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 |
| XSS RULE 2 | [Cross Site Scripting Prevention Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-2-attribute-encode-before-inserting-untrusted-data-into-html-common-attributes) | OUI | OUI | OUI |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

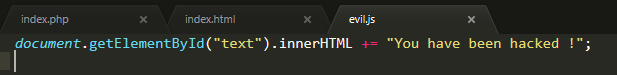
## RULE #0 - Never Insert Untrusted Data Except in Allowed Locations

[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 exécute 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 enregistrées dans la table.

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



Ici même en tentant de remettre notre faille, cela ne fonctionnera pas car cela considérera la faille comme un paramètre.

Une image contenant texte

Description générée automatiquement

## XSS Rule 2

[Cross Site Scripting Prevention Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-2-attribute-encode-before-inserting-untrusted-data-into-html-common-attributes)

Le problème ici concerne encore l’encodage des données non sûres, cette fois dans un attribut href.

Une image contenant texte

Description générée automatiquement

Ici, le script pourrait très bien remplacer le contenu de href par  ce qui exécutera la fonction attack. Une façon très simple de contrer ceci est de remplacer les guillemets par leur version encodé, c’est-à-dire &#x22. Ainsi le code ne sera pas exécuté mais le texte sera affiché en brut.