

LA SECURITE

1

Table des matières

1. LES PROBLEMATIQUES
2. LES SOLUTIONS
3. LES REGLES DE LA SECURITE
4. LA SECURITE INTERNE
5. SECURISER LE CODE
6. SECURISER LES BASES DE DONNEES ET LES FICHIERS
7. LA VEILLE : LES SITES A CONSULTER
8. LES NEWSLETTER

Table des matières

- 9. LES VERSIONS (OS, LANGAGES, DEPENDANCES)
- 10. OWASP
- 11. HTTP/HTTPS
- 12. LES MOTS DE PASSE
- 13. LE SEL
- 14. LE POIVRE
- 15. LES INJECTIONS DE CODE
- 16. XSS (Cross-site Scripting)
- 17. XSRF ou CSRF (Cross-site Request Forgery)

1. LES PROBLEMATIQUES

- Actuellement, la plus grande faille de sécurité est le vecteur humain
 - ✓ Les attaques internes sont devenues fréquentes
 - ✓ Ces attaques sont plus difficiles à repérer
- L'utilisation du Cloud rend la détection des attaques encore plus difficile
- Les sites web sont très exposés aux attaques (code et API)
- Il n'y a pas de « 100% de sécurité »

2. LES SOLUTIONS

- Mettre en place une sécurité interne (PSSI*)
- Sécuriser le code
- Sécuriser les bases de données et les fichiers
- Faire une veille de sécurité, notamment pour sa technologie
- Eviter les failles de sécurité en mettant à jour les OS, langages, dépendances
- Empêcher un hacker d'avoir la main sur un ordinateur (failles XSS, CSRF)
- Empêcher un hacker de s'authentifier ou de récupérer des droits, des privilèges

* Politique de Sécurité du Système d'Information

3. LES REGLES DE LA SECURITE

- Rappel des critères de sécurité des SI
 - Disponibilité : accessibilité au moment voulu
 - Intégrité : les données sont bien celles attendues (exactitude et complétude)
 - Confidentialité : accès des personnes aux seules informations autorisées
 - Preuve : traçabilité, imputabilité

4. LA SECURITE INTERNE

- Mettre en place des Authentifications fortes (double authentification) et supprimer les accès des personnes partant de l'entreprise
- Surveiller les bases de données
- Contrôler au maximum le Cloud, les appareils nomades
- Bloquer les ports non nécessaires
- Utiliser un Firewall (réseau public/réseaux privés)
- Utiliser un Serveur Proxy (interdire certains sites, journaliser les accès)
- Gérer des Wi-Fi, avoir un Wi-Fi particulier pour les invités
- Utiliser un VPN (réseau privé virtuel) pour les accès à distance

5. SECURISER LE CODE

- Développer un code clair, documenté et commenté : principe KISS : Keep it Simple, Stupid ou « il n'y a pas de sécurité dans l'obscurité »
- Si le code est difficile à maintenir => ajout possible de failles
- Ne développer que les fonctionnalités indispensables
- Une classe => une fonctionnalité. Un faible couplage limite la portée des attaques
- Appliquer les principes SOLID
([https://fr.wikipedia.org/wiki/SOLID_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique)))
- Sécuriser toutes les entrées au niveau du front et du serveur
(**l'utilisateur doit toujours être considéré comme malveillant**)
- Gérer la mémoire et les accès aux bases de données

5. SECURISER LE CODE

- Traiter les exceptions et leur code retour pour les BDD
- Journaliser les traitements (logs) pour garder une trace des attaques
- Utiliser les outils de sécurité existants et testés
- Ne donner à l'utilisateur que le minimum de renseignements
- Sécuriser les données au sein de transactions
- [anssi-guide-recommandations mise en oeuvre site web maitriser s tandards securite cote navigateur-v2.0 \(gouv.fr\)](#)

OIV (opérateur d'importance vitale)

6. SECURISER LES BASES DE DONNEES ET LES FICHIERS

- Sécuriser les fichiers : Par l'intermédiaire de listes de contrôle d'accès (ACL), on leur associe des droits d'accès ou permissions afin de contrôler à un niveau très fin, qui peut lire, écrire ou exécuter un fichier
- Mettre en place le maximum de contraintes au niveau des champs des bases de données (non nuls, uniques, liste ou intervalle de données...)
- Traiter les injections SQL

6. SECURISER LES BASES DE DONNEES ET LES FICHIERS

- Mettre en place des triggers si nécessaire
- Vérifier toutes les données coté serveur avant leur traitement dans la base de données.
- **Ne donner aux applications et personnes que les droits dont ils ont besoin**
- Créer des utilisateurs pour les applications (users)

7. LA VEILLE : LES SITES A CONSULTER

✓ La cybersécurité

- ➡ <https://www.cert.ssi.gouv.fr/> => abonnement au flux RSS
- ➡ <https://www.ssi.gouv.fr/> (ANSSI)
- ➡ Pharos : signaler un contenu suspect
- ➡ [Have I Been Pwned: Check if your email has been compromised in a data breach](#) : Ai-je été compromis ?

✓ La protection des données

- ➡ <https://www.cnil.fr/professionnel>

7. LA VEILLE : LES SITES A CONSULTER

- ✓ La veille technologique et sécurité
 - <https://news.ycombinator.com/> (hacker news) (libre) => soumis au vote
 - <https://lobste.rs/> (invité par un membre pour contribuer) => dans l'ordre d'arrivée
 - <https://hackernoon.com/>
 - <https://www.freecodecamp.org/news/> (tuto + cours)
 - <https://dev.to/>
 - <https://www.journalduhacker.net/>
 - <https://news.humancoders.com/>
 - <https://www.nextinpact.com/>

8. LES NEWSLETTER

- Ex : hackernews, lobste.rs, journalduhacker : tous les week-end, résumé de ce qui s'est passé la semaine
- Ils ont des flux RSS aussi
- Par exemple pour les CMS (comme Drupal), chercher si ils ont une newsletter

9. LES VERSIONS (OS, LANGAGES, DEPENDANCES)

- ✓ Convention de nommage (SEMVER : Semantic Versioning)
 - Rappel : version majeure, version mineure, patch, Modificateur (RC Release Candidate :2.6.5-rc => patch 5 presque prêt)
 - Prendre en priorité les LTS (Long-Term Support)
 - Si version majeure, pas forcément retro-compatible avec la version antérieure

9. LES VERSIONS (OS, LANGAGES, DEPENDANCES)

- ✓ Github : dependabot (service gratuit)
 - Robot qui scanne les dépendances et prévient quand une d'entre elles est obsolète ou avec des failles de sécurité
 - S'inscrire et lui préciser les dépôts
 - Pull request ou prévenir ou faire lui-même (push) comme les failles de sécurité critiques
- ✓ WhiteSource Renovate :
<https://www.whitesourcesoftware.com/free-developer-tools/renovate/>
- ✓ Owasp dependency check

9. LES VERSIONS (OS, LANGAGES, DEPENDANCES)

- ✓ Npm audit
 - Peut s'automatiser avec une Intégration Continue
- ✓ Maven upgrade
- ✓ Bien retester à chaque fois

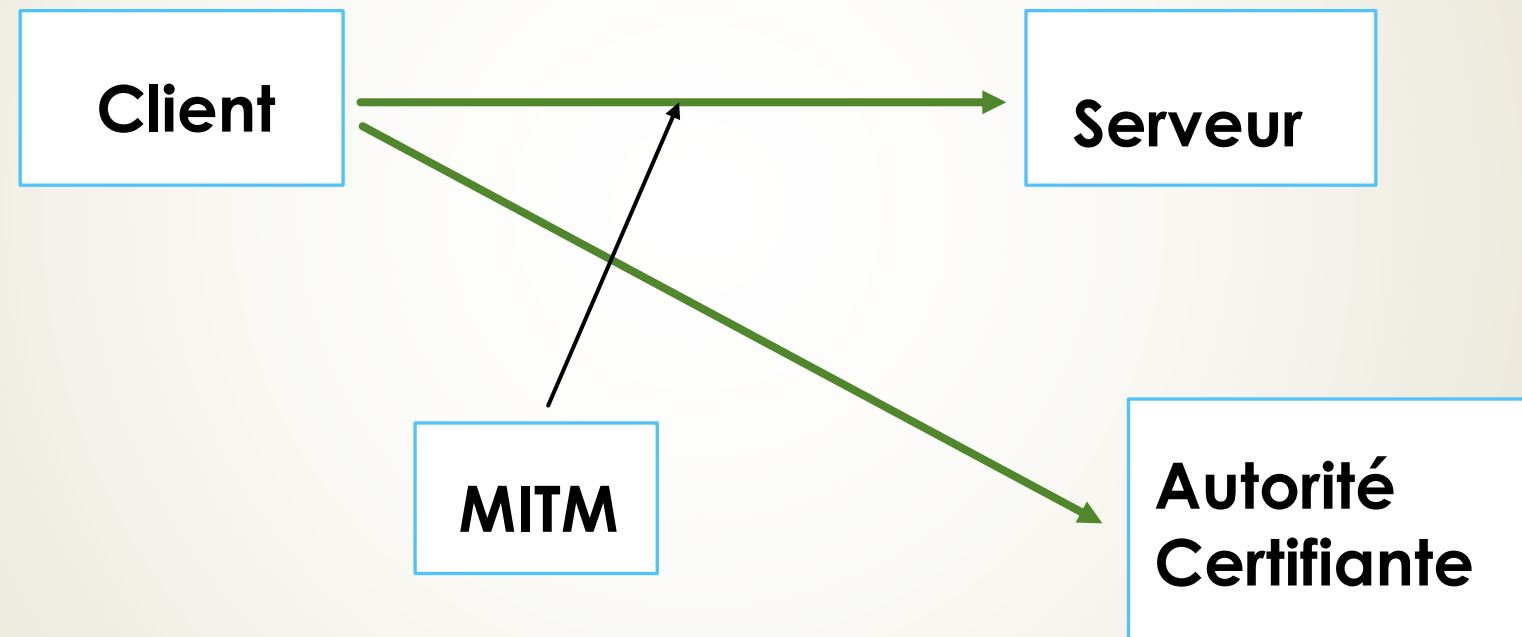
10. OWASP

- ✓ <https://owasp.org/>
- ✓ Fondation à but non lucratif, open source, ouverte
- ✓ Autorité en matière de sécurité du Web
- ✓ Propose des Bibliothèques Java pour les injections de code par exemple : OWASP Java HTML Sanitizer
- ✓ Owasp SAMM (Software Assurance Maturity Model) = CMMI
- ✓ Owasp ZAP (Zed Attack Proxy)
- ✓ <https://owasp.org/www-project-top-ten/>
- ✓ <https://owasp.org/Top10/fr/>

11. HTTP/HTTPS

- http : en clair sur le net (par défaut le port 80)
- https : http ssl (Secure Sockets Layer) ou tls (Transport Layer Security) (par défaut le port 443)
 - ✓ Chiffrement
 - ✓ Devient la norme
 - ✓ Clé commune entre le client et le serveur qui va changer entre chaque connexion

11. HTTPS



- Ce processus s'appelle le HandShake
- MITM : Man In The Middle

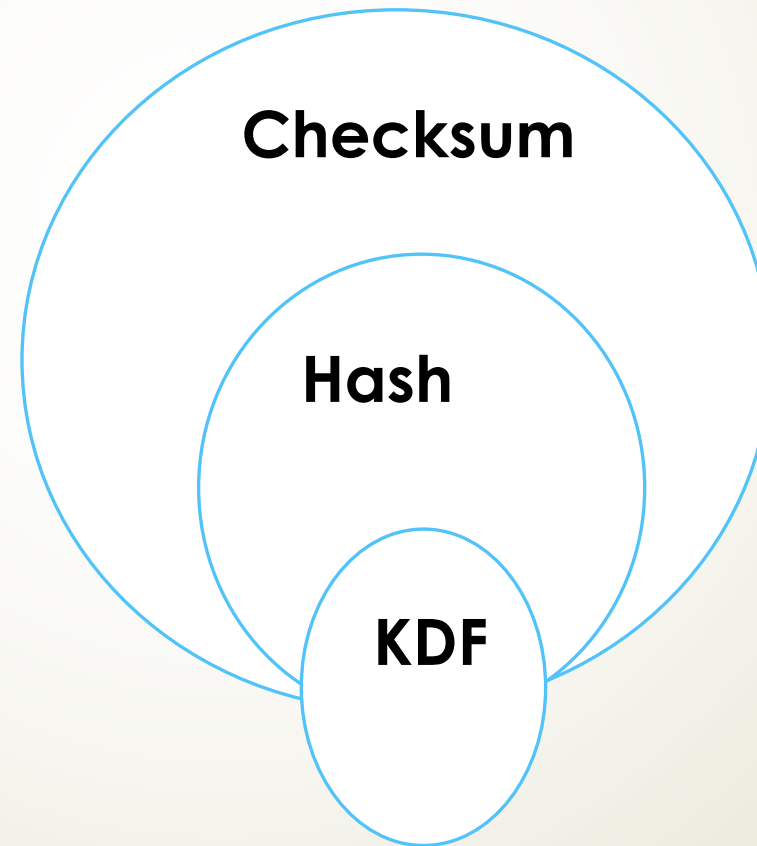
11. HTTPS

- Ssl et tls sont des protocoles d'échange de clés asymétriques
- Une clé privée et une clé publique sont générées
- La clé privée ne sort pas du serveur : elle sert à chiffrer les données envoyées et déchiffrer les données reçues
- La clé publique va déchiffrer et rechiffrer. La clé publique est disponible
- Clé publique : l'autorité certifiante certifie au client que c'est le bon serveur qui a envoyé la clé publique

12. LES MOTS DE PASSE

- **Ne jamais faire soit même ses algorithmes de sécurité**
- Le Mdp doit être crypté/haché coté serveur pour raison de sécurité : il n'y a pas de sécurité coté front. Il faut donc être en https pour que le Mdp soit chiffré sur le net
- Différences entre cryptage/hachage et chiffrement (cipher)
 - ✓ On peut déchiffrer (decipher) si on a la clé de chiffrement mais on ne peut pas décrypter
 - ✓ Crypter les mots de passe

12. LES MOTS DE PASSE



12. LES MOTS DE PASSE

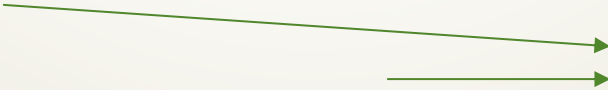
➤ Les Checksum

- Vérifier l'intégrité d'une donnée
 - Additions ou opérateurs logiques
 - Sortie toujours de même taille : ex entre 0 et 100
 - Rapide
 - Collisions : même résultat pour 2 entrées différentes car éventail entrées infini mais résultat du checksum fini
- ✓ CRC32

12. LES MOTS DE PASSE

- Les Hash non cryptographiques
 - Transformer les clés en entiers pour le mettre dans un index
 - ✓ En java, on peut mettre un objet en clé avec les Map
 - Taille fixe
 - Minimum de collisions
 - 2 valeurs proches doivent avoir des valeurs très différentes (répartition des clés adjacentes)
- ✓ HMAC
- ✓ MD5 : plus utilisable dans le cadre de la sécurité pour cause de collision. Il existe des algorithmes qui trouvent quoi donner en entrée pour que MD5 pense que c'est ce qu'il veut. Mais il est utilisé pour des vérif car très rapide

12. LES MOTS DE PASSE

- KDF (Key Derivation Functions) Ce sont des Hash cryptographiques
 - Checksums qui respectent les conditions des Hash et qui respectent en plus
 - ✓ Lent et couteux (mémoire) => sinon, on pourra trouver une collision plus facilement
 - Pour une authentification, on ne fait ça qu'une fois
 - ✓ RSA
 - ✓ SHA-x
 - ✓ BCRIPT
 - ✓ ARGON2 (dont variants)
- Utilisent RSA
- 

12. LES MOTS DE PASSE

- Passage d'un algo à un autre : il faut un champ supplémentaire (genre boolean) pour recrypter le mdp avec le nouvel algo si l'utilisateur est encore avec l'ancien algo (champ topé si nouvel algo)

12. LES MOTS DE PASSE

- Les algo de cryptage donnent toujours le même résultat avec la même entrée
- Attention aux contraintes pour les mots de passe avec majuscule, caractères spéciaux... ils ne pourront pas les retenir, ça n'a pas d'intérêt et ça permet juste de forcer l'utilisateur à mettre un certain nombre de caractères. Il vaut mieux demander une longueur minimale => ils peuvent utiliser des phrases

12. LES MOTS DE PASSE

- 100 Mdp les plus courants : ex password
- Tableau avec ces Mdp et leur résultat : rainbow table (wiki)
- Brute-force attack : consiste en un attaquant soumettant de nombreux mots de passe dans l'espoir de deviner éventuellement une combinaison
 - ✓ Solution : en cas de 3 mauvais mots de passe, attendre 2 minutes, puis 10 minutes, puis 1 heure, etc... ça empêchera de faire du brute-force
- Les algos sont open source pour que ce soit le plus robuste possible : tout le monde peut le tester et voir les failles éventuelles

13. LE SEL

- Combinaison aléatoire personnelle stockée dans l'enregistrement de l'utilisateur avec un séparateur (\$) dans argon2 et bscript)
- ✓ Cryptage « mdp » + sel
- ✓ Enregistrement sel + \$ + cryptage
- ✓ Bibliothèque cryptographique ex Libsodium

14. LE POIVRE

- Valeur aléatoire, assez longue, pour toute l'application, qui n'apparaît pas dans la base de données. Ce peut être dans une variable d'environnement, dans un fichier, dans une classe
- ✓ Exemple symfony : dans le fichier .env, variable APP_SECRET
- ✓ On ne peut pas le changer sans recrypter tous les mdp avec le nouveau, sinon on ne retrouvera pas les valeurs des mdp.

15. LES INJECTIONS DE CODE

L'injection SQL est la plus commune

- Dans un seul statement, on peut faire plusieurs requêtes en mettant un « ; »
- Exemples d'injection SQL : dans un formulaire pour une insertion d'un user
 - ✓ Gabin ') ; drop database nomBDD ;
 - ✓ Gabin ') ; rollback ; insert into user ...
- Solution : échappement des caractères SQL comme le « ; », le ') » etc ...
 - ✓ avant de rentrer l'info dans la BDD. exemple en java avec JDBC, le preparedStatement va les échapper (\). Utiliser également PDO ou un ORM
 - ✓ A la sortie de la BDD : les < c:out > de la JSTL, twig sous symfony

15. LES INJECTIONS DE CODE

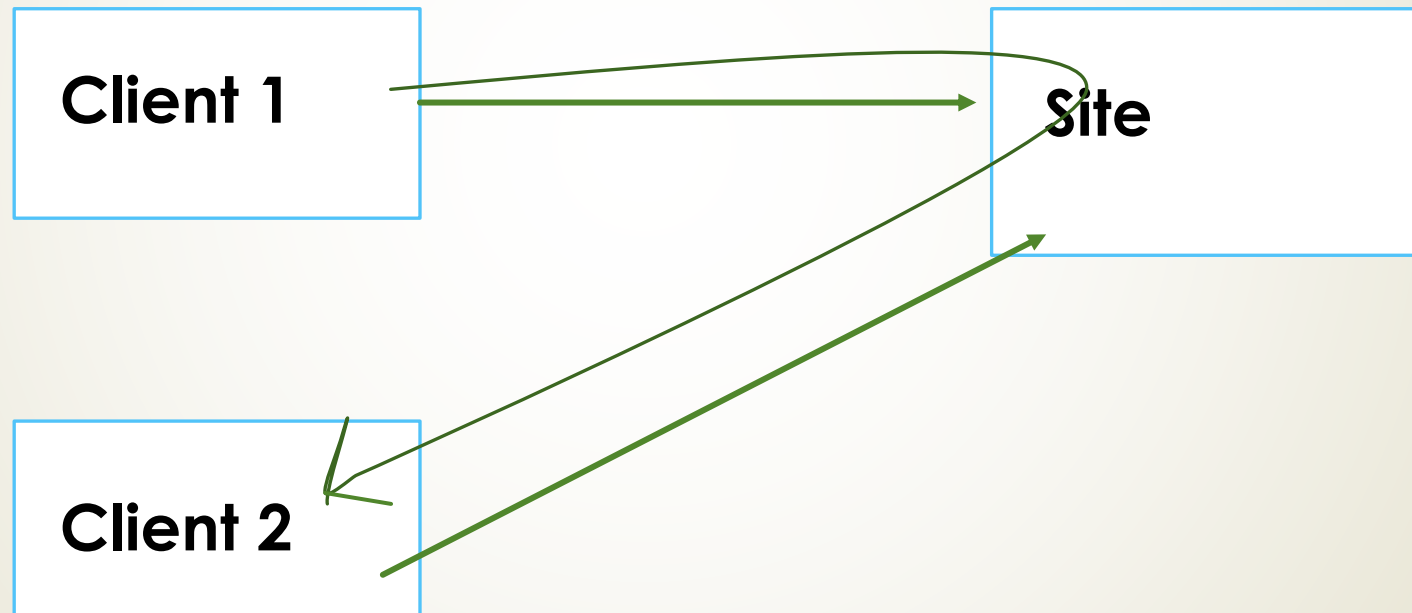
Autres injections de code :

Toute méthode permettant l'insertion de contenu interprétable dans son contexte d'exécution est susceptible de permettre une injection de code. Ainsi, toute insertion non contrôlée de chaîne de caractères dans une page HTML induit une exposition à une vulnérabilité XSS.

(ANSSI : RECOMMANDATIONS POUR LA MISE EN ŒUVRE D'UN SITE WEB : MAÎTRISER LES STANDARDS DE SÉCURITÉ CÔTÉ NAVIGATEUR – pdf page 23)

https://owasp.org/Top10/fr/A03_2021-Injection/

16. XSS (Cross-site Scripting)



16. XSS (Cross-site Scripting)

- Faille XSS : « Scripting en passant par le site »
- Le Client1 peut écrire des scripts qui seront exécutés sur le Client2 en passant par le site quand le client2 visite le site.
- XSS wikipedia => les Risques : redirection, hameçonnage (ou phishing), vole des sessions, rendre le site inaccessible)
- Script : JavaScript ou WASM (WebAssembly) bytecode pour navigateur.
- Le serveur ne sert que de relais.
- Solution : le but est d'intercepter le script quand il arrive sur le serveur et de l'empêcher d'être exécuté sur un autre client

16. XSS (Cross-site Scripting)

- C'est une variante d'une injection de code : pas vers le serveur mais vers un autre client
- Dans un formulaire ou un champ texte : formulaires de contact, blog...
Gabin `<script ... />`
- Exemple dans un blog : commentaires stockés dans la BDD puis affichés et interprétés tels quels dans la page
- Vrai pour tous les langages interprétés (langages de script)
 - ✓ Exemple : `<script>alert('bonjour')</script>` dans un champ texte
 - ✓ Idem avec php, python, etc...
 - ✓ Peut être aussi un script qui écoute ce que tape l'utilisateur
 - ✓ Bitly (wiki) : réduction d'URL : bit.ly/1fmO6Z1+

16. XSS (Cross-site Scripting)

➡ Les solutions

⇒ Remplacer tous les caractères spéciaux ou douteux par l'équivalent HTML : les `htmlspecialchars` (< ; etc...) avant d'entrer la valeur dans la BDD

➤ Dépend de la technologie

✓ Php : `htmlspecialchars()`

✓ Python : `html.encode()`

✓ Java : `StringEscapeUtils` de 'Apache commons Lang'

16. XSS (Cross-site Scripting)

- ⇒ Stocker tel quel dans la BDD et retraiter en HTML
- ✓ `<c : out>` pour la JSP en java
- ✓ Symphony : twig (htmlentities)
- ⇒ Utiliser FTPS (sécurité par ssl ou tls) pour faire le transfert ou la maj d'un site plutôt que FTP, en cas d'écoute active, surtout pour les langages interprétés non compilés

17. XSRF ou CSRF (Cross-site Request Forgery)

- CSRF est une classe d'attaques qui force un utilisateur à exécuter, à son insu, des actions privilégiées sur une application tierce sur laquelle il est authentifié.
- Ce type d'attaques a lieu lors de la navigation sur un site piégé qui émet des requêtes vers un site de confiance, mais vulnérable au CSRF

(ANSSI : RECOMMANDATIONS POUR LA MISE EN ŒUVRE D'UN SITE WEB : MAÎTRISER LES STANDARDS DE SÉCURITÉ CÔTÉ NAVIGATEUR – pdf page 8)

- Ne pas stocker d'informations sensibles dans les cookies
- Dans le cadre de la défense en profondeur et à l'exception des jetons de session, il est recommandé de ne pas stocker des informations sensibles dans les cookies
- Leur utilisation n'est souhaitable que pour le stockage temporaire d'informations de faible volume, pour lesquelles la perte ou la divulgation sera sans conséquence. (page 19)

17. XSRF ou CSRF (Cross-site Request Forgery)

- Exemple : L'authentification par token
 - ✓ Cookie avec un token avec date d'expiration
 - ✓ 2 champs dans la BDD : token + date d'expiration
 - ✓ Filtre qui vérifie si le token est présent dans la BDD et si la date n'est pas dépassée => authentification dans la session
- Si faille XSS, JS sur le navigateur du client2 peut accéder aux cookies, donc le client2 peut se faire passer pour le client1 auprès du site ou un autre site en récupérant le token d'authentification

17. XSRF ou CSRF (Cross-site Request Forgery)

➤ Solution

- Utiliser un token CSRF : champ input hidden qui va être aléatoire et change à chaque page
- Utiliser un JWT (Json Web Token)
- Utiliser le header referer : l'en-tête de requête Referer contient l'adresse de la page web précédente à partir de laquelle un lien a été suivi pour demander la page courante. L'en-tête Referer permet donc aux serveurs d'identifier la provenance des visiteurs d'une page)