Question 1 : See code for boot.js, trusted.js, mashup1.js. Without changing this code, write code for attacker.js in order to make trusted.js execute unwanted code.

Ici, nous allons surcharger la référence de `XMLHttpRequest` de la manière suivante :

```
// attacker.js

let originalXMLHttpRequest = window.XMLHttpRequest;

class XMLHttpRequest {
    constructor() {
        console.log('Code executed by the hacker in XMLHttpRequest constructor');
        this.http = new originalXMLHttpRequest();
    }
    open(method, url, async) {
        console.log('Code executed by the hacker in XMLHttpRequest open');
        this.http.open(method, url, async);
    }
    send() {
        console.log('Code executed by the hacker in XMLHttpRequest send');
        this.http.send();
    }
}
```

Ainsi, lorsque le code va utiliser `XMLHttpRequest` et les méthodes : `constructor`, `open` et `send`, nos codes vont être exécutés.

Question 2 : Change boot.js (and if needed trusted.js) to avoid the attack

Il faut définir notre objet `XMLHttpRequest` stocké dans la variable `xmlhttp` dans le fichier `boot.js` qui est exécuté avant le script de l'attaquant. Il faut ensuite gelé l'objet, pour qu'il ne puisse plus être modifié avec la fonction `Object.freeze`. Ainsi l'attaquant ne pourra pas modifier les propriétés de l'objet stocké dans la variable `xmlhttp`.

```
// boot.js

const xmlhttp = new XMLHttpRequest();
Object.freeze(xmlhttp);
```

Question 3 : Let adapi.js be the code for some external gadget. Assume that code for adapi.js has been verified and cannot access window directly, however it

— has access to function integrator whenever it is available. For each of the following
— versions of the mashup, can the external gadget adapi.js
— read the value of secret ?
— obtain a pointer to window ?

Version 1 :

— Lire secret : oui
— Accès windows : oui

```
function integrator() {
```

```
    secret = 42;
    return this;
}
```

Version 2 :

— Lire secret : non
— Accès windows : oui

```
function integrator(){
    var secret = 42;
    return this;
}
```

Version 3 :

— Lire secret : non
— Accès windows : non

```
(function () {
    secret = 42;
    return this;
})()
```

Version 4 :

— Lire secret : non
— Accès windows : oui

```
integrator = function(){
    var secret = 42;
    return this;
}
```

Question 4 : Assume you have a function lookup that will replace any access to a property of the form o[prop] in attacker code by lookup(o, prop). The goal of lookup is to prevent any access to a special property "secretproperty". Which of the following 2 implementations of lookup satisfy this goal ? Justify your answer.

Version 1 :

```
lookup1 = function(o, prop){
    if (prop === 'secretproperty'){
        return "unsafe!";
    } else {
        return o[prop];
    }
}
```

Safe : Non

Définition d'un objet `String` initialiser avec la valeur "secretproperty".

Bypass v1 :

```
prop = new String("secretproperty");
```

Bypass v2 :

Définition d'un objet contenant la définition de la propriété `toString`, en envoyant la chaine de caractères "secretproperty".

```
prop = {toString : function() {return "secretproperty";}};
```

Mitigation :

Utilisation d'un == à la place d'un ===.

```
lookup1 = function(o, prop){
    if (prop == 'secretproperty'){
        return "unsafe!";
    } else {
        return o[prop];
    }
}
```

Version 2 :

```
lookup2 = function(o, prop){
    var goodprop = { publicproperty: 'publicproperty', secretproperty:
        'publicproperty'}[prop];
    return o[goodprop];
}
```

Safe : Oui