Geek Culture

intigriti Challenge 0921 by Bug Emir & Pepijn van der Stap

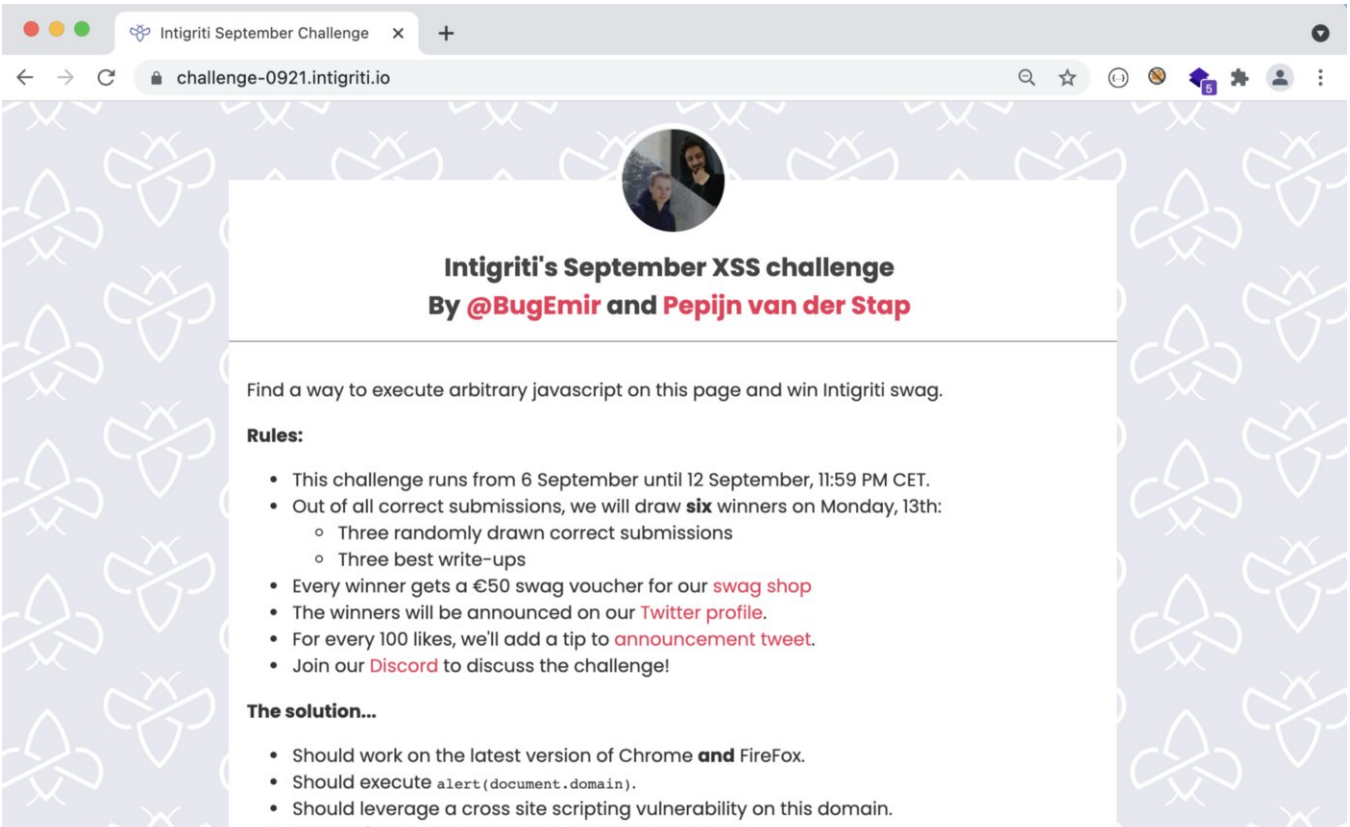
RahulSL Follow
Sep 13 · 9 min read

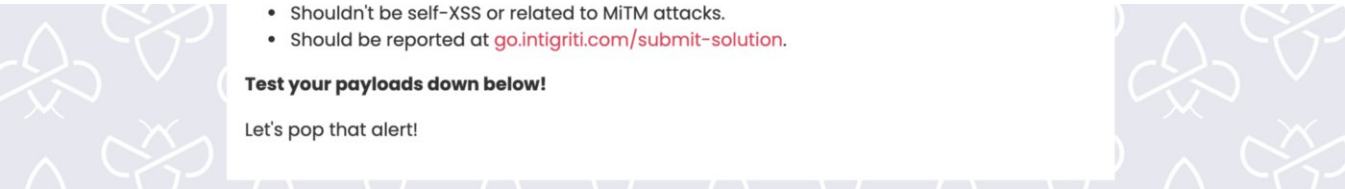
Yet another awesome XSS challenge from Intigriti. By solving the challenge learned cool stuff about DOM-based XSS, hope you will enjoy reading this article about how I solved this challenge & won Intigriti Swag for one the best writeup 😋

Let's get to it.

https://challenge-0921.intigriti.io/

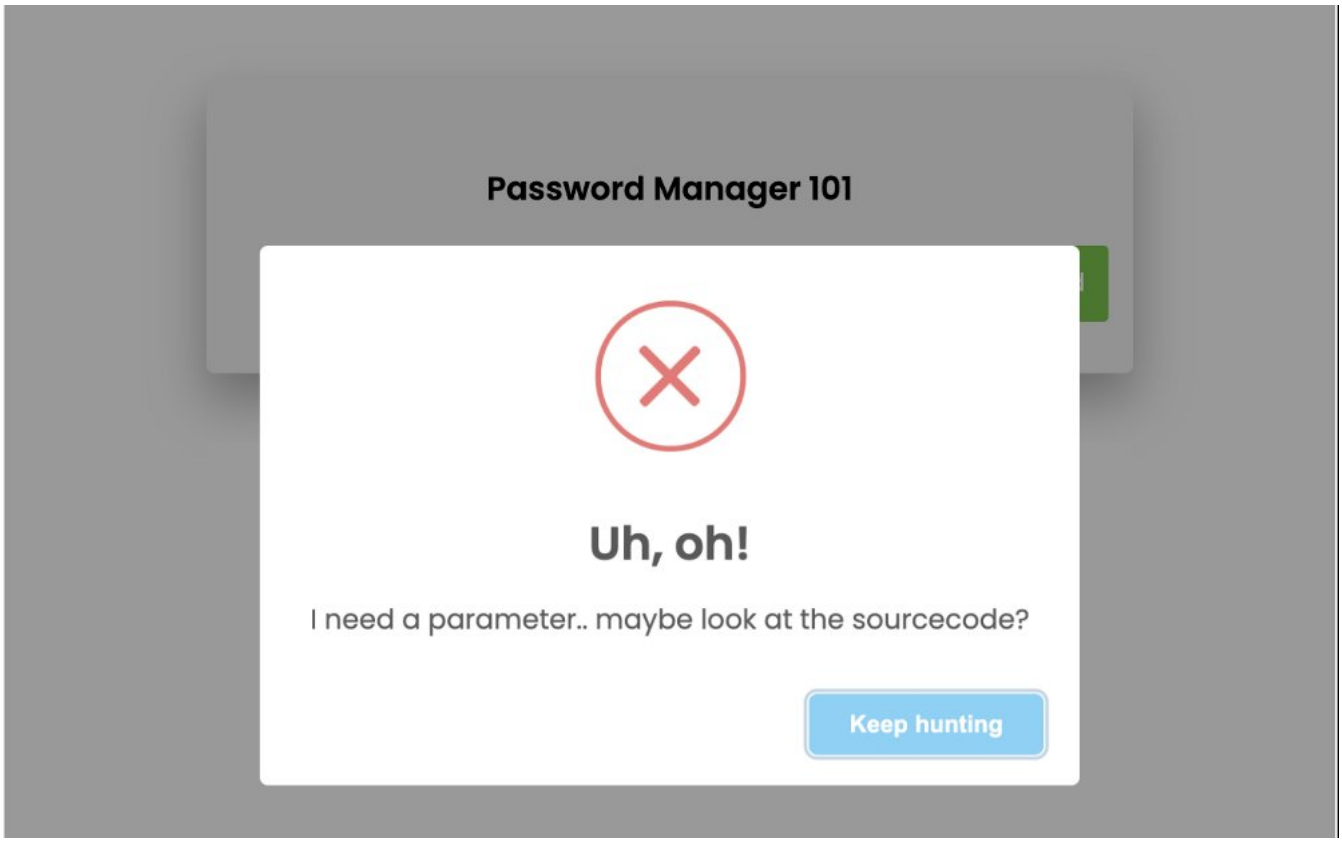
. . .





Challenge Overview & Challenge Area with user input field, hmm XSS 😍 ??

Let's enter some characters into the text field and see what happens!



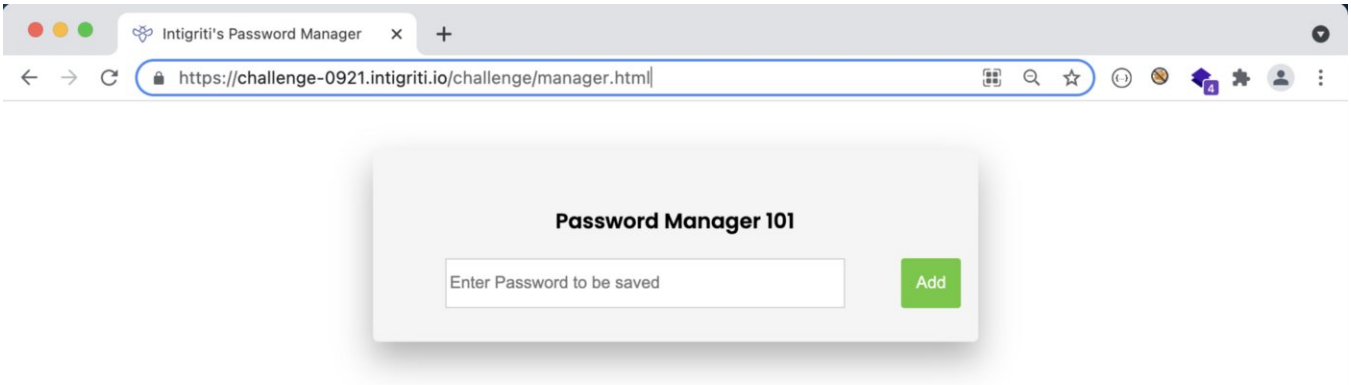
As you can see model error message popup requesting a parameter. This is a good indication there might be a parameter that takes in a value to save a password, also as you can see the second part of the message “may be look at the source code?”. Ok, let’s look at the source code then.

. . .

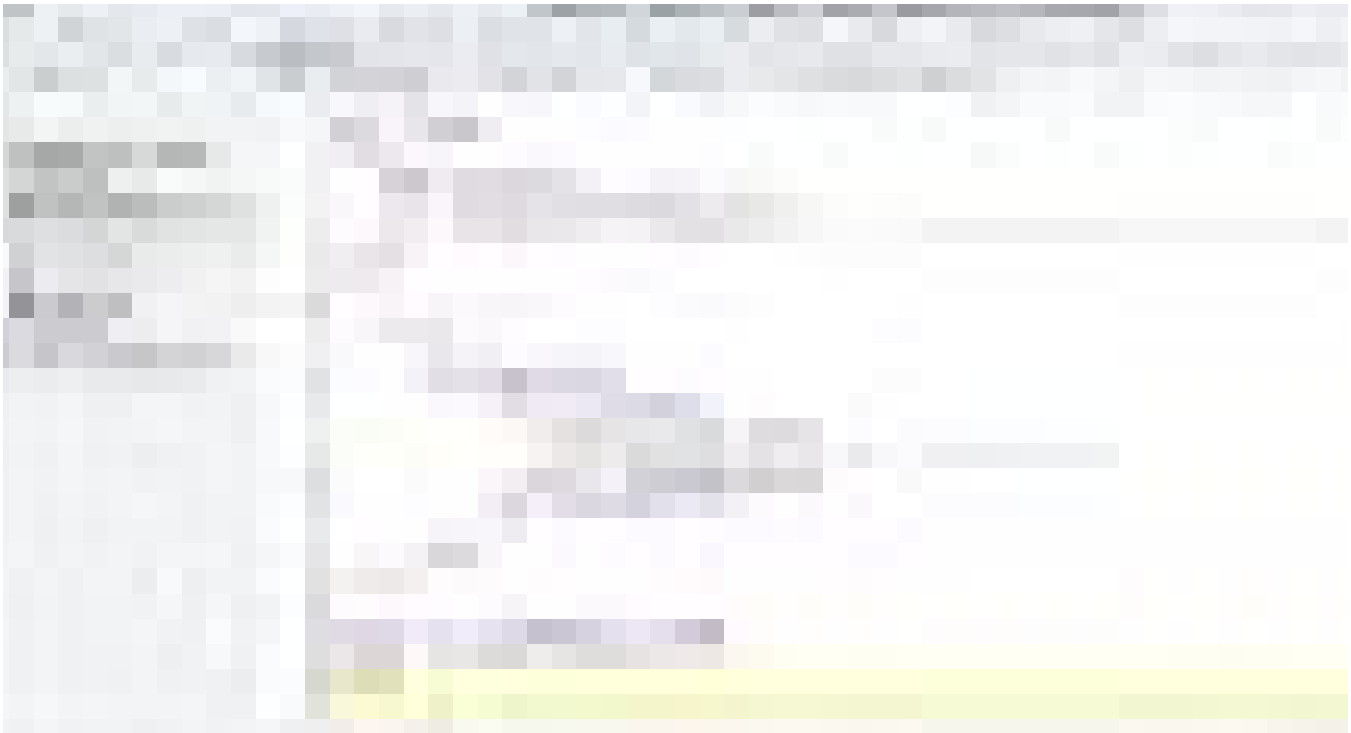
```
    <meta property="og:title" content="August XSS Challenge - Intigriti">
    <meta property="og:description" content="Find the XSS and WIN Intigriti swag.">
    <meta property="og:image" content="https://challenge-0921.intigriti.io/share.jpg">
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">
    <link href="style.css" rel="stylesheet">
  </head>
▼<body cz-shortcut-listen="true">
  ▼<section id="wrapper">
    ▼<section id="rules">
      ▼<div id="challenge-container" class="card-container">
        ▶<div class="card-header">…</div>
        ▼<div id="challenge-info" class="card-content">
            <p>Find a way to execute arbitrary javascript on this page and win Intigriti swag.</p>
            <b>Rules:</b>
          ▶<ul>…</ul>
            <b>The solution...</b>
          ▶<ul>…</ul>
            <b>Test your payloads down below!</b>
            <p>Let's pop that alert!</p>
          </div>
        </div>
      ▼<div class="card-container">
        ▼<iframe src="challenge/manager.html" width="100%" height="600px">
          ▼#document
              <!DOCTYPE html>
            ▼<html lang="en">
              ▼<head>
                ▶<style type="text/css">…</style>
                  <meta charset="UTF-8">
                  <title>Intigriti's Password Manager</title>
                  <link rel="stylesheet" href="style.css">
                  <link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css" rel="stylesheet">
                </head>
              ▼<body>
                ▶<div class="container">…</div>
                  <script async src="manager.js"></script>
                  <script src="sweetalert.min.js"></script>
                ▶<div class="swal-overlay" tabindex="-1">…</div>
                </body>
              </html>
          </iframe>
        </div>
      </section>
    </section>
  </body>
…</html> == $0
```

We can see main page uses an iframe with source ***(src=”challenge/manager.html”)*** and uses two javascript files. Let’s visit this page then.

https://challenge-0921.intigriti.io/challenge/manager.html



Let's check the code inside those javascript files as an error message hinting.



In that page's source, we can find **javascript library "sweetalert.min.js & "manager.js"** being loaded.

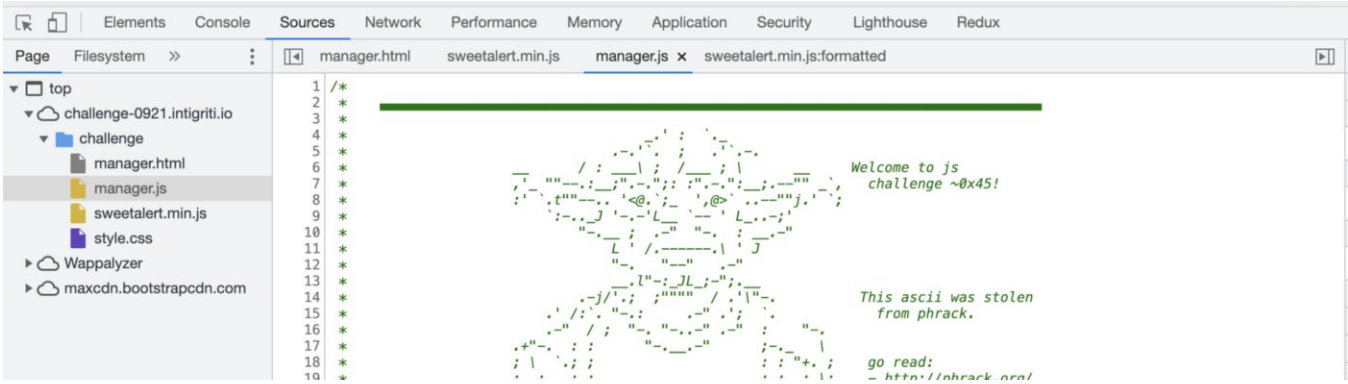
First checked the *sweetalert.min.js* file, because of the error alert message popup asking for a parameter. Found that is file is for the SweetAlert library for those cool popups.

This is very bottom part of the “sweetAlert.js” file!

When I saw the above-highlighted part of the code thought, “ok this is an indication this library might be using the old version”. As you can see ***/guide/#upgrading-from-1x***, then googled **SweetAlert vulnerabilities**. Got some article showing this library is (older versions) vulnerable to XSS attacks.

Crafted some payloads entered into the input box and no luck, still getting the same message “I need a parameter”. Time to check the other Javascript file **“manager.js”**….. Here are all the fun begins… 😄

. . .



```
*    - hmm, is it [illegible]?
*
*    ================================================================
*/

(function(c, d) {
    function fm(c, d) {
        return b(c - 0x2f3, d);
    }
    var e = c();
    while (!![]) {
        try {
            var f = parseInt(fm(0x4cf, 0x652)) / (-0x112c + 0x2056 + -0xf29) * (-parseInt(fm(0x482, 0x5ee)) / (-0x1a7b + -
            if (f === d) {
                break;
            } else {
```

{} Line 41, Column 3 Coverage: n/a

Hola Yoda.. What are you doing here? 👽

Page | Filesystem | manager.html | sweetalert.min.js | manager.js × | sweetalert.min.js:formatted

- top
  - challenge-0921.intigriti.io
    - challenge
      - manager.html
      - manager.js
      - sweetalert.min.js
      - style.css
  - Wappalyzer
  - maxcdn.bootstrapcdn.com

```
*    - hmm, is it [illegible]?
*
*    ================================================================
*/

(function(c, d) {
    function fm(c, d) {
        return b(c - 0x2f3, d);
    }
    var e = c();
    while (!![]) {
        try {
            var f = parseInt(fm(0x4cf, 0x652)) / (-0x112c + 0x2056 + -0xf29) * (-parseInt(fm(0x482, 0x5ee)) / (-0x1a7b + -
            if (f === d) {
                break;
            } else {
                e['push'](e['shift']());
            }
        } catch (g) {
            e['push'](e['shift']());
        }
    }
}(a, 0x9a1c7 + 0xa21 * -0x4 + -0x1 * -0xad81));

function b(c, d) {
    var e = a();
    b = function(f, g) {
        f = f - (0x242 + -0x1feb * -0x1 + -0x2163);
        var h = e[f];
        if (b['HZYYok'] === undefined) {
            var i = function(m) {
                var n = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/=';
                var o = '';
                var p = '';
                for (var q = -0xfcd * -0x1 + -0x1 * 0x1a51 + 0x2a1 * 0x4, r, s, t = -0x35 * -0xe + -0x43 * -0x3b + -0x1257;
                    s = n['indexOf'](s);
                }
                for (var u = 0xd6 * 0x24 + 0xc08 + 0x10 * -0x2a2, v = o['length']; u < v; u++) {
                    p += '%' + ('00' + o['charCodeAt'](u)['toString'](0x1 * 0xaf5 + 0x8d4 + -0x13b9))['slice'](-(-0x1d95 +
                }
                return decodeURIComponent(p);
            };
            b['hPvxJz'] = i;
            c = arguments;
            b['HZYYok'] = !![];
        }
        var j = e[-0x19da + 0xb * -0x113 + -0x25ab * -0x1];
        var k = f + j;
        var l = c[k];
```

What is this ??

{} Line 41, Column 3 Coverage: n/a

Hexadecimal.. god this going to be more fun to crack this... 😬

This looks interesting… □ ***"This ASCII was stolen from phrack."*** Okay.. & **"go *read: — http://phrack.org/*"**. There might be some hints for payloads here, visited http://phrack.org

After spending some time on ***http://phrack.org/*** realized it's a rabbit hole, couldn't find anything related to this challenge. I am not sure maybe someone else found some useful stuff here related to this XSS challenge. (Will wait for the official report).

> *But there are some cool articles about some exploits, I believe it is worth a read, Okay maybe later…*□

. . .

Before moving to check the code for that mystery parameter below part caught my eye (Highlighted on above image).

"- hmm, is it □□□□□□□□□□□?" whooo Braille ♨ but I cannot read it, let's ask google what this means? (http://xahlee.info/comp/unicode_braille.html)

□□□□□□□□□□□ means **obfuscated,** this explains why code in this js file looks funny. Anyway, let's see what more surprises are in here □

Tip # 1 from Intigriti, but this time I've already known that this code is obfuscated and needs deobfuscating (reverse engineering).

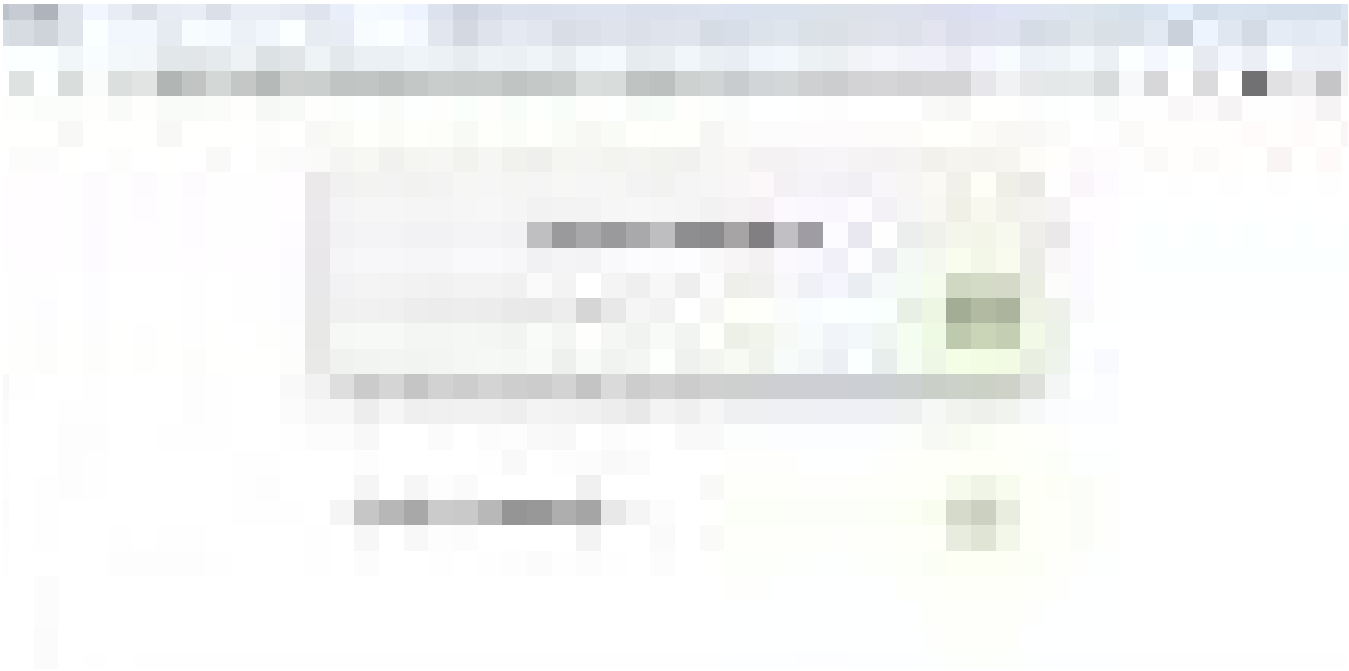
But before asking google for deobfuscating help I've decided to scan the code and found ***_0x5195*** being used all over the manager.js file.

Array "***_0x5195***", the game-changer… 😚 So to see the array values I used this javascript code in the developer console

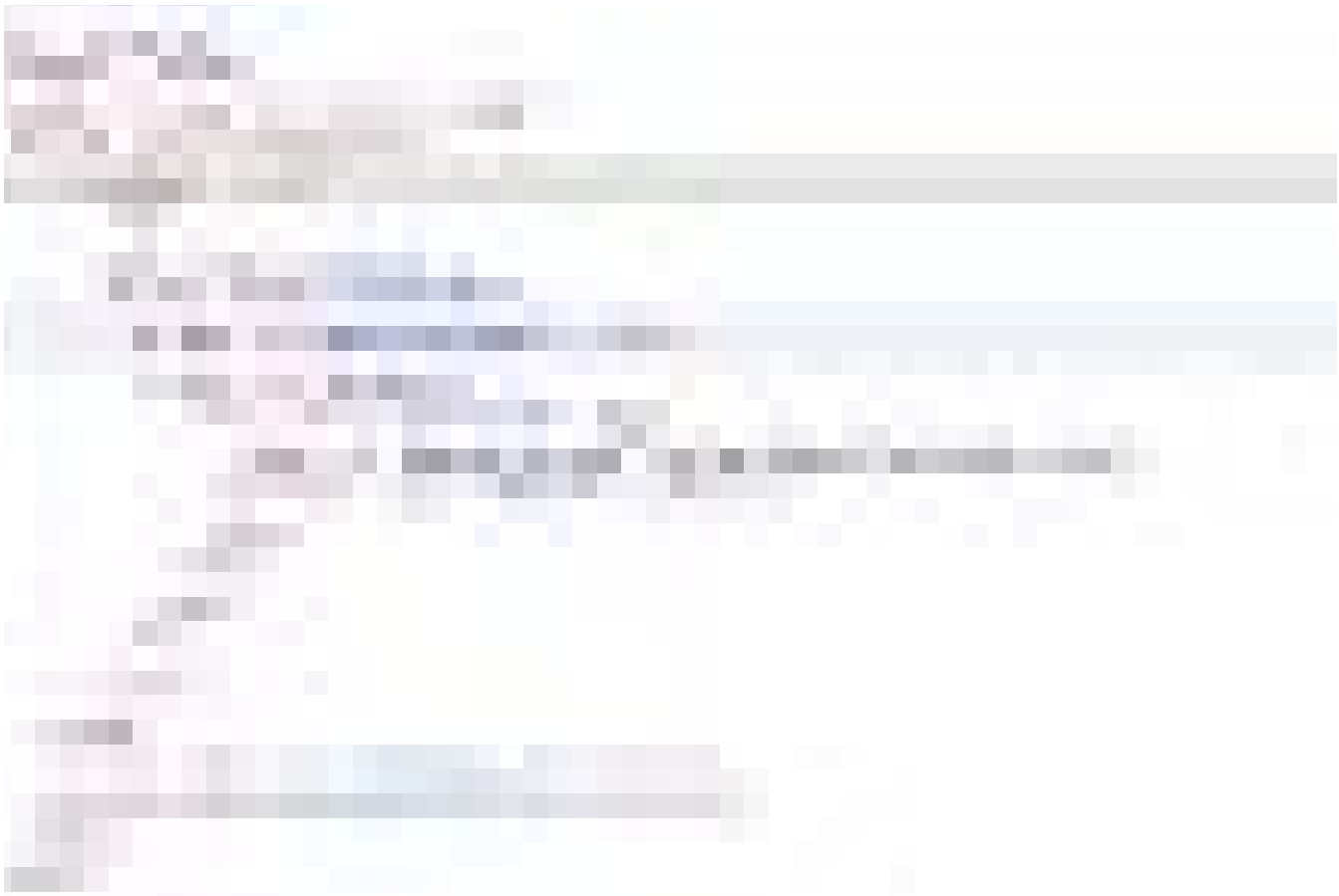
_0x5195.forEach(e => {console.log(e)}) → This will list out all the values inside the array, and found the parameter its requiring → "**password=**" 🙃 *(Or.. simply copy* ***_0x5195*** *and paste inside the developer console and press enter… you will see all the values and index numbers)*

Tried below payload straight away..

https://challenge-0921.intigriti.io/challenge/manager.html?password=%3Cscript%3Ealert(1)%3C/script%3E

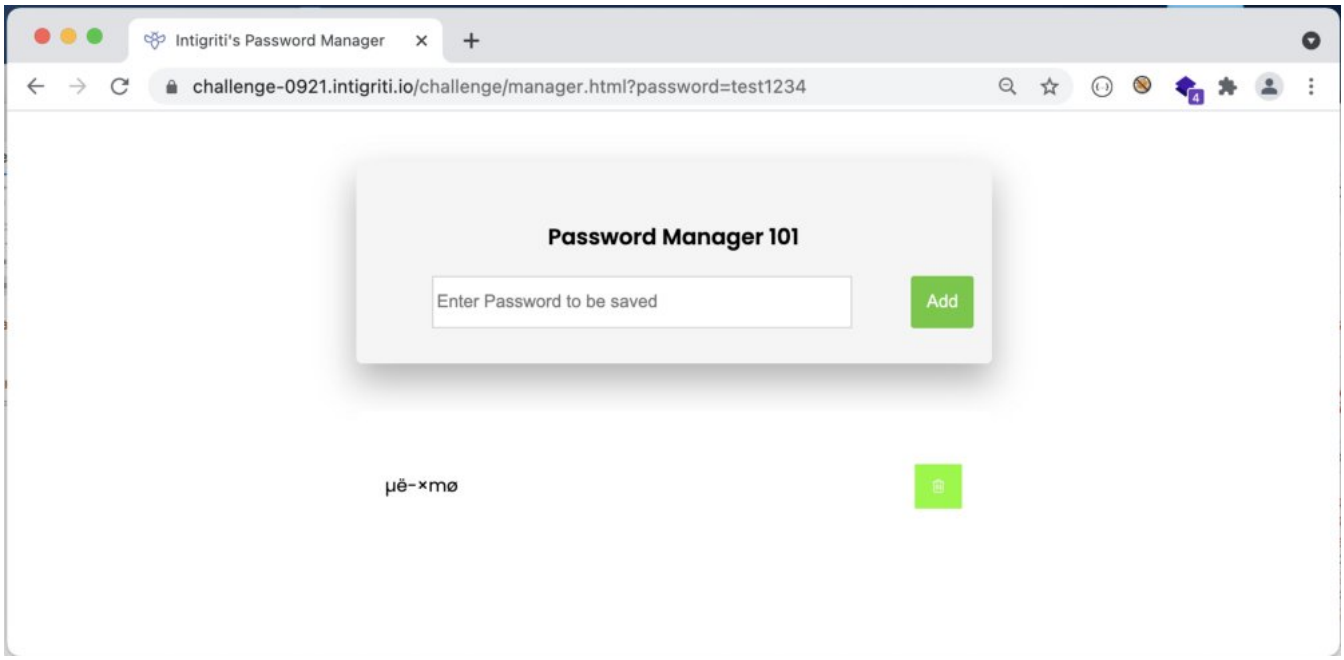


Ohh 😵 something being added “amsterdam_coffeeshop”.



Provided ***<script>alert(1)</script>***… and where is my payload?? 😬 Inspected the HTML no payload, □ what is this “amsterdam_coffeeshop”

. . .



At this point I thought some filter here replacing the payload with ***“amsterdam_coffeeshop”***, then decided to put some plain text like “test1234” to see this get reflected anywhere in the HTML, but something wired happened ***“amsterdam_coffeeshop”*** got replaced with **“µë-×mø”.** I get this wired characters when enter value length is multiple of 4 (4,8,12,16).. actually there is code to check this.

```
122   const m = k[_0x5195[0xe * -0x157 + 0x1b7 * -0x4 + 0x199f]];  m: 9  k:
123   if (!m || m % (-0x27f + -0x1a1 + 0x424) !== -0x73f + 0x7d + 0xa * 0xad
```

Code Line 123

Another fun fact is, by accident remove one character from ***“test1234”*** → ***“test123”*** then again ***“amsterdam_coffeeshop”*** appeared. Then realised ok the values enter has some affects how content which get added to the page and wanted to find where all this

happening in the code?

Also noticed whenever *"amsterdam_coffeeshop"* get added to the page in developer console there was a message ***"Try Harder"*** (this index 701: "try hard" in array *_0x5195*).

Time for deobfuscating the whole code, again google to rescue. http://jsnice.org https://www.dcode.fr/javascript-unobfuscator

Started to see some javascript specific function but still, the code doesn't make sense. Here comes the developer skill, DEBUGGING. Placed some breakpoints, on the main array ***_0x5195*** and start to look at values being added to variables. Also copied all the values from Array ***_0x5195*** a text file incase needs to use later, indeed it helped, you will see below how!

Index 700 looks familiar at this point

• • •

After spending some time debugging with breakpoints found a javascript function that responsible to convert values provide to “password” parameter to this wired **“µë-×mø”** characters. It was on line 1424

```
1409    document[_0x5195[-0x12b3 * -0x1 + 0x4 * 0x1e3 + -0x19be]](_0x5195[-0x21e5 + 0x2 * -0x3cb + -0x107 * -0x2b])[_0x5195[0x29 * 0x61 + 0x1f
1410        function g3(c, d) {
1411            return fp(d, c - -0x13);
1412        }
1413        if (document[_0x5195[-0x14df * -0x1 + 0x3a * 0x7f + -0x3124]](_0x5195[0x1 * 0x1f91 + 0x27d * -0x9 + 0x679 * -0x1])[_0x5195[0x3 * 0
1414            var k = {};
1415            k[g3(0x79, 0x82)] = _0x5195[0x165e + -0x203 + 0x11a7 * -0x1];
1416            k[g3(0xb7, 0x198)] = _0x5195[-0x1 * 0xaf + -0x5 * -0x4a3 + 0x1 * -0x13cb];
1417            k[g3(-0x101, -0x97)] = _0x5195[0x83 * 0x22 + -0x1df2 + -0x9 * -0x1b2];
1418            k[g3(-0x125, 0x73)] = _0x5195[-0x41 * 0x2f + 0x4fa + 0x9ac];
1419            swal(k);
1420        } else {
1421            if (window[_0x5195[0x3 * 0xacf + -0x3 * 0x2b3 + -0x15a9]][_0x5195[0x24f8 + 0x126f + 0xa9 * -0x53]][_0x5195[-0xb * -0x38c + -0x
1422                var l = i(_0x5195[-0x1fb * -0x11 + 0x14 * 0xb1 + -0x2cc4])[_0x5195[0x5 * 0x673 + 0x48e + -0x2213]](_0x5195[0x1 * -0xc73 +
1423                if (e(l) === !![]) {
1424                    var m = atob(l);
1425                } else {
1426                    var m = _0x5195[0x3 * -0x851 + 0x1302 + 0x8ad];
1427                    console[_0x5195[-0x3 * 0xb3 + 0x96e + -0x74c]](_0x5195[-0x393 + -0x11ab + 0x17fb]);
1428                };
1429                document[_0x5195[0xf07 + -0x3 * 0x9fd + 0xf71]](_0x5195[0xa28 + -0x1 * -0x21af + -0x2919])[_0x5195[-0x10 * 0xca + 0x1196 *
1430                var n = document[_0x5195[-0x1f78 + -0x371 + 0x1f * 0x125]](_0x5195[0x4d1 * 0x2 + 0x6 * -0x446 + 0x12c3]);
1431                for (var o = 0x498 + 0x221 * -0x11 + -0x1 * -0x1f99; o < n[_0x5195[0x5 * 0x79d + -0x85f * 0x1 + -0x1db1 * 0x1]]; o++) {
1432                    n[o][_0x5195[0x15dd + 0x248 + -0x1574]] = function() {
1433                        this[_0x5195[-0xe83 + 0xc05 + 0x2f1]][_0x5195[0x1261 + -0x2393 * -0x1 + -0x3332]]();
1434                    };
```

atob() what this function/method does? Turn to be Game Changer

> *When searching for Javascript use https://developer.mozilla.org and in google search bar* **javascript atob MDN**

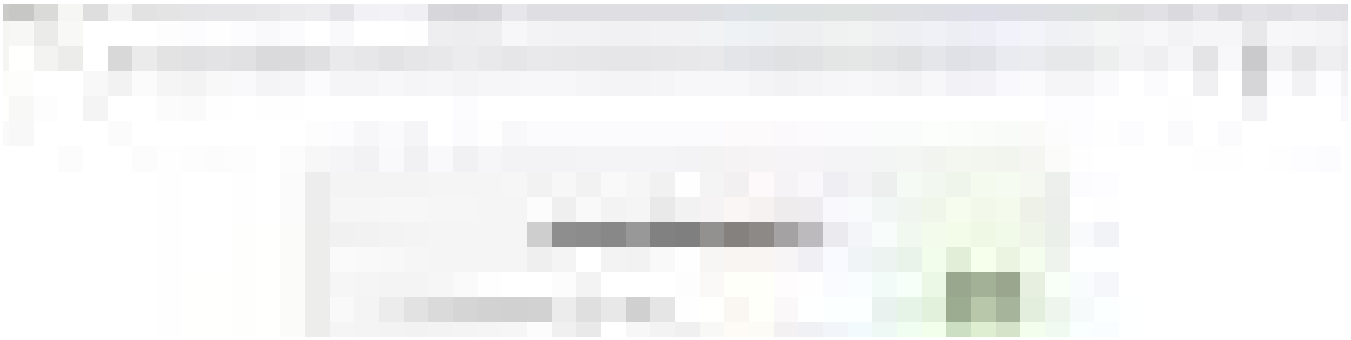
Google time again for “**atob()**”. This function is to decode base64 encoded values. YES YES YES, time for CyberCehf

atob() - Web APIs | MDN

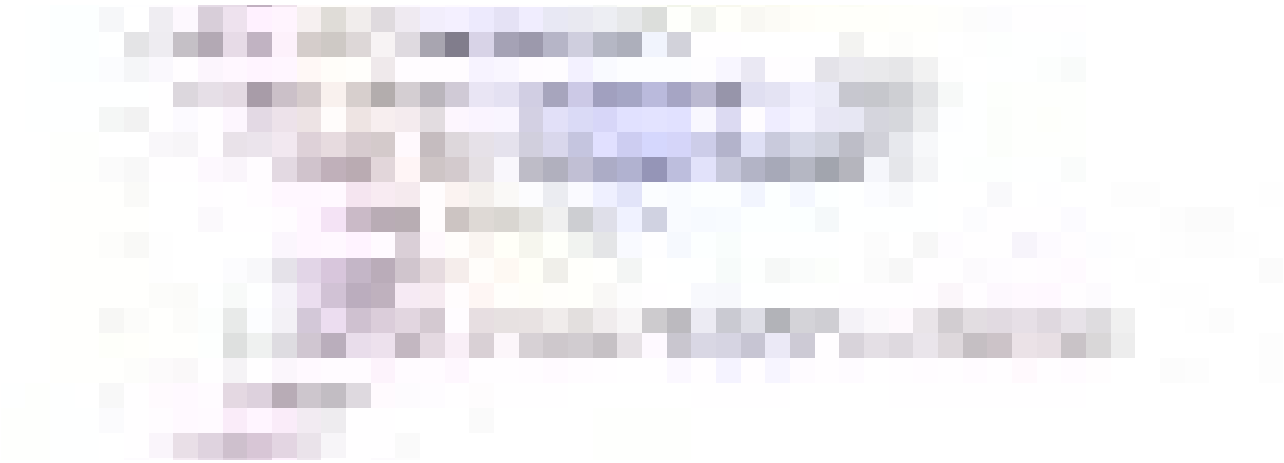
The atob() function decodes a string of data which has been encoded using Base64 encoding. You can use the btoa()...

developer.mozilla.org

Crafted payloads with base64 encoded (<img/src=x onerror=alert(1)>) aaaannndd.. No POPUP.. what is going on 😂

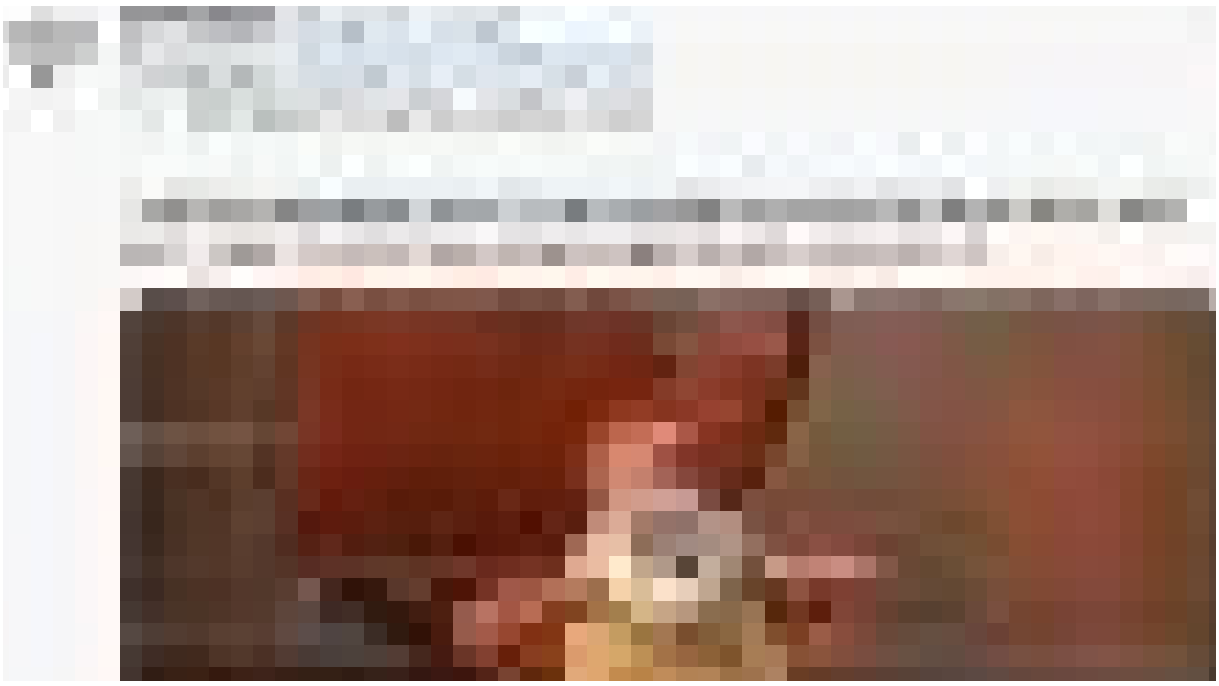


Let's check the HTML,



Where is the rest of the payload ?

After trying many base64 payloads & checking the HTML noticed that all the ***events*** gets cuts off. So there is a ***Filters/Sanitizer*** being used in the code as we cannot see any other Javascript library file. Same time second Tip being published on twitter by Intigriti, let's have a look. (This tip helped me to crack the ***Filters/Sanitizer***)



They are hinting about MEMORY and “AntIH4Ck3RC0D3zzzzzzzzz”, Ohh I have seen this in that magic array ***_0x5195*** index 681 (**681: “AntIH4Ck3RC0D3zzzzzzzzz”**)

And “Memory” make sense, since code uses lots of Closures & IIFE

Closures - JavaScript | MDN

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the...

developer.mozilla.org

IIFE - MDN Web Docs Glossary: Definitions of Web-related terms | MDN

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. The name...

developer.mozilla.org

So something must be leaking in the Global Scope… Yap it is…. (see the below image)

Hey Hello AntIH4Ck3RC0D3zzzzzzzzz

> *It is always good the keep an eye on* ***Scope*** *tab when analysing/debugging unclear code, you can see variables and their values being assigned to them. I believe most of the hackers miss this part.* So next time do not miss this area.

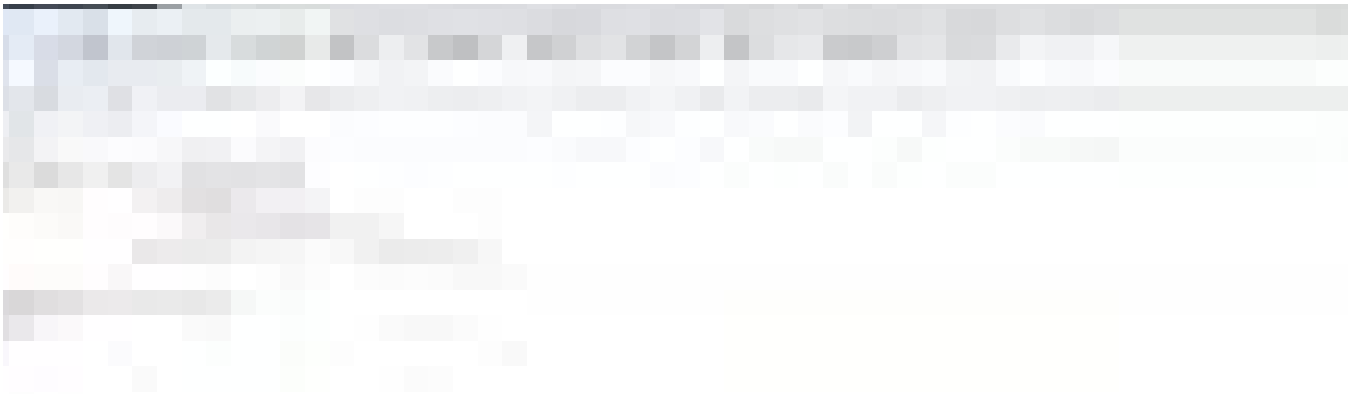
As you can see **AntIH4Ck3RC0D3zzzzzzzzz** starts with capital "A". This indicated its a class.. again seen index 668: "constructor" in array ***_0x5195.***

> In many programming languages constructor method is a special method of a class for creating and initializing an object of that class.

So what is this AntIH4Ck3RC0D3zzzzzzzzz class do… It's obvious this is the Filter/sanitizer library, why? look at the image above it has version number 2.0.8, try running below code in the developer console.

AntIH4Ck3RC0D3zzzzzzzzz.MCAST_MSFILTER

AntIH4Ck3RC0D3zzzzzzzzz.version



Developer console

So what we do with it.. come on we are hackers.. Google dork it ***"2.0.8" sanitizer***

https://www.google.com/search?client=safari&rls=en&q=%222.0.8%22+sanitizer&ie=UTF-8&oe=UTF-8

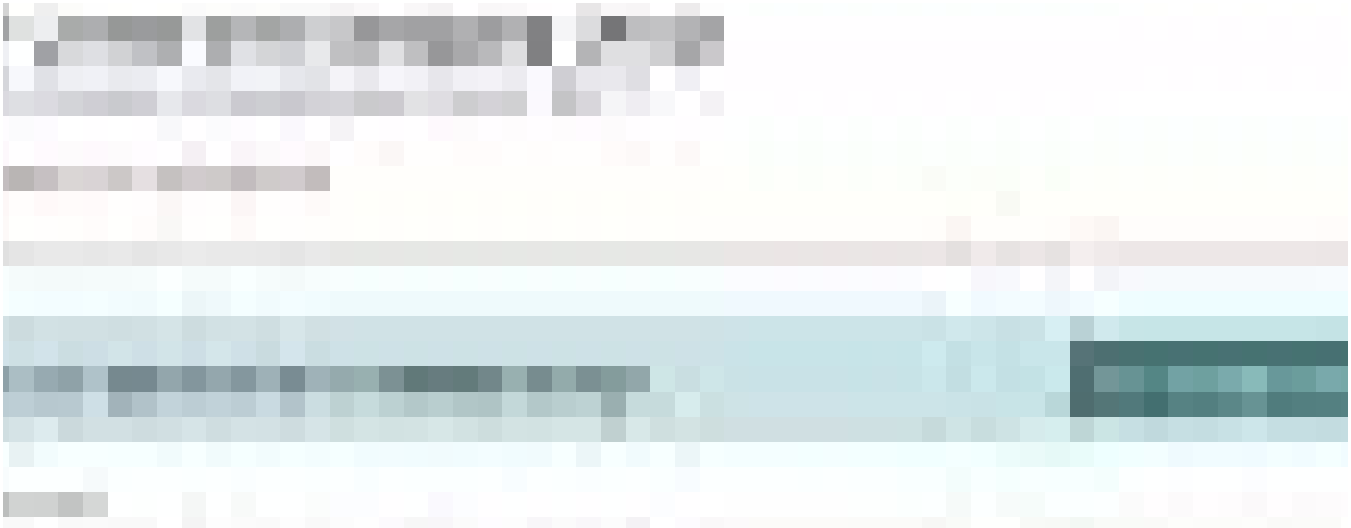
Hey Hello DomPurify

https://www.npmjs.com/package/dompurify/v/2.0.8 → Same version as AntIH4Ck3RC0D3zzzzzzzzz.version, GOOD GOOD… Lets check for vulnerabilities in this library.

Snyk - dompurify@2.0.8 vulnerabilities

Learn more about vulnerabilities in dompurify@2.0.82.3.1, DOMPurify is a DOM-only, super-fast, uber-tolerant XSS...

snyk.io



So now we know this version is vulnerable for XSS. Let's ask our good buddy Google…

dompurify xss bypass - Google Search

Edit description

www.google.com

This awesome article from portswigger.net saved my day ,

Bypassing DOMPurify again with mutation XSS

After seeing Michał Bentkowski's DOMPurify bypass and the resulting patch, I was inspired to try and crack the patched...

portswigger.net

DOMPurify payload from PortSwigger

Payload time baby….. 😎

Raw Payload:

<math><mtext><table><mglyph><style><! — </style><img title=" — ></mglyph><img	src=1	onerror=alert(document.domain)>" >

CyberChef — To Base64 → URL Encoded

Why do we need URL Encode after base64 □

Since base64 strings can contain the “+”, “=” and “/” characters. which could alter the meaning of your data.

Payload with Base64 Encoded+URL encoded:

“https://challenge-0921.intigriti.io/challenge/manager.html?password=PG1hdGg+PG10ZXh0Pjx0YWJsZT48bWdseXBoPjxzdHlsZT48IS0tPC9zdHlsZT48aW1nIHRpdGxlPSItLSZndDsmbHQ7L21nbHlwaCZndDsmbHQ7aW1nJlRhYjtzcmM9MSZUYWI7b25lcnJvcj1hbGVydChkb2N1bWVudC5kb21haW4pJmd0OyI+”

✹ Booom !!! Finally !!!

. . .

Thanks for reading, hope you learned something from this post. Stay Safe !!!

> ***Make sure you give this post some 🖐 and my blog a follow if you enjoy this post and want to see more.*** □ □ □□

. . .

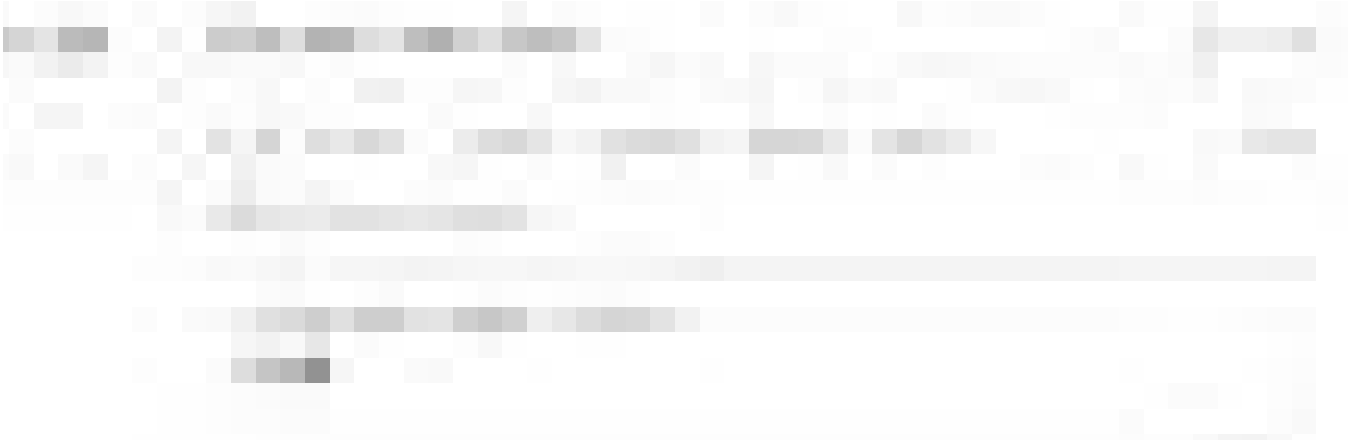
BOUNS READ:

How I did reverse engineering *manually* some part of the code to get a feel of what these codes are doing. Lets do this simple console.log() part…

```
1425            } else {
1426                var m = _0x5195[0x3 * -0x851 + 0x1302 + 0x8ad];
1427                console[_0x5195[-0x3 * 0xb3 + 0x96e + -0x74c]](_0x5195[-0x393 + -0x11ab + 0x17fb]);
1428            };
1429            document[_0x5195[0xf07 + -0x3 * 0x9fd + 0xf71]](_0x5195[0xa28 + -0x1 * -0x21af + -0x2919])[_0x5195[-0x10 * 0xca + 0x1196
1430            var n = document[_0x5195[-0x1f78 + -0x371 + 0x1f * 0x125]](_0x5195[0x4d1 * 0x2 + 0x6 * -0x446 + 0x12c3]);
1431            for (var o = 0x498 + 0x221 * -0x11 + -0x1 * -0x1f99; o < n[_0x5195[0x5 * 0x79d + -0x85f * 0x1 + -0x1db1 * 0x1]]; o++) {
1432                n[o][_0x5195[0x15dd + 0x248 + -0x1574]] = function() {
1433                    this[_0x5195[-0xe83 + 0xc05 + 0x2f1]][_0x5195[0x1261 + -0x2393 * -0x1 + -0x3332]]();
1434                };
1435            };
1436            var p = document[_0x5195[-0x1 * 0x1723 + 0x5 * 0x494 + -0x13 * -0xb]](_0x5195[-0x1c07 + 0x18cc + 0x76 * 0xd]);
1437            for (var o = -0x70 * -0x1a + -0x13 * -0x12b + -0x2191; o < p[_0x5195[0x2321 + 0x6b * -0xb + -0x1e87]]; o++) {
1438                p[o][_0x5195[-0x1f63 * 0x1 + 0x2631 + -0xd * 0x51]] = function() {
1439                    this[_0x5195[-0x8eb + 0x1b0 + 0xa01]][_0x5195[-0x1d3c + 0x20da * -0x1 + 0x40db]](_0x5195[-0x1333 + 0x18a4 + -0x5
1440                };
1441            };
1442            document[_0x5195[-0x115a + -0x1 * -0x19e5 + -0x80a]](_0x5195[-0x2 * -0xcab + 0xb9 * 0x2e + -0x37e1])[_0x5195[-0x12e1 + -
1443        } else {
1444            var q = {};
1445            q[g3(0x79, 0x1f2)] = _0x5195[-0x2358 + -0x16b5 + 0x3cc1];
1446            q[g3(0xb7, 0x203)] = _0x5195[-0x186d + -0xd95 + 0xc5 * 0x35];
1447            q[g3(-0x101, -0x29b)] = _0x5195[0x25 * -0x32 + 0x569 * -0x1 + 0xf59];
1448            q[g3(-0x125, 0x56)] = _0x5195[-0x1651 + 0x2330 + -0xa28];
1449            swal(q);
```

console[_0x5195[-0x3 * 0xb3 + 0x96e + -0x74c]](_0x5195[-0x393 + -0x11ab + 0x17fb]);

Step 1: We know _0x5195 is the array.. so lets copy [-0x3 * 0xb3 + 0x96e + -0x74c] and paste in google…



Step 2: Take 0x9 and paste in google and hit search (Hexadecimal 0x9 = 9)

So what is “9” for us in this challenge.. it is the index 9 of the array _0x5195, so let’s have a look index 9 then *(that’s why copied all the values to a text file from array _0x5195 as mentioned earlier)*

index 9 is “log”, let’s replace it *(Note I am replacing these in my code editor)*

console.**log**(_0x5195[-0x393 + -0x11ab + 0x17fb]);

Lets move to second part.. **[-0x393 + -0x11ab + 0x17fb]**, same steps again, copy to google and the see the result,

It’s **0x2BD**..Copy this value, paste again & search..

You can click on “0x2BD” to decimal link to see the value or you can see “701” bit below. 1

Let’s see what is index 701 in array _0x5195… It’s “try harder”… Let’s put this in…

console.log(“try harder”) 😎

This is how i did it..

Simpler way of doing this...

Just copy and paste into developer console and press enter

console[_0x5195[-0x3 * 0xb3 + 0x96e + -0x74c]](_0x5195[-0x393 + -0x11ab + 0x17fb])

Bug Bounty | Xss Filter Bypass | Xss Bypass | Infosec | Intigriti