OWASP	Nature de risque	Description	Gravité	Conséquence(s)	Résolution(s)
A01	Contrôles d'accès défaillants	Pour respecter le principe de confidentialité, nous devons faire en sorte que les utilisateurs (mal-intentionnés ou non) aient uniquement accès aux ressources auxquelles ils doivent avoir accès. Dans notre projet l'administateur est le seul à pouvoir accèder au backoffice depuis le site vitrine. Un contrôle d'accès doit donc être mis en place sur l'ensemble des ressources à accès restreint de notre application, et ce contrôle d'accès ne doit pas pouvoir être contourné. Nous devons impérativement ajouter un contrôle d'accès (côté serveur !) sur les ressources qui n'ont pas vocation à être publiques.	**Très élevé**	Un hacker peut accéder au back office et empêcher d'autres administrateurs d'y accéder. Le hacker peut modifier le contenu du site et accéder à des données sensibles tels que les réponses au questionnaire.	- contrôle des droits d'accès aux fonctionnalités
A02	Défaillances cryptographiques	Toujours dans le but de respecter le principe de confidentialité, nous devons nous assurer que les données transitant avec notre application et stockées dans sa base de données ne puissent pas être divulguées. Les échanges entre clients et serveurs doivent être chiffrés. Certaines données sensibles (exemple : les mots de passe des utilisateurs pour l'accès au backoffice) doivent être stockées en base de données. Au cas où cette base de données serait compromise, ces données sensibles doivent être hachées.	**Très élevé**	Un hacker peut accéder au back office et empêcher d'autres administrateurs d'y accéder. Le hackeur peut modifier le contenu du site et accéder à des données sensibles tels que les réponses au questionnaire.	- identifiéer les données sensibles, et s'assurer qu'elles sont cryptées avec l'algorithme adéquat, lors de la transmission et lors du stockage. - effcetuer des tests
A03	Injection	L'injection consiste à exploiter l'absence de validation/nettoyage des données envoyées par un utilisateur via un champ texte (formulaire), ou directement dans l'URL ! Il en existe plusieurs types, mais les plus courantes sont l'injection SQL et le Cross Site Scripting (XSS). L'injection SQL consiste à saisir une requête SQL dans un champ texte à la place des données attendues. Le Cross Site Scripting (XSS) consiste à envoyer du code javascript dans un champ texte non protégé.	**Très élevé**	Si l'application n'est pas correctement protégée, la requête SQL sera exécutée et pourra entrainer des fuites ou des pertes de données. Il existe deux types d'injections SQL : les injections aveugles et les injections avec retour d'information. Si l'application est vulnérable à la faille XSS, dans le meilleur des cas, le code javascript sera interprété une seule fois, pour l'utilisateur actuel (le hacker, donc). Mais dans le pire des cas, le code javascript sera stocké en base de données et exécuté pour tous les visiteurs du site !	- gestion des privilèges des utilisateurs - utilisation de requêtes préparées - implémenter des jetons CSRF- - utiliser l'attribut SameSite sur les cookies
A05	Mauvaise configuration de sécurité	Configuration des politiques CORS (côté serveur), informations potentiellement divulguées par les messages d'erreur ne doivent pas être visibles des visiteurs	**Très élevé**	Des erreurs de partage de ressources entre origines se produisent lorsqu'un serveur ne renvoie pas les en-t^tes HTTP requis par la norme CORS. Il faut configurer l'API pour qu'elle puisse répondre à la norme CORS.	- la configuration des politiques CORS doit être effectuée dans notre code coté serveur (ex: Nelmio cors) - configuration des pages erreurs (404, 500...)
A06	Composants vulnérables/obsolètes	Il faut faire attention à ne pas utiliser de composants (logiciels, bibliothèques, frameworks, etc.) vulnérables (pour lesquels des failles sont connues, exemple : Log4j) ou obsolètes (exemple : vieilles versions de PHP, Symfony, etc.).	**élevé**	Cette vulnérabilité permet l'éxécution de code à distance.	- cf à la doc
A07	Authentification de mauvaise qualité	Il est impératif de protéger nos applications des attaques bruteforce, et si possible l'idéal est de mettre en place l'authentification par facteurs multiples. Cette méthode de piratage cryptographique consiste à deviner les combinaisons possibles d'un mot de passe ciblé jusqu'à ce que le mot de passe correct soit découvert.	**Très élevé**	Piratage informatique.	- hashage du mot de passe - politique de mot de passe strictes - authentification multi-facteurs - verrouillage des tentatives de connexion - captcha - supprimer les comptes inutilsés
A09	Carence de contrôle & de journalisation	Il faut mettre en place des mécanismes de journalisation sur nos applications : enregistrer toutes les tentatives de connexions (réussies ou échouées), toutes les actions "critiques" (ajout/modification d'un compte utilisateur par exemple).	**élevé**	Les journaux ne doivent pas être stockés en local, afin d'éviter qu'ils soient effacés/altérés. Certaines actions doivent déclencher des alertes, afin de pouvoir réagir au plus vite en cas d'attaque.	Avec le framework symfony, on peut récupérer le journal dans le dossier var/log
	Risque juridique	Les données recueillies doivent être conformes aux RGPD.	**peu**	Risque de sanction pécuniaires.	- informer l'utilisateur de la durée de la sauvegarde de ses données