Eric BOUDIN

Florian NAUD

Soulaiman ZABOURDINE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ATTAQUE | LIEN | Programme vulnérable? | Exploitation | Défense | Page |
| XSS RULE 0 | [Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-0-never-insert-untrusted-data-except-in-allowed-locations) | OUI | OUI | OUI | 2 |
| XSS RULE 1 | [Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content) | OUI | OUI | OUI | 2 |
| SQL INJECTION 1 | [SQL\_Injection\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html) | OUI | OUI | OUI | 3 |
| SQL INJECTION 2 | [SQL\_Injection\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html) | NON | NON | OUI | 3 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

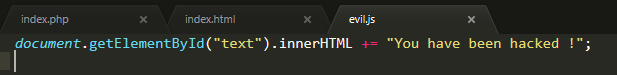
## RULE #0 - Never Insert Untrusted Data Except in Allowed Locations

[Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html" \l "rule-0-never-insert-untrusted-data-except-in-allowed-locations)

Une image contenant texte

Description générée automatiquement

Ici on execute du code venu d’une autre origine que nous. Ainsi, on ne peut garantir qu’il n’y est pas du code malveillant. Dans ce cas-là, cela change l’affichage du site.





Pour se protéger de ceci, on peut mettre appeler le script depuis un iframe. Ainsi la page ne sera plus modifiée.

Une image contenant texte

Description générée automatiquement

## RULE #1 - HTML Encode Before Inserting Untrusted Data into HTML Element Content

[Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content)

Ici, c’est lors de l’envoie de données dans un input que se situe la faille.

Une image contenant texte

Description générée automatiquement

Ici, si l’on met du code JS dans l’input, il sera exécuté.

Une image contenant texte

Description générée automatiquement

Pour s’en protéger, il faut encoder les données :

Une image contenant texte

Description générée automatiquement

Une image contenant texte, personne, capture d’écran

Description générée automatiquement

SQL INJECTION 1

Nos données peuvent être vulnérable si on ne contrôle pas ce qu’injecte les utilisateurs de nos applications. Notre requête sql renvoie la valeur d’un utilisateur donné.



Ici le problème c’est que l’utilisateur peut rentrer n’importe quoi dans le script comme

 (Ce qui donnera la requête sql 2)

On peut donc voir qu’il récupère toutes les données enregistré dans la table.

Pour nous protéger de cela on peut préparer une requête sql.



Ici même en tentant de de remettre notre faille, ça ratera car ça considéra la faille comme un paramètre.

Une image contenant texte

Description générée automatiquement

SQL INJECTION 2

Il suffit de donner seulement les droits nécessaires à un utilisateur de la base de données et seulement eux.

Exemple : Si c’est sur une application qui ne sert qu’à afficher des données, on ne donnera l’accès qu’aux requêtes SELECT pour cette utilisateur. Il n’aura pas besoin d’effectuer d’autres requêtes comme un INSERT INTO.