creates a public key, contained in a certificate, which is used to encrypt the data. But also another private key held only by the owner of the certificate. This is then used for decryption.
  - There is also a certificate which contains the identity of the site and the public key, this acts as a certification authority, it is a trusted third party. It allows us to verify the identity to ensure that a third party has not distributed false public keys to intercept all communications.

1.4 Do I use a proper and strong authentication scheme?

- The authentication of the site is done in a secure way. When a user enters his information, it is sent to the server in a secure way thanks to the SSL/TLS protocol. The password is robust (special characters, capital letters...), the password is hashed via the BCRYPT function, which generates a random salt and generates a hash by passing through a certain number of times in this function (10 by default, OWASP advises to increase this to 12).

1.5 Do my security features rely on secrecy, beyond credentials?

- Passwords are hashed in the database.
- When a user logs in and enters a password, it is hashed and compared to the one in the database. This makes it possible to keep passwords secret.
- Private channels have been set up with pusher to ensure end-to-end encryption. This means that no one except authorised subscribers can read the payload data, not even pusher.
- TLS/SSL also ensures the encryption of data from client to server as explained above. These channels provide end-to-end authenticity

1.6 Am I vulnerable to injection (URL, SQL, Javascript and dedicated parser injections) ?

- We are protected against XSS injections (or cross-site scripting).
  - Whenever a user provides a form the @CSRF tag is then added to it, which automatically checks that the token in the input matches the token stored in the session. If they match, then we know that the authenticated user is the one making the request

- In vue.js, when making POST and GET requests to axios to retrieve json, these lines have been added to the headers, which do the same thing as the @CSRF tag.
- Laravel provides a robust query generator and an eloquent ORM. And thanks to them most queries are protected by default in Laravel applications. Laravel will translate the code into a prepared statement and execute.
- There are vulnerabilities with this system, for example you should not allow user input to dictate the column names referenced by your queries. This is not our case.
- But also via validation rules, which we don't do either.
- We can prevent mass assignment by adding explicitly the fields that a model contains by using protected properties, "fillable" or "guarded":

1.7 Am I vulnerable to data remanence attacks ?

- Data persistence is the residual representation of digital data that remains even after an attempt is made to delete or erase the data. This persistence may result from the fact that the data is left intact by a nominal file deletion operation, by a reformatting of the storage medium that does not delete the data previously written to the medium, or by the physical properties of the storage medium that allow for the recovery of previously written data. Data persistence may make it possible for sensitive information to be disclosed unintentionally if the storage medium is released into an uncontrolled environment (e.g., if it is thrown away or lost).
- We don't leave any residual data on our site, we don't use temporary variables to transfer data, so yes we are protected.

1.8 Am I vulnerable to replay attacks ?

- A replay attack is a form of network attack in which a transmission is maliciously repeated by an attacker who has intercepted the transmission.
- Nous sommes protégés contre ce type d'attaque car nous protégeons notre site avec un certificat SSL. Le canal SSL lui-même est protégé contre les attaques par rejeu grâce au MAC (Message Authentication Code), calculé à partir du secret du MAC et du numéro de séquence. (Le mécanisme MAC est ce qui garantit l'intégrité de la communication TLS).

1.9 Am I vulnerable to fraudulent request forgery?

- Cross-site request forgeries are a type of malicious exploit whereby unauthorized commands are performed on behalf of an authenticated user.

  - Cross-site request forgeries are a type of malicious exploit whereby unauthorized commands are performed on behalf of an authenticated user.

  - Laravel automatically generates a CSRF token for each user. This token is used to verify that the authenticated person is the person who is actually making requests to the application. Since this token is stored in the user's session and changes with each new session regeneration, a malicious application will not be able to access it.

1.10 Am I monitoring enough user activity so that I can immediately detect malicious intents, or analyse an attack a posteriori ?

- Do I simply reject invalid entries, or do I analyse them ?

  - No, this safety aspect was not developed in time.

1.11 Am I using components with know vulnerabilities ?

- Laravel, version : 9.14.0, no known vulnerabilities.
- PHP, version 8.1.6, no known vulnerabilities.
- Pusher, version 7.0.2, no known vulnerabilities.

1.12 Is my system updated ?

- The system is correctly updated.

1.13 Is my access control broken (cf. OWASP 10)?

- Do I use indirect references to resource or functions ?

  - As the application does not have any particular role such as moderators or administrators. There is no special need to check whether a particular user has permission to do a particular action. All people have exactly the same permissions.

  - However, a system has been implemented that allows users to access only the conversations of friends. Messages are then retrieved between the authenticated user and the friend in question.

1.14 Is my authentication broken (cf. OWASP 10) ?

- A rate limit system has been put in place, so it is only possible to try 3 times to enter a user's password before having to wait 3 minutes.
- The credentials, api... are not visible in the repository code, the .env file has not been published. The database has remained there for simplicity of correction. But it should not normally be there.
- It is impossible to access the site in http, when the user tries to access it, the url is automatically forced to its equivalent in https.

1.15 Are my general security features misconfigured (cf. OWASP 10) ?

All our plugins and dependencies are up to date, we have deactivated the directory listing of our web server, so only the public folder is accessible. (js, css...).

The debug mode has been disabled because it can reveal sensitive server information by outputting all your environment variables. Make use of the debug hide app configuration option in Laravel to prevent this.

Conclusion

texte de la conclusion