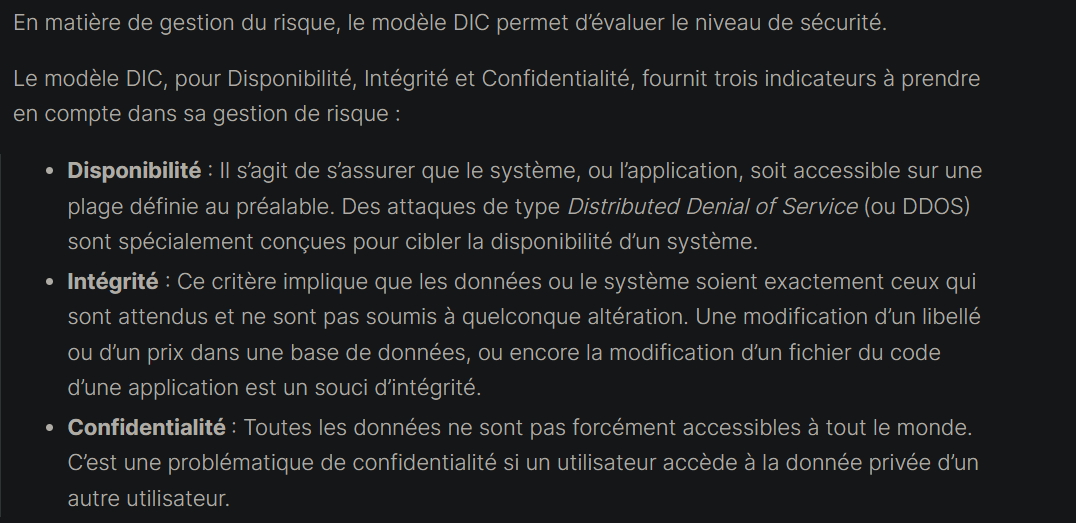
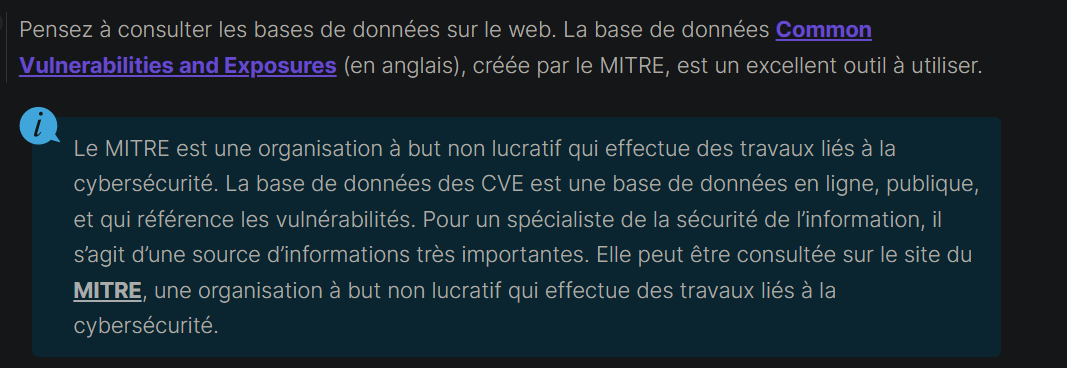
**Open web application security project => OWASP, organisation qui mesure les risques de sécurité dans les applications webs !**





<https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=axios>

<https://www.cve.org/CVERecord?id=REACT>

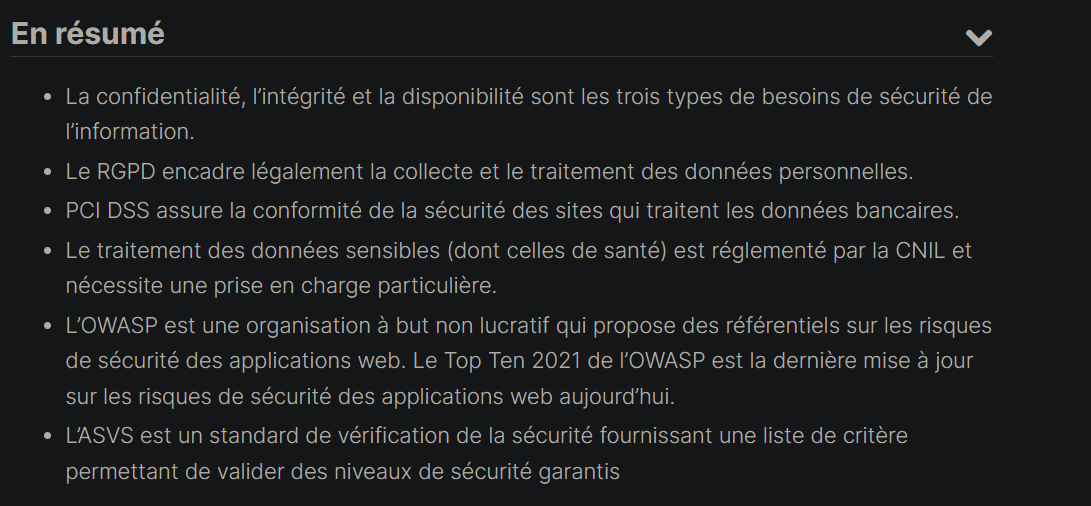
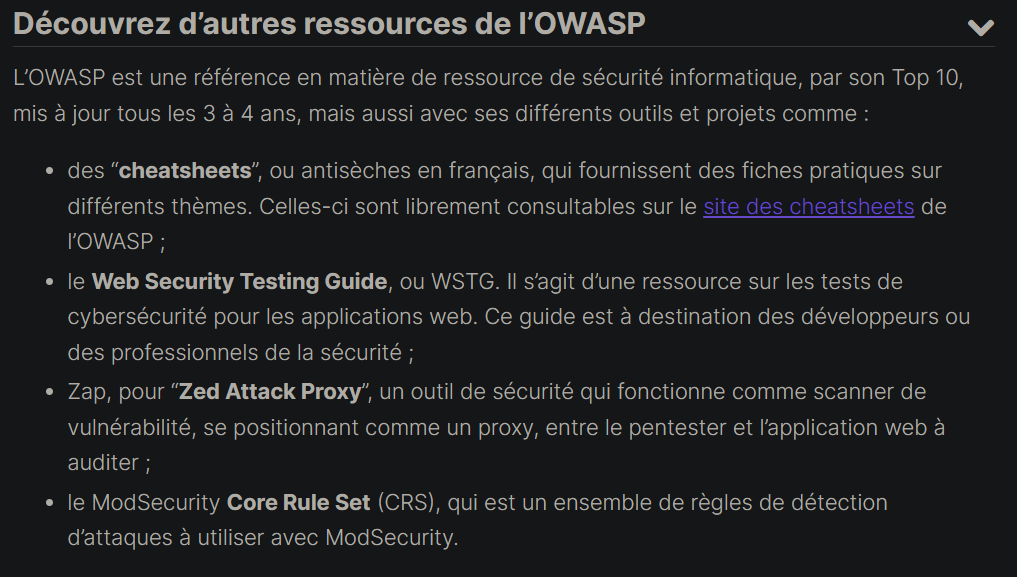
owasp propose plusieurs outils pour répondre aux risques liés aux applications webs !

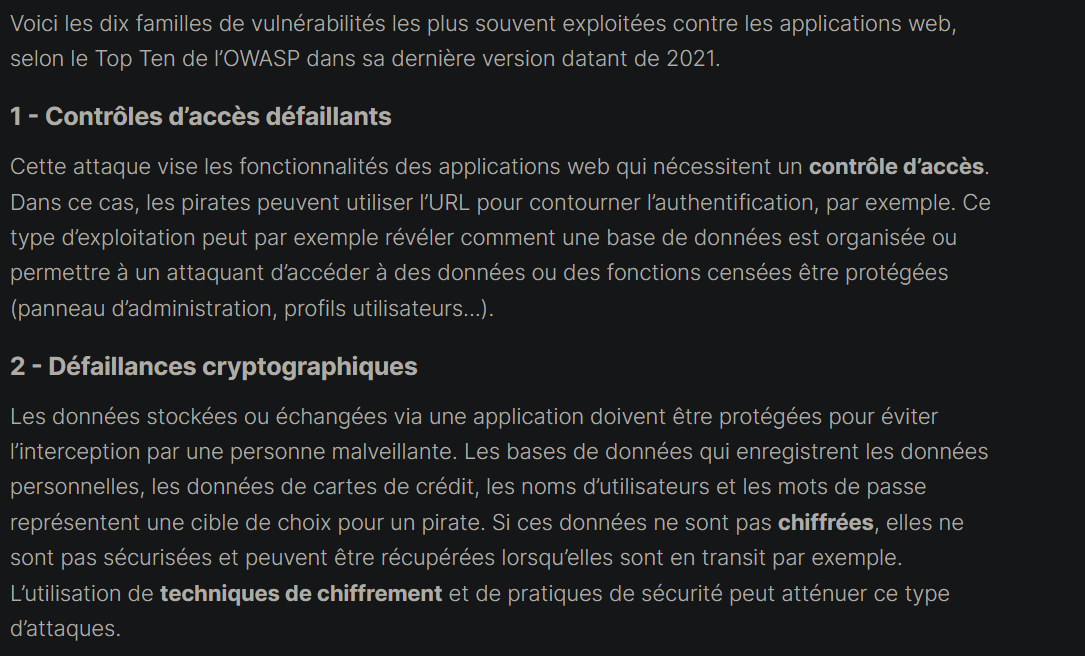
un de ses outils ASVS : application security verification standard, un projet qui fournit une méthode, pour tester la sécurité des applications webs, et une liste d’exigence pour un développement sécurisé qui répond à 3 objectifs ;

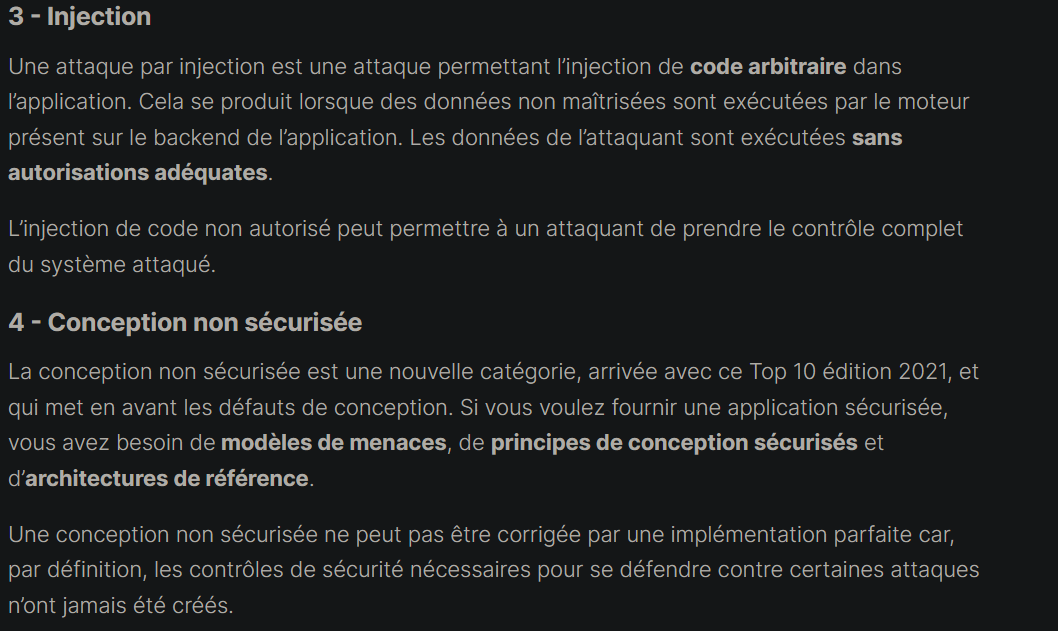
Évaluer le degré de confiance qu’on peut avoir vis-à-vis de l’application web, donner des conseils de bonne pratique de sécurité, spécifier les exigences de vérification de sécurité de l’application !

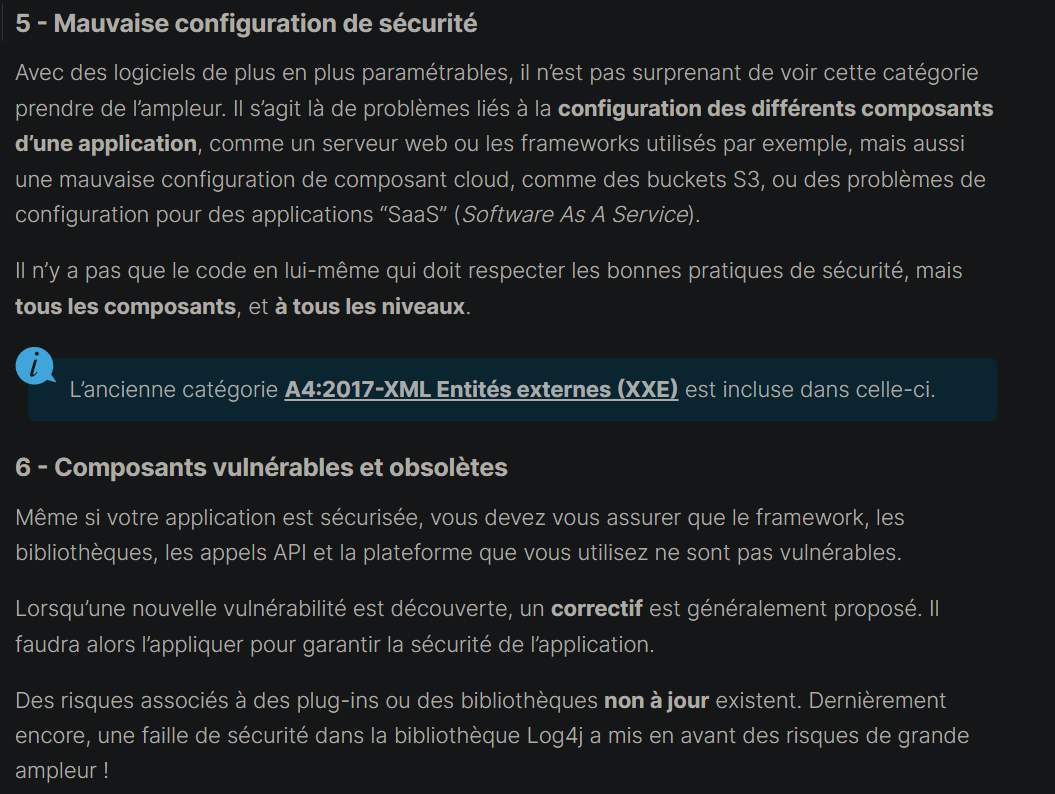


La norme**Payment Card Industry Data Security Standard (PCI DSS)** est une norme établie pour toutes les entreprises qui traitent des**données bancaires**.



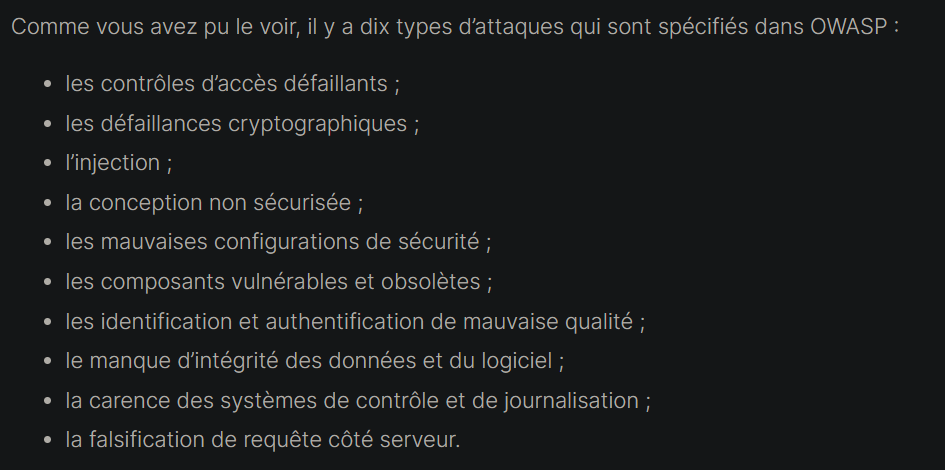












**Empêchez l’exploitation des contrôles d’accès défaillants.**

Le contrôle d’accès, c’est la manière dont une application web gère les autorisations une fois que vous avez procédé à une authentification.

Autrement, contrôler l’accès revient à configurer votre application web, pour vous assurer que vos utilisateurs ne peuvent accéder qu’aux données autorisées par leurs rôles ! Si les contrôles ne sont pas bien sécurisés, un attaquant pourrait accéder à des parties non autorisées et non protégées de notre applications, par exemple en devinant les urls des pages auxquels il n’est pas censé avoir accès.

Je suis 3 principes du contrôle d’accès :

* Le refus par défaut : tout ce qui n’est pas explicitement autorisé est bloqué, à l’exception des ressources publiques, tout accès doit être conditionnée par une authentification.
* Le moindre privilège, on attribue aux différents profils que les accès qu’ils lui sont nécessaires. Les mécanismes du contrôle d’accès doivent être centralisé tout au long de notre application, cela parmi d’éviter les exceptions et les erreurs d’implémentation !
* Exécution du contrôle côté serveur : tout mécanisme de contrôle d’accès implémenté coté client ! peut être contourné voire désactiver par un utilisateur malveillant, donc on s’assure que le contrôle d’accès est programmé coté serveur !

Authorization may be defined as "the process of verifying that a requested action or service is approved for a specific entity" ([NIST](https://csrc.nist.gov/glossary/term/authorization)). Authorization is distinct from authentication which is the process of verifying an entity's identity. When designing and developing a software solution, it is important to keep these distinctions in mind. A user who has been authenticated (perhaps by providing a username and password) is often not authorized to access every resource and perform every action that is technically possible through a system. For example, a web app may have both regular users and admins, with the admins being able to perform actions the average user is not privileged to do so, even though they have been authenticated. Additionally, authentication is not always required for accessing resources; an unauthenticated user may be authorized to access certain public resources, such as an image or login page, or even an entire web app.

<https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Authorization_Cheat_Sheet.md>

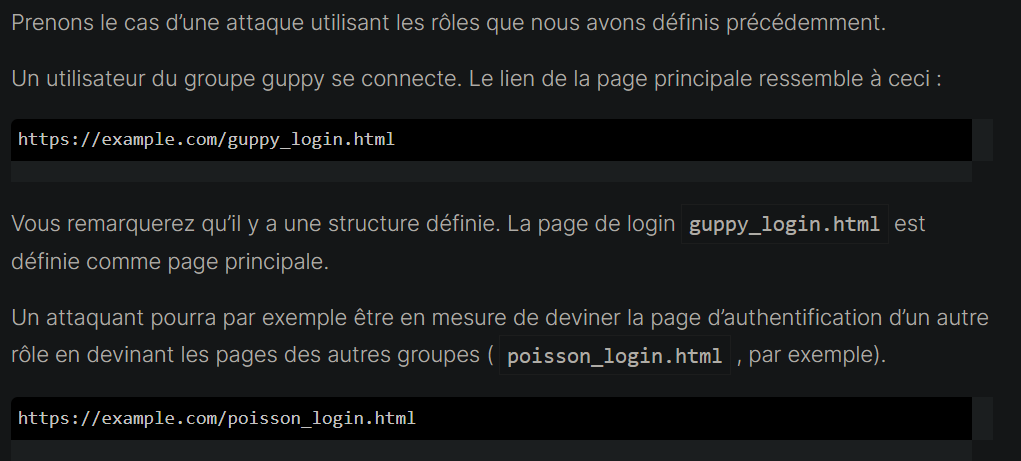
Flaws related to authorization logic are a notable concern for web apps. Broken Access Control was ranked as the most concerning web security vulnerability in [OWASP's 2021 Top 10](https://owasp.org/Top10/A01_2021-Broken_Access_Control/) and asserted to have a "High" likelihood of exploit by [MITRE's CWE program](https://cwe.mitre.org/data/definitions/285.html). Furthermore, according to [Veracode's State of Software Vol. 10](https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-volume-10-veracode-report.pdf), Access Control was among the more common of OWASP's Top 10 risks to be involved in exploits and security incidents despite being among the least prevalent of those examined.

The potential impact resulting from exploitation of authorization flaws is highly variable, both in form and severity. Attackers may be able read, create, modify, or delete resources that were meant to be protected (thus jeopardizing their confidentiality, integrity, and/or availability); however, the actual impact of such actions is necessarily linked to the criticality and sensitivity of the compromised resources. Thus, the business cost of a successfully exploited authorization flaw can range from very low to extremely high.

Both entirely unauthenticated outsiders and authenticated (but not necessarily authorized) users can take advantage of authorization weaknesses. Although honest mistakes or carelessness on the part of non-malicious entities may enable authorization bypasses, malicious intent is typically required for access control threats to be fully realized. Horizontal privilege elevation (i.e. being able to access another user's resources) is an especially common weakness that an authenticated user may be able to take advantage of. Faults related to authorization control can allow malicious insiders and outsiders alike to view, modify, or delete sensitive resources of all forms (databases records, static files, personally identifiable information (PII), etc.) or perform actions, such as creating a new account or initiating a costly order, that they should not be privileged to do. Furthermore, if logging related to access control is not properly set-up, such authorization violations may go undetected or a least remain unattributable to a particular individual or group.

Les attaques contre le contrôle d’accès.

* **Les attaques courantes contre le contrôle d'accès se produisent lorsqu'une URL permet de contourner l'authentification.** Les pirates informatiques utilisent la connaissance des formats et des modèles pour deviner l'URL des pages privilégiées qui n'ont pas été configurées de manière sécurisée. Pour se protéger contre ce type d'attaque, il est possible de mettre en place une **restriction URL**.



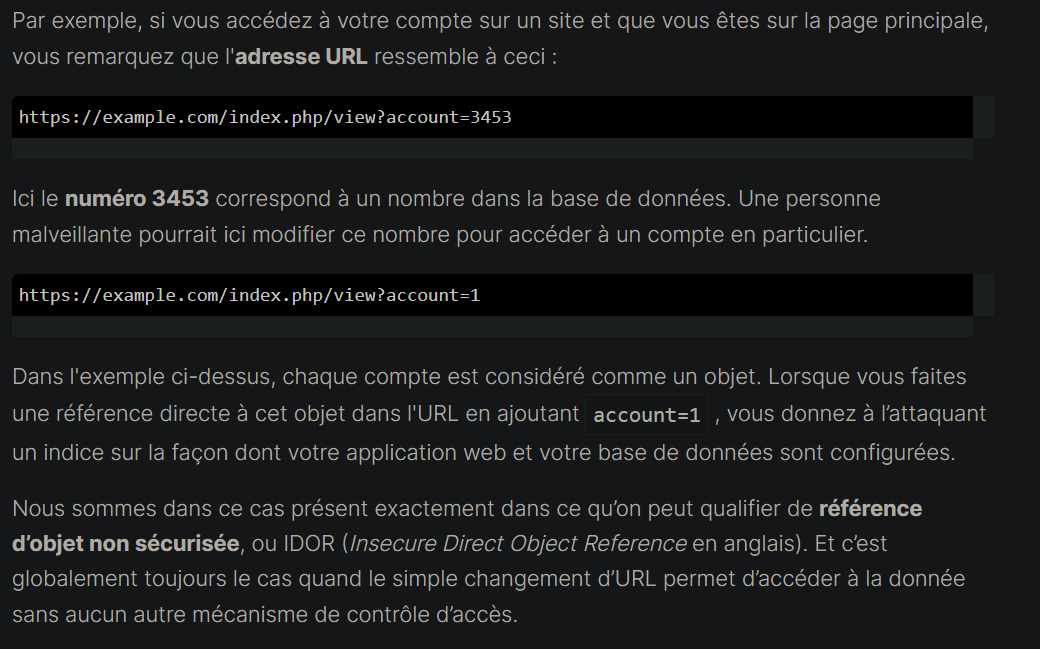


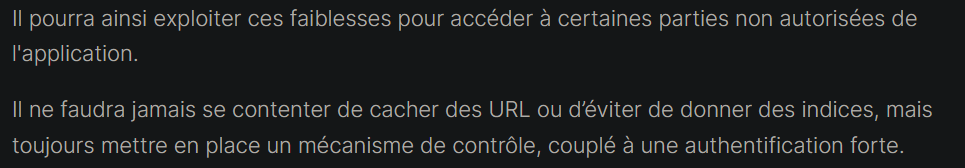


* **Comprenez les références directes d’objets non sécurisées.**

Un utilisateur malveillant dispose de techniques pour accéder à une grande partie du code d'une application web. Une partie de ce code peut révéler **comment une base de données est organisée**. Le fait de fournir quelques informations sur la structure de l’application web peut permettre à un utilisateur non autorisé d’effectuer des actions malveillantes et de contourner les accès prédéfinis.

Externaliser la configuration de l’application web.



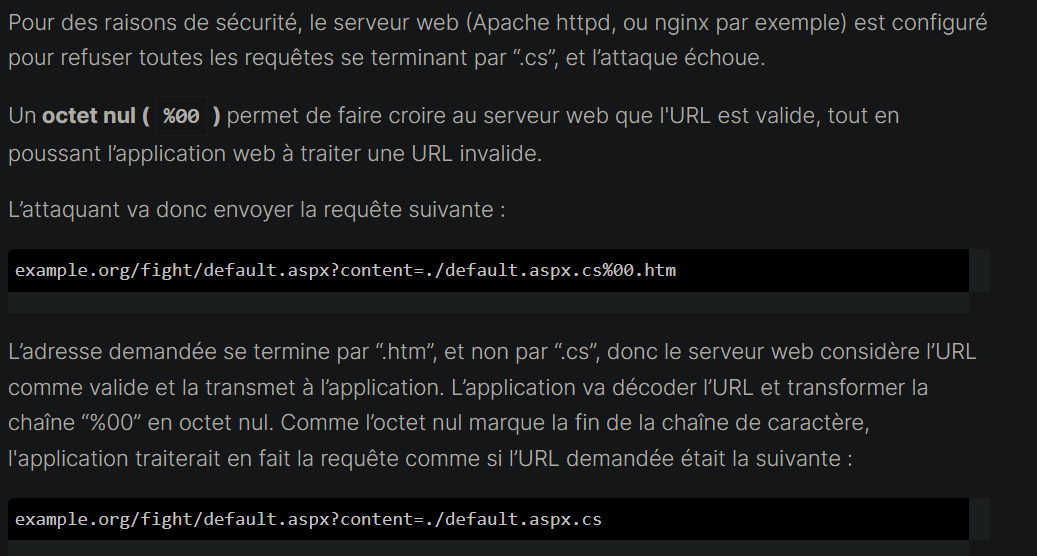
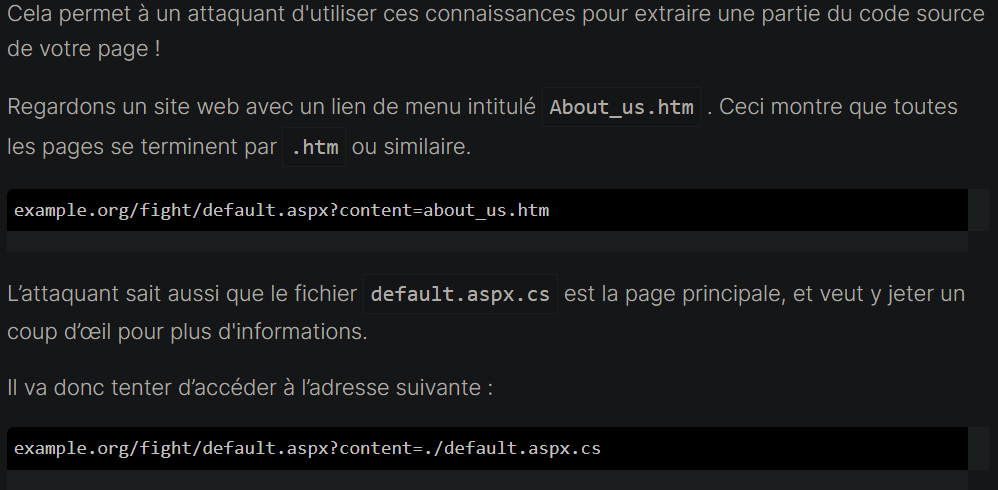


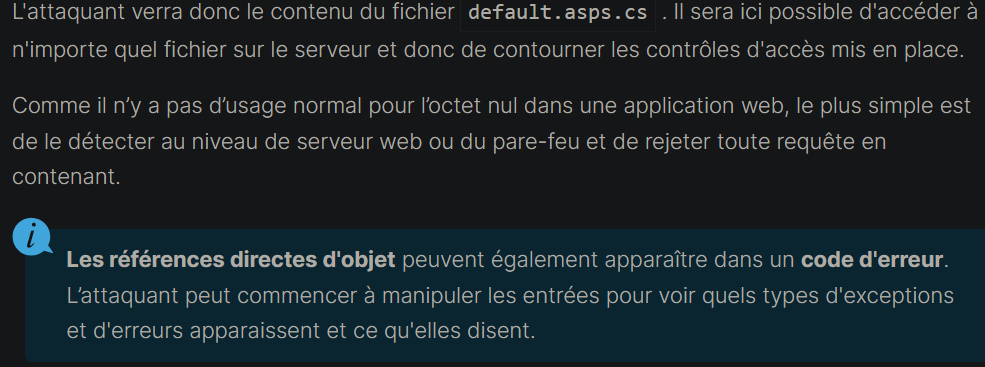
* **Attaque null byte : octet null !**

Une attaque d’octet null permet de contourner certains filtres au niveau du serveur http, en créant une confusion entre l’adresse demandée et celle comprise par l’application.

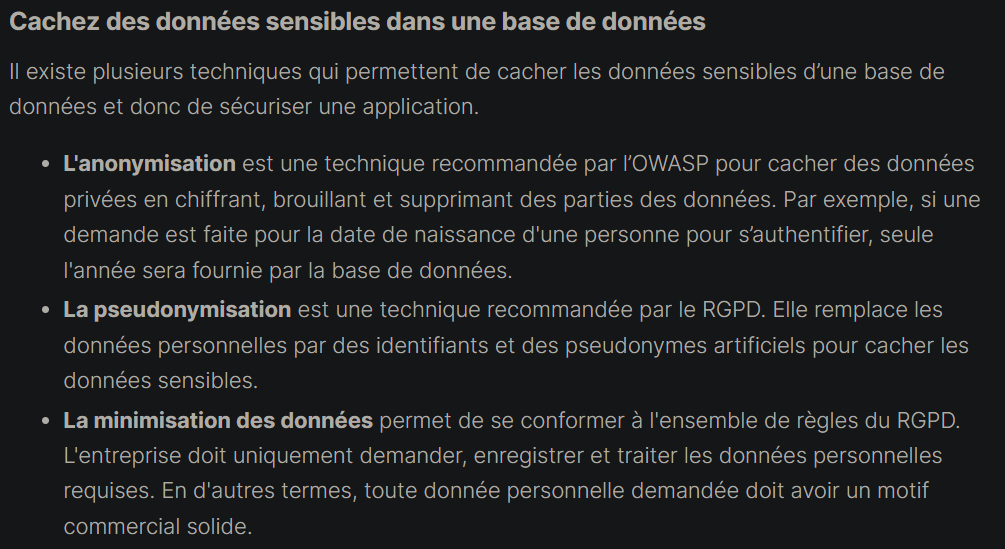
En effet, en insérant la chaîne de caractères “%00” dans une URL, il y a de fortes chances pour que l’application web remplace cette chaîne par un caractère de code ASCII 0, qui sert de marqueur de fin de chaîne de caractères dans de nombreux langages de programmation.

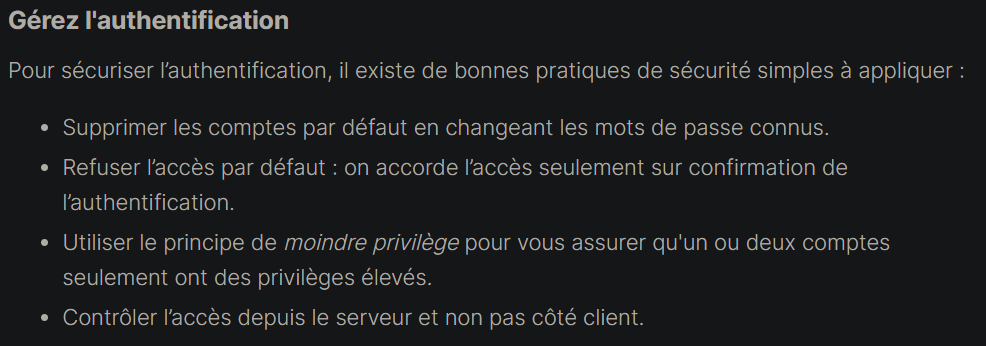
Cela permet à un attaquant d'utiliser ces connaissances pour extraire une partie du code source de votre page !

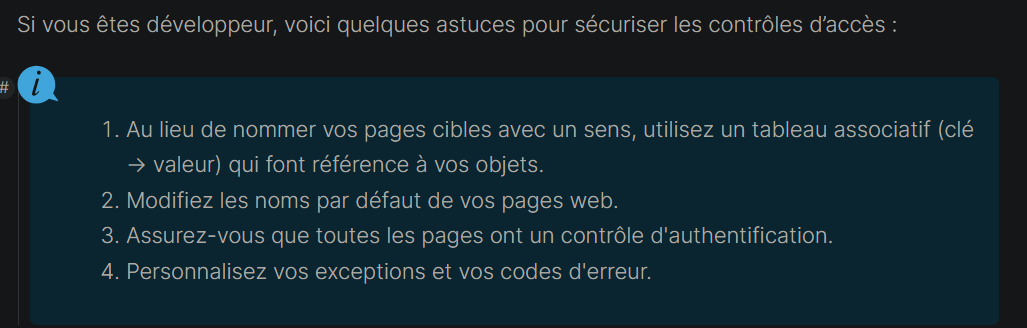


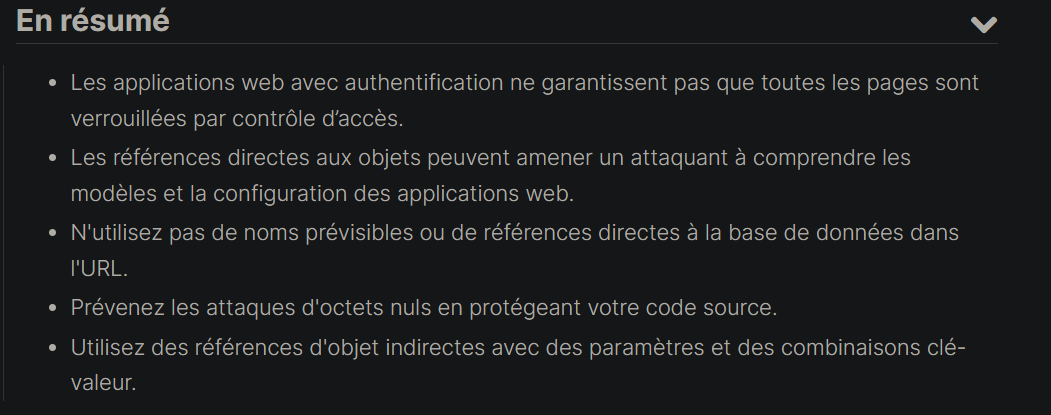


* **Sécurisez une base de données**

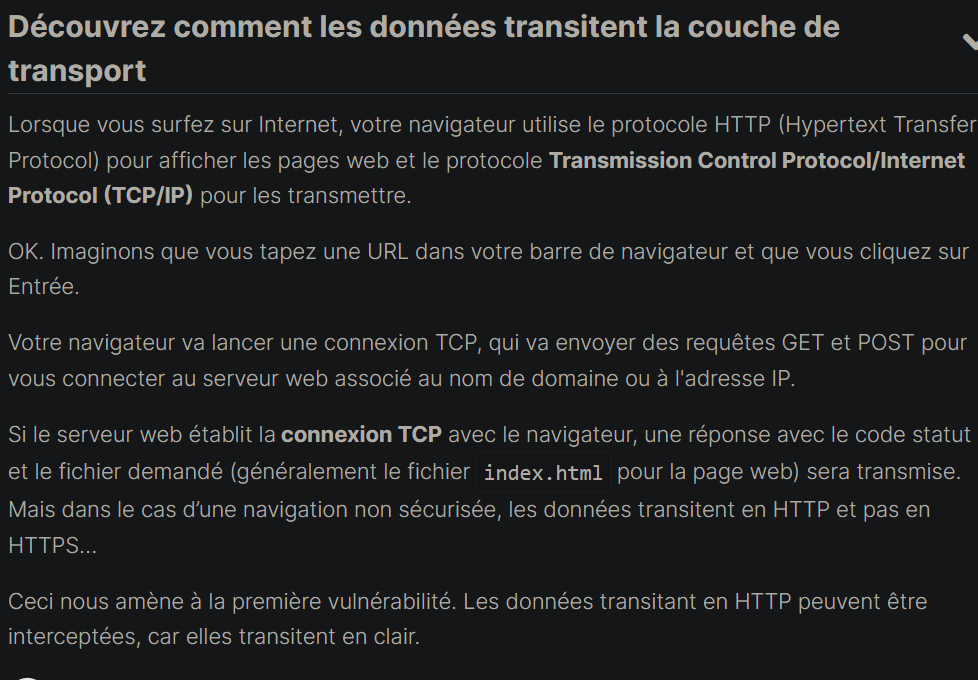


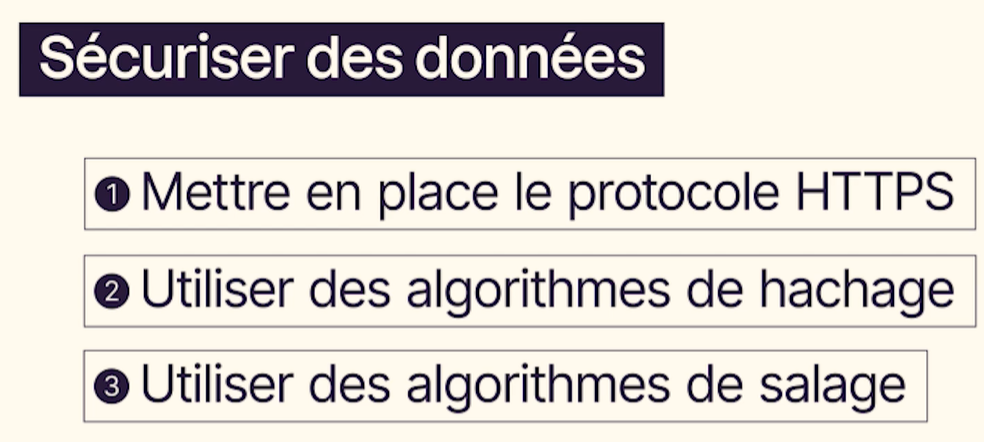






**Sécuriser les données sensibles grâce à la cryptographie :**

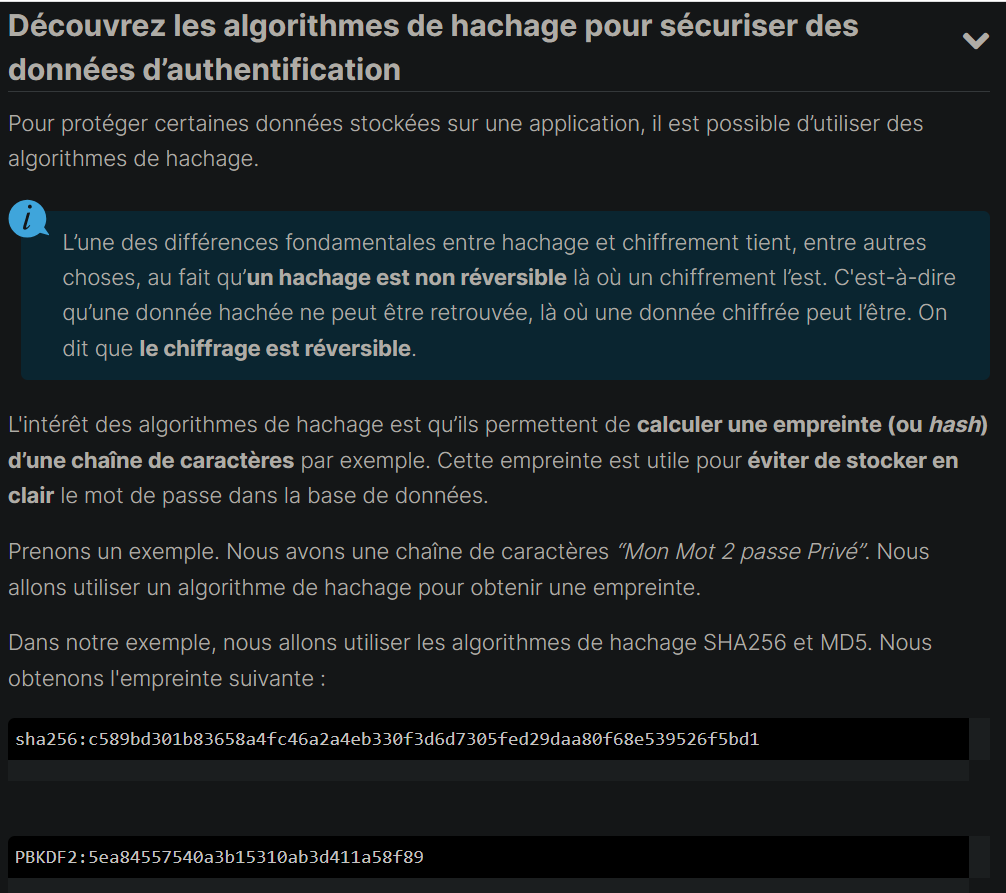


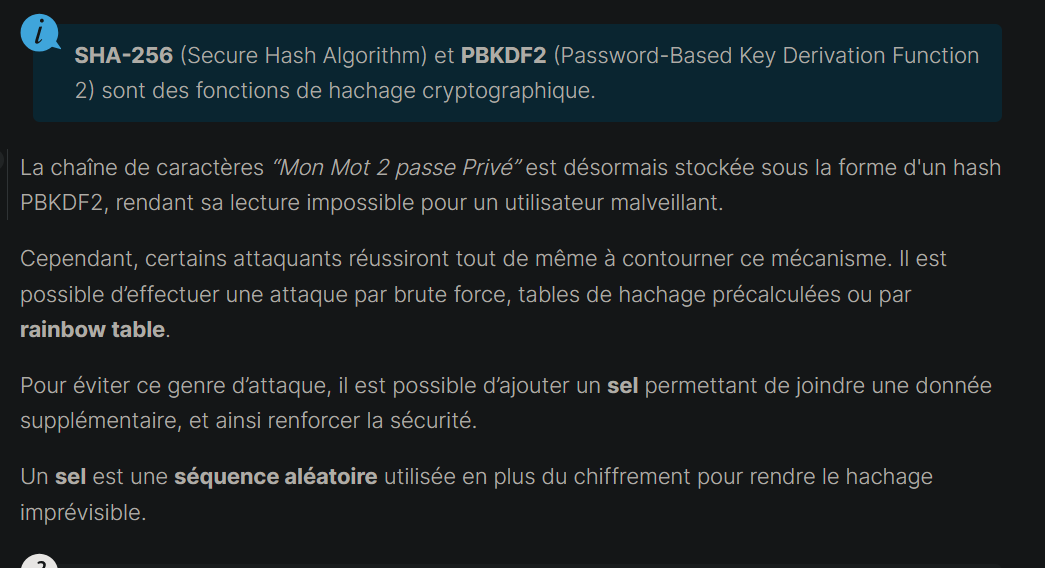


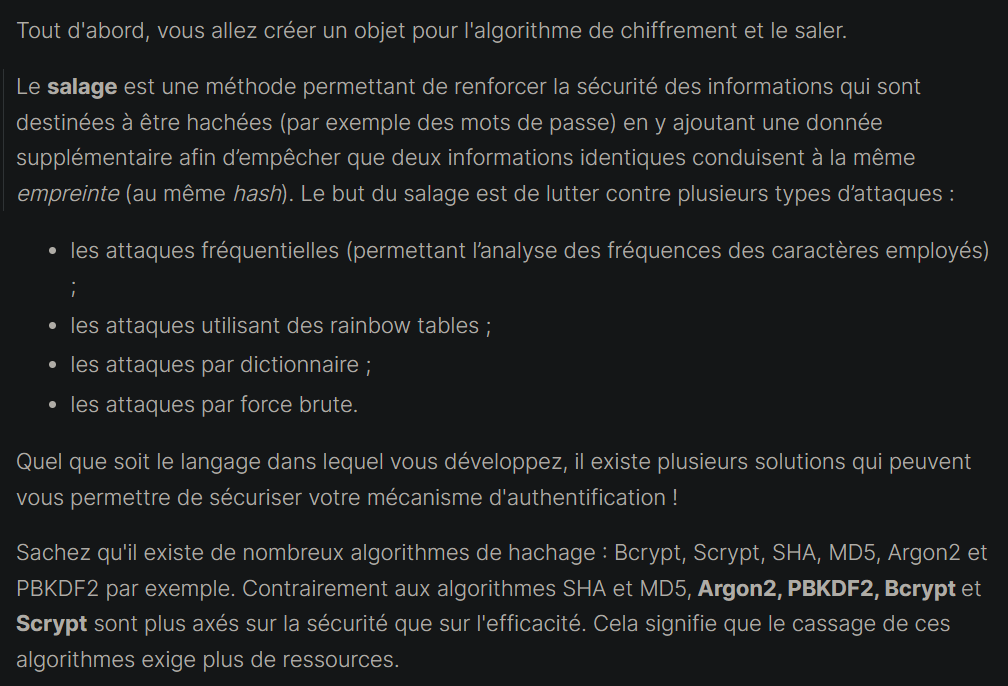
# How to configure HTTPS in a React app on localhost

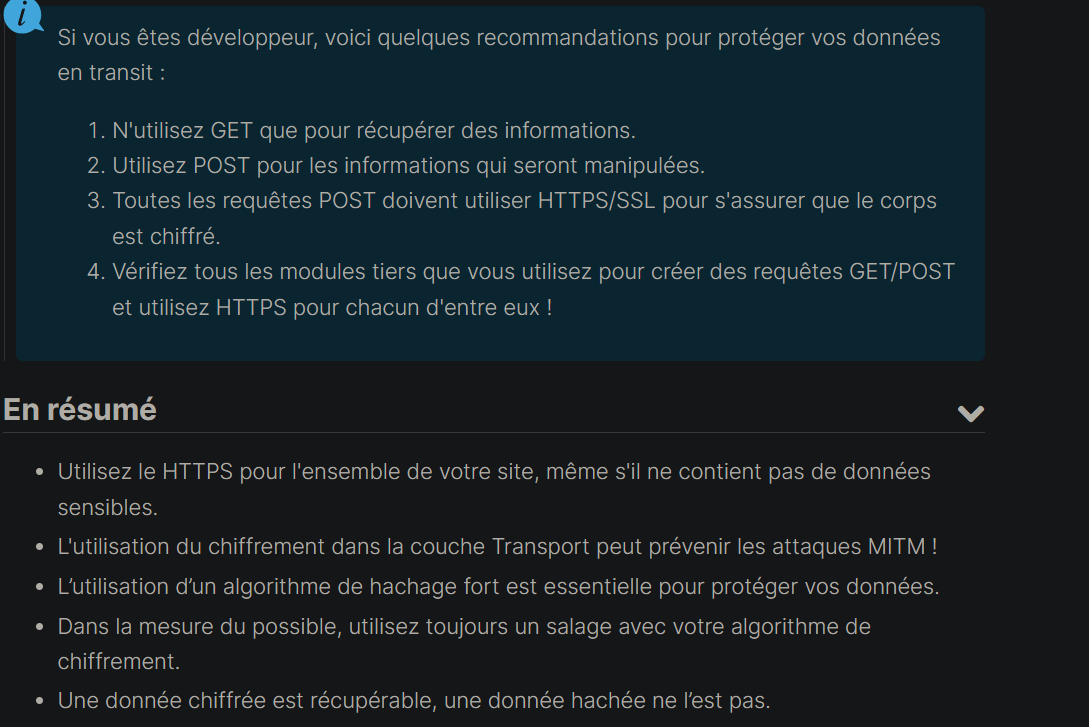
<https://flaviocopes.com/react-how-to-configure-https-localhost/>











**Protégez votre code contre l’injection :**

Attaque par injection consiste à insérer du code arbitraire dans une application pour modifier son fonctionnement.

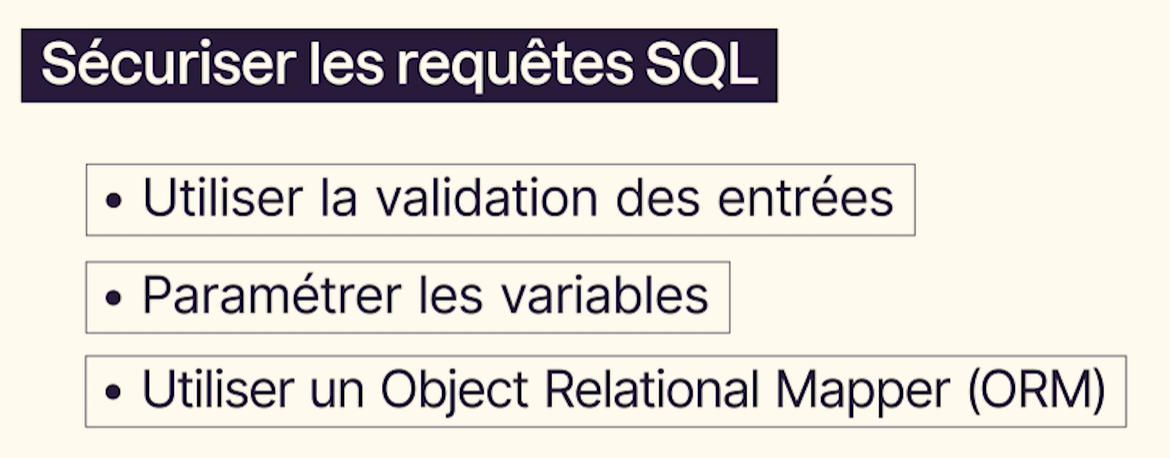
# [**Is SpringBoot Data JPA Repository Safe Against SQL Injection?**](https://stackoverflow.com/questions/73617743/is-springboot-data-jpa-repository-safe-against-sql-injection)

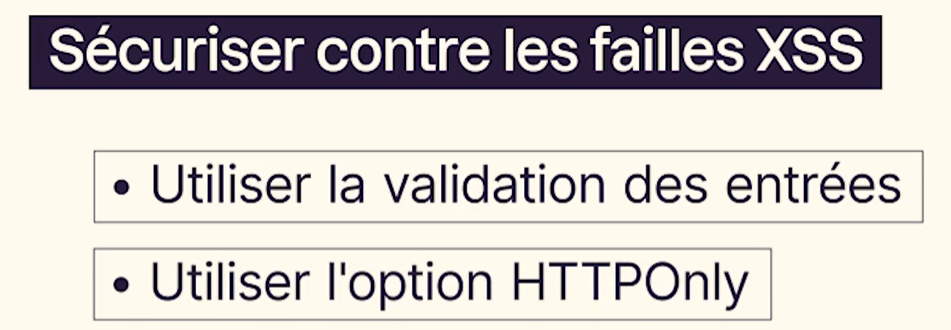
<https://stackoverflow.com/questions/73617743/is-springboot-data-jpa-repository-safe-against-sql-injection>

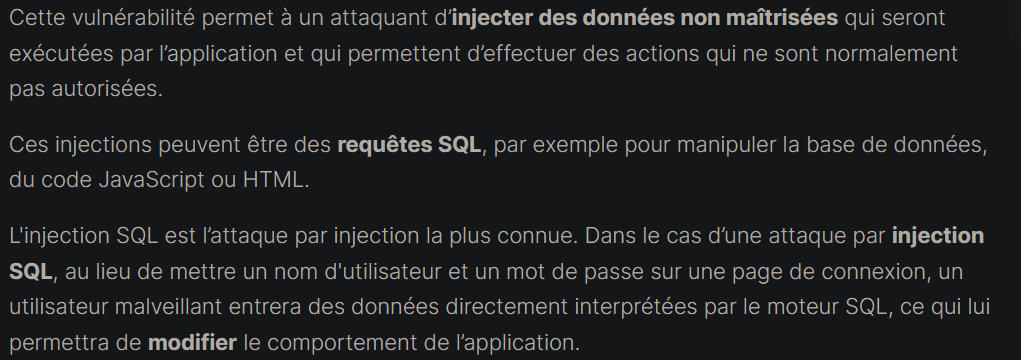
<https://stackoverflow.com/questions/8263371/how-can-prepared-statements-protect-from-sql-injection-attacks/8265319#8265319>

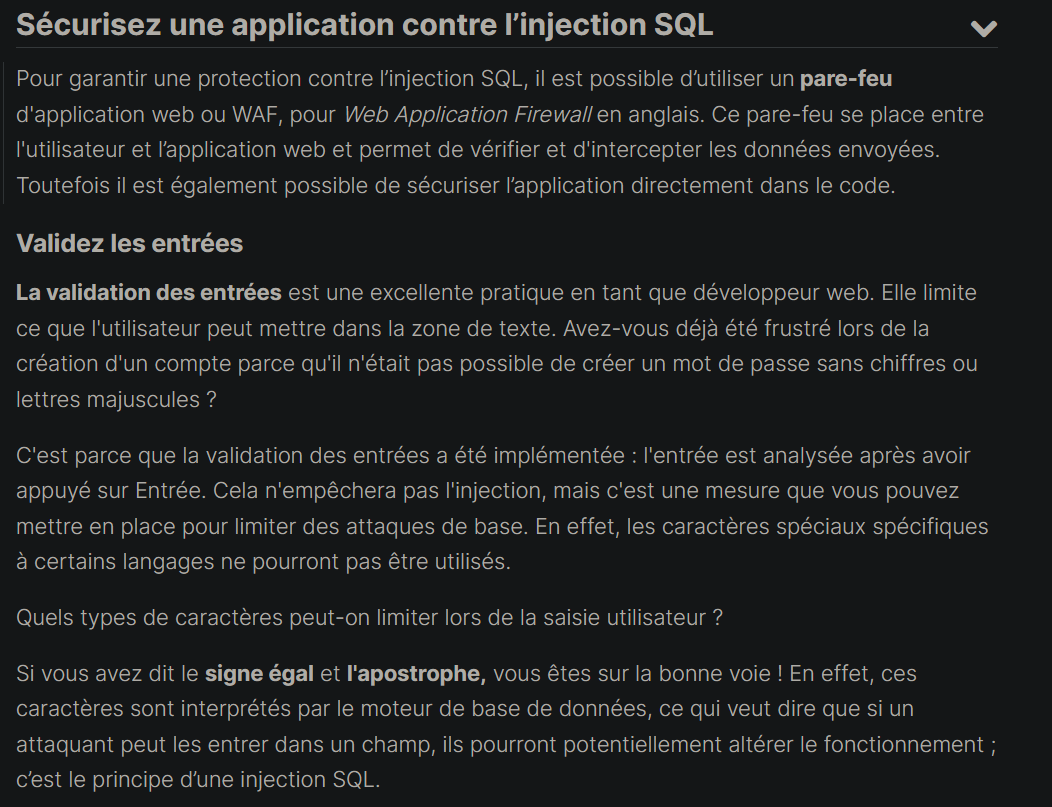
<https://stackoverflow.com/questions/41661193/is-spring-data-jpa-safe-against-sql-injection/41661236#41661236>

spring data jpa is safe against sql injection, only native query can be vulnerable, but if we use @Param it becomes secure also !

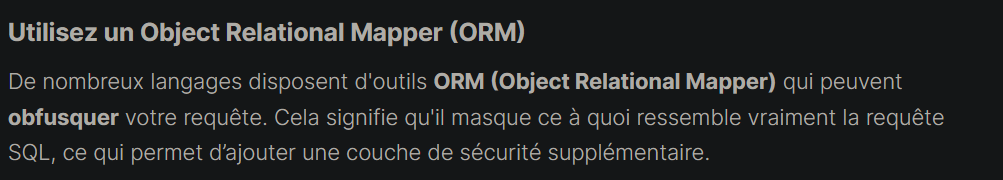






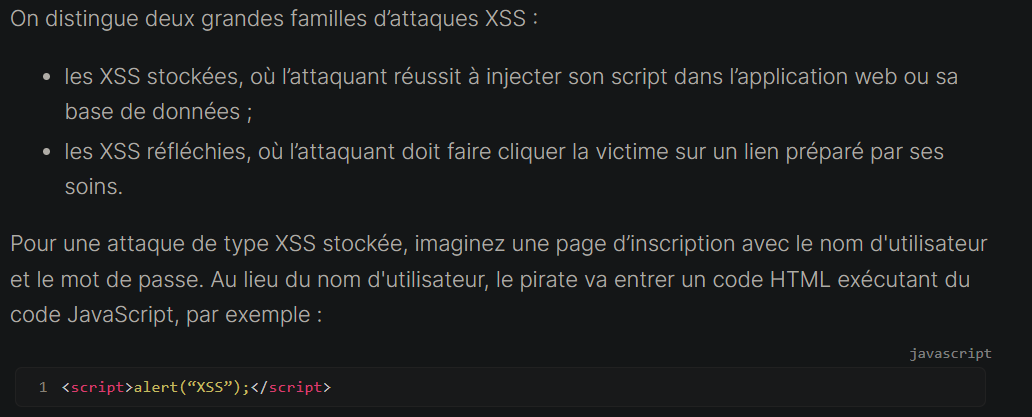




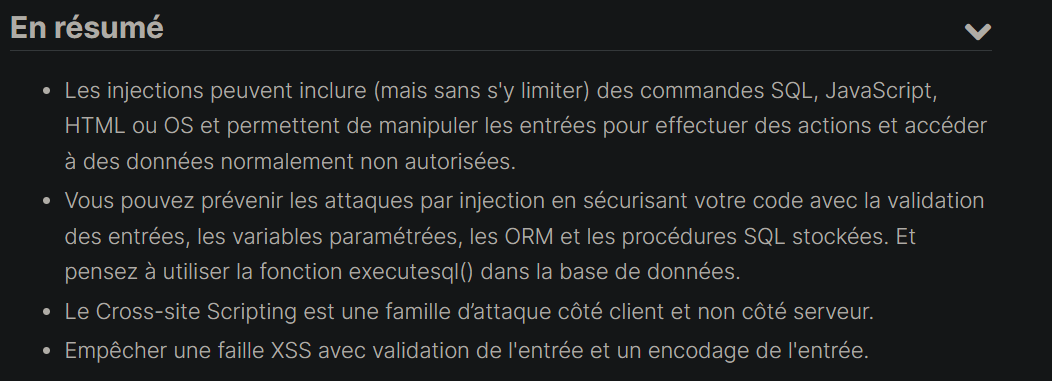


XSS : cross-site scripting









**Il faut sécuriser une application web dès sa conception !**

