Question 1 : Perform a store XSS attack to guestbook.php. Defend this with a CSP header.

Il faut rentrer dans le champ input de la page web du code javascript à exécuter dans une balise script. Le navigateur va contacter un autre script avec la valeur de l'input dans le paramètre "message".

```
http://localhost:8080/tp2/ressources/guestbookleavemessage.php?message=<script>alert('xss')</script>
```

Le champ de l'utilisateur va être stocké côté serveur. Nous permettant de réaliser une XSS stored. Lorsque l'utilisateur va recharger le page "tp1/guestbook.php", le contenu du fichier va être récupéré et le code contenu à l'intérieur va être exécuté.

Pour mettre en place le CSP header sur le script php. Il faut rajouter le code php suivant au début du script.

```php
header("Content-Security-Policy: script-src 'self'");
```

Maintenant le client va exécuter que les balises script qui ont pour origine celui du serveur, donc localhost.

Question 2 : xssme.php perform all the attacks and generate protectedxssme.php to defend. Do htmlentities or htmlspecialchars work in every context of xssme.php ? If not, explain and correct.

Script : tp1/xssme.php

Paramètre : htmlcontext

```
htmlcontext=<script>alert('xss');</script>
```

Paramètre : attributecontext1

```
attributecontext1='' onerror=alert('xss')
```

Paramètre : attributecontext2

```
attributecontext2=" onerror=alert('xss')
```

Paramètre : attributecontext3

```
attributecontext3=' onerror=alert('xss')
```

Paramètre : scriptcontext

```
scriptcontext=alert('xss')
```

Paramètre : attributecontextonerror

```
attributecontextonerror=alert('xss')
```

Défenses :

Il faut mettre en place le CSP sur le script-src avec pour valeur self.

```php
header("Content-Security-Policy: script-src 'self'");
```

Un autre moyen est d'encoder les inputs utilisateurs avec des fonctions d'encoding comme htmlentities et htmlspecialchars en php.

Question 3 : Write code to defend and attack for each of the contexts described for XSS and DOM-XSS in the OWASP cheatsheets

Exploitation XSS sur les différents scripts :

domxss4.html :

```
http://localhost:8080/tp2/ressources/domxss4.html?x=<script>alert('xss');</script>
```

domxss5.html :

Mettre une balise script dans l'input ne fonctionne pas, car dans la spécification de HTML "a<script> tag inserted with `innerHTML` should not executed." (Source : Element: innerHTML property - Web APIs | MDN (mozilla.org))

Le moyen est donc d'injecter du code javascript, mais sans utiliser une balise script. L'une des manières de faire est d'utiliser une balise img.

```
<img src='' onerror=alert('xss');>
```

domxss6.html :

window.location.hash permet de récupérer les fragments qui sont spécifiés dans l'url par l'utilisateur c'est à dire, l'ensemble des paramètres spécifiés après le caractère "#".

```
http://localhost:8080:/tp2/ressources/domxss6.html#xss'onclick='alert(1)
```

Le code javascript s'exécute lorsque l'utilisateur clique sur la balise input.

Défenses contre XSS :

- Utiliser le mécanisme CSP
- Encoder les inputs utilisateurs en caractères html

Défenses contre XSS-DOM

- Utiliser le mécanisme de CSP
- Encoder les inputs utilisateurs en caractères html
- Utiliser l'API Trusted Types (Trusted Types API - Web APIs | MDN (mozilla.org))
- Utiliser le mode stricte de javascript ("use script")
- Ne pas utiliser innerHTML sur des inputs non vérifiés et utilisé plutôt à la place innerText.

Question 4 : Investigate how to use Trusted types for DOM-XSS and the new attack

https ://portswigger.net/daily-swig/untrusted-types-researcher-demos-trick-to-beat-trusted-types-protection-in-google-chrome

Fonctionnement de la Trusted types API :

La trusted types API permet de forcer la validation des chaines de caractères employées pour l'utilisation des fonctions constituant des "injections sinks". Les injections sinks sont des fonctions qui insèrent du HTML dans le document, des fonctions qui créent un nouveau document de même origine avec un balisage contrôlé par l'appelant, des fonctions qui exécutent du code et des setteurs pour les attributs d'éléments qui acceptent une URL pour charger ou exécuter du code).

Si l'utilisateur utilise une string au lieu d'un trusted type fournit par une fonction de validation définie dans un objet TrustedTypePolicyFactory, l'utilisateur va recevoir une "TypeError".

L'attaque : CVE-2022-1494

Il n'y a pas beaucoup d'information sur la vulnérabilité, à l'exception que le problème vient "d'une validation insuffisante des données dans Trusted Types dans Google Chrome avant la version 101.0.4951.41 permettait à un attaquant distant de contourner les trusted types policies via une page HTML élaborée."

Source : NVD - CVE-2022-1494 (nist.gov)