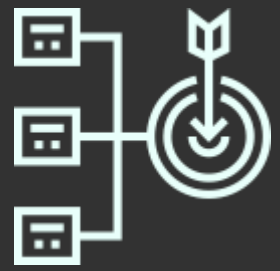
SÉCURITÉ
NOTIONS DE BASE

Jérôme GRASSET – grasset.prepa@gmail.com

OBJECTIFS DU COURS

- Vous sensibiliser à l'idée de réfléchir à la sécurité de vos applications web
- Présenter quelques problèmes courants, et des remèdes
- Vous donner quelques éléments de culture générale informatique



Ce qui suit n'est pas un « cours de sécurité »

En conséquence, pour les exemples d'exploitation de failles :

- le code est écrit pour créer la faille (en particulier pour les injections)
- les contremesures sont simples

SÉCURITÉ : DE QUOI ?

- Des données, dans la ou les bases de données :
  - Données personnelles : RGPD
  - Données « savoir » de l'entreprise
  - Informations confidentielles (compta par exemple)
  - …

  Sécuriser : éviter les fuites, les altérations, les suppressions…
- Du « site web » : éviter que son contenu puisse être détourné, modifié
- Du serveur web, au sens machine qui héberge le site : éviter que des intrusions se fassent via le site web (et puissent se propager sur le réseau)
- Des clients web : éviter que du contenu malveillant soit exécuté sur un poste client suite à une visite sur le site web.

PRINCIPAUX ELÉMENTS MIS EN PLACE

- Authentification : « qui est le visiteur ? », mais aussi « qui est le site web ? »
- Contrôle d'accès : zones réservées aux différentes catégories d'utilisateurs
- Protection des (bases de) données
- Utilisation d'un canal de communication protégé entre l'utilisateur et le site

Problème : comment limiter les risques de défaillance sur ces points ?

CANAL DE COMMUNICATION : HTTPS

- Mise en œuvre de HTTPS : un « réglage » du serveur, pas d'impact sur le développement du site
- Principe d'une communication HTTPS :
  - le client demande au serveur l'établissement d'une connexion sécurisée, en donnant les méthodes de chiffrement qu'il supporte
  - le serveur répond en indiquant
    - la méthode de chiffrement qu'il choisit
    - son certificat, qui garantit son authenticité
    - sa clé publique
  - le navigateur crée une clé de chiffrement symétrique, il la transmet au serveur en la chiffrant de façon asymétrique avec la clé publique du serveur
  - le serveur confirme la réception, la communication peut s'établir avec le chiffrement symétrique
- *Une parenthèse : préconisation de l'ANSSI pour « l'analyse du traffic HTTPS » :* https://www.ssi.gouv.fr/guide/recommandations-de-securite-concernant-lanalyse-des-flux-https/

AUTHENTIFICATION : MOTS DE PASSE

- Choix (ou génération) du mot de passe : doit être « sûr » (pas 123456…)
- Possibilité de modification : pas nécessairement laissée à l'utilisateur
- Si possible s'assurer que la communication du mot de passe est sûre (par exemple envoi par un autre canal (sms, courrier papier…))
- Stockage du mot de passe, côté serveur :
  - correctement chiffré dans la base de données
  - ne se promène pas dans l'application (pas placé en session par exemple)
- Utiliser l'authentification par tiers de confiance peut être envisagé :
  - Google, Facebook, …
  - Mécanismes « privés » : par exemple serveur CAS dans une université
- Double authentification

AUTHENTIFICATION : MOTS DE PASSE

- Utilisation du mot de passe :
  - ne pas le faire circuler client/serveur de façon trop visible : pour nous utiliser post, pas get
  - si (vraiment) besoin le « dissimuler » : diverses possibilités, comme des claviers « virtuels »
- Rappels :
  - la communication est chiffrée (HTTPS) entre le client et le serveur, mais les messages sont en clair sur le client et sur le serveur
  - POST ne sécurise pas : on peut voir les données envoyées dans la requête HTTP, par exemple avec les outils de dev des navigateurs
  - mais en POST, contrairement à GET :
    - les données ne se voient pas dans l'URL
    - elles ne sont pas (toujours…) stockées dans les caches et journaux qui conservent les URL

CONTRÔLES D'ACCÈS

- Contrôle d'accès : vérification que le visiteur a bien le droit d'accéder à la ressource qu'il demande
- A mettre en place sur toutes les URL « exécutables » (php) :
  - Pages web
  - Scripts qui répondent à des requêtes asynchrones
  - Intérêt de passer par un routeur/dispatcher pour limiter les scripts exécutables

  On peut facilement voir le traffic HTTP, on a donc accès à beaucoup d'URL autres que les URL des pages.

- Pour nous : contrôle à base de sessions php, éventuellement avec ajout de token (cf CSRF plus loin)

CORS : CROSS-ORIGINE RESOURCE SHARING

- Cross-Origine Resource Sharing : accéder à des ressources ayant des origines multiples.
- Origines différentes :
  - la page reçue par le client contient des demandes de ressources situées à une autre origine
  - origine : domaine, protocole, port
- Autoriser à ce qu'une page demande des ressources hors du site qui l'a « servie » :
  Exemple : Bootstrap ou jQuery fournis par des CDN (Content Distribution Network)
  ⇨ les CDN doivent autoriser une demande venant d'une page fournie par un autre serveur
- Pour les requêtes asynchrones (« AJAX ») émises par un script :
  - origines différentes : interdit par défaut pour récupérer des données (pas pour envoyer des données au serveur)
  - le serveur qui reçoit la demande peut indiquer dans les entêtes les origines extérieures qu'il autorise : configuration Apache ou fonction php header dans chaque fichier (par exemple).

DES MAILLONS FAIBLES…

(On s'intéresse ici aux éléments qui relèvent du développeur, pas directement à l'information et à l'éducation de l'utilisateur ni à l'infrastructure matérielle ou logicielle « fournie ».)

A contrôler :

- Tous les flux reçus côté serveur :
  - Envoi de données via des formulaires
  - Envoi de données en requêtes asynchrones
  - Upload de fichiers
- Les messages d'erreurs non contrôlés qui donnent des informations sur l'application
- Tout autre mode de diffusion de l'information sur la structure interne… par exemple code publié pour les CMS ou frameworks web

EXEMPLES D'EXPLOITATION DE « FAILLES »

- Injection SQL : faire exécuter du code SQL via les données envoyées au serveur
- Injection de code (« XSS ») : faire exécuter du code (JS) via les données envoyées au serveur
- CSRF : Cross-Source Request Forgery :
  - amener l'utilisateur sur une page d'un site « malveillant »
  - adresser une requête HTTP via formulaire au site ciblé : si la session de l'utilisateur y est ouverte, la requête peut être acceptée

EXEMPLES D'EXPLOITATION DE « FAILLES »

Remarques préalables :

- L'exploitation des failles peut nécessiter une préparation longue, en particulier l'usage de serveurs
- La recherche d'une faille peut sembler longue également…
- … mais l'attaquant peut disposer d'une infrastructure déjà en place et d'outils automatisés
- Il n'est pas nécessaire d'être très compétent pour exploiter des boîtes à outils de hacking : beaucoup de « hackers » potentiels

INJECTION SQL

- Injection SQL : faire exécuter du code SQL via les données envoyées au serveur, par exemple via un formulaire
- Pour éviter : nettoyer les chaines de caractères, rendre inopérants les caractères qui permettent l'injection
- Avec PDO ou mysqli :
  - requêtes préparées, nettoyage automatique
  - sinon (mauvaise idée, cf : https://www.php.net/manual/en/pdo.quote.php) : fonctions de nettoyage, comme PDO::quote(chaine)

Attention : il s'agit d'éviter l'utilisation de code SQL, des fonctions comme htmlspecialchars n'ont pas le même usage.

Exemple et principe en pratique

INJECTION DE CODE (XSS)

- Injection de code (« XSS ») : faire exécuter du code (JS) via les données envoyées au serveur, par exemple un formulaire
- 2 approches :
  - Reflected XSS. Le code est « volatile », par exemple un lien qui contient une URL et le code injecté : la victime clique sur le lien et déclenche elle-même l'injection.
  - Stored XSS. Le code est stocké sur le site, par exemple en base de données : les visiteurs sont victimes lorsqu'ils accèdent aux pages « infectées »

  (une doc (site intéressant) : https://owasp.org/www-community/Types_of_Cross-Site_Scripting)
- Ce que peut faire le code javascript exécuté :
  - Redirection vers une page « pirate »
  - Création de faux formulaires, de connexion par exemple
  - Transmission d'informations vers un site pirate (par exemple « vol de cookies », comme un cookie de session)
  - …

INJECTION DE CODE (XSS)

Injection de code (« XSS ») suite.

- Pour éviter : nettoyer les chaînes de caractères, pb similaire à une injection SQL, protections également.
- En php, a minima, filtrer les chaînes de caractères des données entrantes :
  - htmlspecialchars() : réécriture 'sûre' des < et >
  - ou htmlentities() : réécriture 'sûre' d'autres caractères (comme les guillemets)
- Protéger les cookies de session (httpOnly en particulier : ne sont plus accessibles par document.cookie, voir par ex https://owasp.org/www-community/HttpOnly)

CSRF - CROSS-SOURCE REQUEST FORGERY

- CSRF :
  - amener l'utilisateur sur une page d'un site « malveillant »
  - adresser une requête HTTP via formulaire au site ciblé, pour une action nécessitant une authentification : si la session de l'utilisateur y est ouverte, la requête peut être acceptée
- Pour éviter, utilisation de tokens (anti-)CSRF (token = jeton) :
  - le serveur génère un token qu'il stocke en session
  - Chaque formulaire ou élément sensible du site contient ce token (par exemple avec un « input hidden » dans les formulaires)
  - toute demande du client doit contenir ce token : le formulaire « pirate » ne le connaît pas
  - Ce token a une durée de vie courte (communément 10 minutes)

CSP : CONTENT-SECURITY-POLICY

- Des directives de sécurité que le serveur adresse aux navigateurs clients
- Permet d'indiquer au navigateur ce qu'il doit et ne doit pas accepter

  (Attention : suppose un navigateur compatible)
- Les règles sont transmises dans les entêtes

- Pour en savoir plus : https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

EN GUISE DE CONCLUSION

- Soyez conscients des risques
- Postulat : vous n'êtes jamais en sécurité totale
- Identifiez les données et pages sensibles
- Lisez les préconisations d'utilisation de vos librairies, frameworks et autres (bonnes pratiques, avertissement de sécurité)
- Limitez les accès autant que vous le pouvez
- N'hésitez pas à déléguer à des personnes ou entités plus compétentes la sécurisation de certaines parties de votre travail
- Faites des sauvegardes… et préparez la reprise après panne…