Stratégie de sécurité

Table des matières

1. Définition

Origin

L'origin défini le nom de domaine de la page sur laquelle on se trouve

Exemple :

- https://www.google.com (origin = google.com)
- https://www.google.com/search?q=hello (origin = google.com)

- SOP (Same Origin Policy)

Le SOP (Same Origin Policy) est une politique de même Origin, c'est une fonction par défaut du navigateur permettant de limiter l'accès au resources de l'Origin. Le SOP permet d'envoyer des requêtes HTTP mais ne permet pas de recevoir de réponse si celle-ci ne provient pas de la même Origin. Si le SOP n'est pas contourné, aucune origine ne peut communiquer avec celle-ci.

- CORS (Cross-Origin Resource Sharing)

Les CORS (Cross-Origin Resource Sharing) partage de ressources entre différentes origin, est une fonctionnalité qui permet de contourner le SOP, celui-ci va permettre la communication entre les différentes origin.

- CSP (Content Security Policy)

Les CSP (Content Security Policy) est une fonctionnalité qui permet de définir des règles des ressources executable par un navigateur, ce qui ne se trouve pas dans cette liste ne pourra être exécuté.

- SRI (Subresource Integrity)

Le SRI (Subresource Integrity) est une fonctionnalité qui permet de vérifier l'intégrité des ressources chargées par le navigateur via un hash de la ressource en question.

- CSRF (Cross-Site Request Forgery)

Le CSRF (Cross-Site Request Forgery) est une classe d'attaque permettant d'injecter du code à l'insu de l'utilisateur authentifié. Ces attaques visent à usurper l'identité de l'utilisateur et à exécuter des actions à son nom.

- XSS (Cross-Site Scripting)

Le XSS (Cross-Site Scripting) est une attaque permettant d'injecter du code à l'insu de l'utilisateur. Ces attaques visent à prendre le contrôle de l'application, ou d'exécuter des actions à l'insu de l'utilisateur.

- SQLI (SQL Injection)

Le SQLI (SQL Injection) est une attaque permettant d'injecter du code SQL lors de l'exécution d'une requête SQL.

2. Stratégie de sécurité

Les protocoles de protection de l'échange de données

L'application que nous allons concevoir va être sur plusieurs navigateurs. Pour commencer, nous allons évoquer le système **HTTP (Hyper Text Transfer Protocol)**, qui est un protocole de transfert d'information permettant d'échanger entre le client et le serveur.

Le problème est que *HTTP*, n'étant pas encore sécurisé, est susceptible de se faire attaquer par le "man-in-the-middle". Cette troisième personne est un hacker, pouvant intercepter le contenu d'un message ou d'une information, le lire et/ou le modifier avant de le renvoyer au vrai destinataire.

Pour sécuriser la liaison, nous allons mettre en place le **TLS (Transport Layer Security)** permettant de chiffrer le contenu devenant ainsi difficile à lire. Alors, *HTTP* avec la sécurisation *TLS* devient *HTTPS*.

HSTS (HTTP Strict Transport Security) permet de verrouiller la sécurité en redirigeant le site HTTP vers un HTTPS. Cependant, cela ne marche pas lors de la première visitation sur un site.

Hachage, salage

Pour plus de sécurité, nous proposons, lorsque nous devons manipuler des données sensibles tels que des mots de passe, préférer ne pas les utiliser en "clair" mais les chiffrer avec une fonction de hachage. Le hachage fonctionne notamment avec un algorithme nommé *SHA-256*, permettant d'obtenir une empreinte unique du fichier chiffré. Ce dernier respecte la *RGPD* et il est sécurisant. Il

est ainsi impossible de revenir en arrière pour avoir le message initial sauf si on a retrouvé la clé correspondante.

Pour palier à ce problème, nous rajouterons une sécurisation supplémentaire au hachage appelé salage. Le salage consiste à ajouter "du sel" (un mot) connu seulement de l'utilisateur, et de l'ajouter à l'empreinte du hachage. L'ensemble doit être mixé à nouveau avec le hachage afin d'obtenir un message chiffré différent. Cette fois-ci, il sera plus difficile de le déchiffrer car il faut connaître "le sel".

Protection du navigateur

Notre application va être utilisée sur internet, c'est pourquoi nous devons sécuriser les navigateurs.

La première protection du navigateur, par défaut est la **SOP (Same-Origin Policy)**. La sécurité de la *SOP* permet à deux sites différents de ne pas s'échanger des informations.

La deuxième protection est le **CORS(Cross-origin resource sharing)**. Ce dernier permet de contourner la *SOP* et d'autoriser certains sites différents à échanger des informations.

La troisième proctection est le **CSP(Constent Secure Policy)** sécurise davantage après avoir autorisé la *CORS*. Le *CSP* fonctionne avec une 'liste blanche', autorisant seulement ce qu'il y a sur cette liste et interdisant le reste.

La dernière est le **SRI(Subresource Integrity)** permet d'analyser si l'intégrité du contenu d'une sous-ressource extérieur au site n'a pas été modifié et ne contient pas de faille. Pour vérifier que le *SRI* n'a pas été modifié, le navigateur doit constater qu'il n'y a pas eu manipulation. Cela fonctionne avec un hachage qui doit correspondre avec le fichier récupéré.

Politique des mots de passes

Notre politique des mots de passe fonctionne avec certains paramètres :

- Le mot de passe doit être chiffré dans la base de données.
- La longueur du mot de passe doit être long, car plus il sera long plus cela sera difficile à hacker.
- La demande d'authentification doit être demandée avant une manipulation sensible
- Un délai d'expiration du mot de passe pour les modérateurs et administrateurs au sein d'un site doit être mis en place pour une durée de quatre mois.
- La méthode de conservation dans un coffre fort permet de garder un mot de passe unique pour l'ouvrir,et contenir une multitude de mots de passe.
- Limitation d'une tentative ratée pour s'authentifier.

La sanitisation

la sanitisation permet de nettoyer les données reçues d'un utilisateur avant de l'envoyer dans la base de données. Toute demande passe par un formulaire. On doit vérifier si le formulaire ne contient rien de dangereux pour notre application : Depuis le front (visibilité sur l'écran): limiter les caractères, obliger à mettre que des numéros ou que des lettres... Depuis le back : vérifier l'existence d'une faille comme SQLI..

Quelle est la différence entre Authentification et Autorisation ?

L'authentification est la vérification de l'identité de la personne souhaitant entrer dans l'application. Après la vérification, si la personne est autorisé à entrer, elle pourra utiliser les ressources disponible pour lui.

L'accès aux données

Pour sécuriser le réseau de l'application, nous allons mettre en place la règle du moindre privilège en appliquant le R-BAC (Role Based Access Control). Cela signifie que chaque personne aura accès à certaines ressources en fonction du rôle occupé au sein de l'entreprise. Il existe différents rôles :

- Administrateur
- Modérateur
- Rédacteur
- Utilisateur

Token

Nous proposons des jetons d'authentification à double facteur qui mettent en place une sécurisation, permettant lors de l'ouverture d'une session, d'obtenir un jeton confirmant ainsi son identité. Cela limitera les risques d'usurpation.

Afin de garder la session de l'utilisateur sécurisé, nous allons prendre des dispositions avec le jeton :

- la durée du jeton délivré ne doit pas excéder quelques minutes lors de la création de sa session en validant par mail ou par sms.
- Les jetons ne doivent pas permettre à l'utilisateur de s'authentifier automatiquement afin d'accéder à sa session.

La session

Une fois authentifié avec succès, l'utilisateur aura accès à sa session suivant la règle du R-BAC, à savoir le moindre privilège. Pour sécuriser une session ouverte, nous proposons de limiter sa durée à une journée. De plus, un cookie HTTP (cookie de navigateur) contient des données du serveur et les envoie vers le navigateur de l'utilisateur. Les cookies conservent des informations sur l'utilisateur (personnalisation, suivi..). Nous exclurons les données sensibles de l'utilisateur dans le cookie afin d'éviter lors d'une attaque, que l'attaquant usurpe l'identité de l'utilisateur.

La sécurisation de l'API

L'API (Application Programming Interface) permet d'accéder à la base de données de notre application, montrant les données, voire les données sensibles. Afin d'éviter les attaques de type SQLI, WSS ou MITM, nous devons sécuriser notre API. Pour la sécuriser, nous avons mis en place HTTPS et les jetons d'identification. De plus, nous recommandons de mettre en place :

- Limiter et anticiper les requêtes lors d'une attaque DDOS.
- Nous conseillons l'utilisation d'une API Stateless permettant de récupérer des informations précises. De plus, il ne garde aucune trace de ces échanges précédents, Permettant ainsi de sécuriser davantage.

journalisation

La Journalisation est le traçage continue des actions effectué sur l'application et sur les différents serveurs. Cela permet une traçabilité des actions si un problème survient.

Dernière étapes de l'application

Lorsque notre application sera sortie, nous proposerons de réaliser un audit PASSI afin de tester la sécurité de notre application. Nous pourons éventuellement faire un bug bounty, faisant appel à des

hackers professionnels dont le métier est de tester la sécurité de notre application contre une récompense.