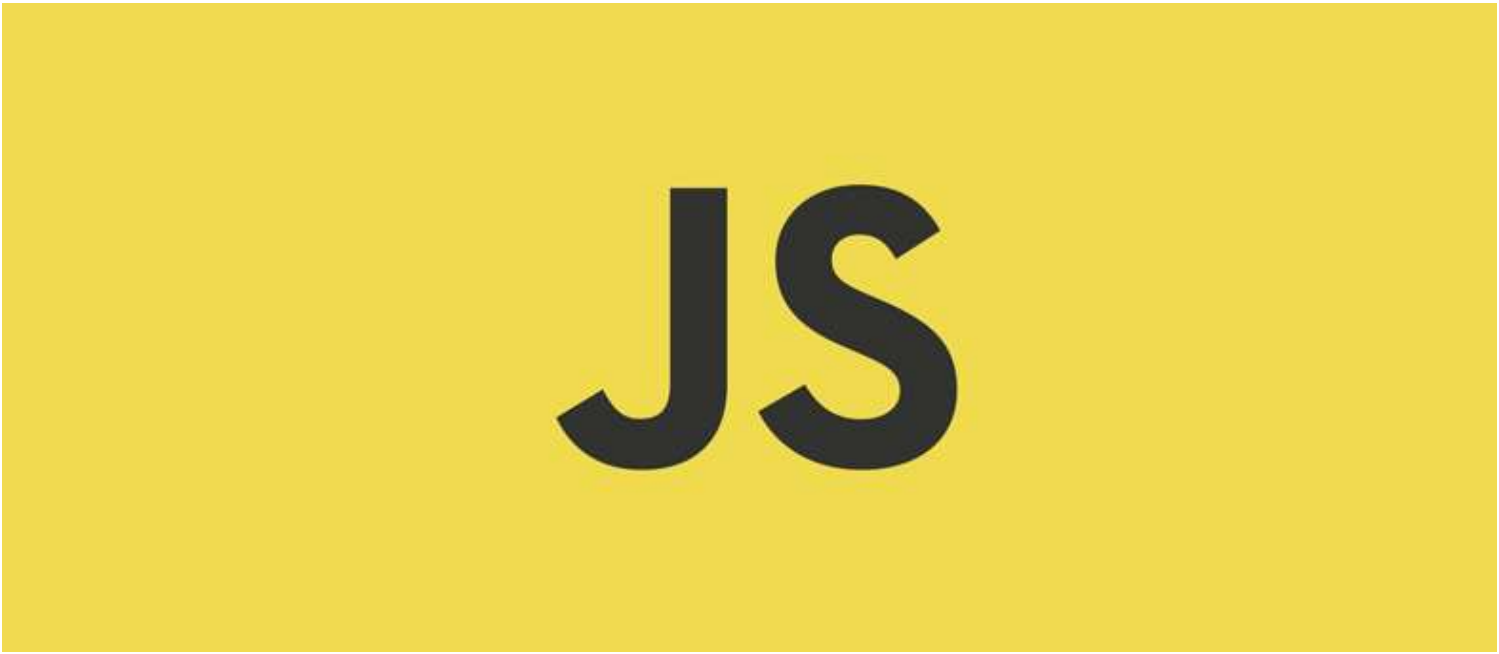
Published in The happy lone guy

Nguyễn Việt Hưng Follow

Dec 22, 2017 · 13 min read · Listen

Save

Breaking down one of the coolest JavaScript obfuscators



Secure your JS code is one of the most important tasks when you have to expose your code to users. That is when JavaScript obfuscator like aaencode, comes to help. Today I'm going to analyze the obfuscated code to know more about how the obfuscated code is executed and in the same time, achieve more knowledge about JS data type.

What is aaencode

You may be unfamiliar with this name, so let me show you what you have missed. The obfuscator named "aaencode" is one of the coolest JS obfuscators created by Yosuke Hasegawa. Unlike other obfuscators, while other obfuscators obfuscated code give you the sense of a solveable code, aaencode make it so hard to read that when you look at it, you don't even think that is a working JS code.

How it looks in aaencode

```
ﾟωﾟﾉ= /｀ｍ´）ﾉ ~┻━┻   //*´∇｀*/ ['_']; o=(ﾟｰﾟ)  =_=3; c=(ﾟΘﾟ) =(ﾟｰﾟ)-(ﾟｰﾟ); (ﾟДﾟ) =
(ﾟΘﾟ)= (o^_^o)/ (o^_^o);(ﾟДﾟ)={ﾟΘﾟ: '_' ,ﾟωﾟﾉ : ((ﾟωﾟﾉ==3) +'_') [ﾟΘﾟ] ,ﾟｰﾟﾉ :(ﾟωﾟﾉ+
'_')[o^_^o -(ﾟΘﾟ)] ,ﾟДﾟﾉ:((ﾟｰﾟ==3) +'_')[ﾟｰﾟ] }; (ﾟДﾟ) [ﾟΘﾟ] =((ﾟωﾟﾉ==3) +'_')
[c^_^o];(ﾟДﾟ) ['c'] = ((ﾟДﾟ)+'_') [ (ﾟｰﾟ)+(ﾟｰﾟ)-(ﾟΘﾟ) ];(ﾟДﾟ) ['o'] = ((ﾟДﾟ)+'_') [ﾟ
Θﾟ];(ﾟoﾟ)=(ﾟДﾟ) ['c']+(ﾟДﾟ) ['o']+(ﾟωﾟﾉ +'_')[ﾟΘﾟ]+ ((ﾟωﾟﾉ==3) +'_') [ﾟｰﾟ] + ((ﾟДﾟ)
+'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ ((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+((ﾟｰﾟ==3) +'_') [(ﾟｰﾟ) - (ﾟΘﾟ)]+(ﾟДﾟ)
['c']+((ﾟДﾟ)+'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ (ﾟДﾟ) ['o']+((ﾟｰﾟ==3) +'_') [ﾟΘﾟ];(ﾟДﾟ) ['_'] =
(o^_^o) [ﾟoﾟ] [ﾟoﾟ];(ﾟεﾟ)=((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+ (ﾟДﾟ) .ﾟДﾟﾉ+((ﾟДﾟ)+'_') [(ﾟｰﾟ) + (ﾟ
ｰﾟ)]+((ﾟｰﾟ==3) +'_') [o^_^o -ﾟΘﾟ]+((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+ (ﾟωﾟﾉ +'_') [ﾟΘﾟ]; (ﾟｰﾟ)+=(ﾟ
Θﾟ); (ﾟДﾟ)[ﾟεﾟ]='\\'; (ﾟДﾟ).ﾟΘﾟﾉ=(ﾟДﾟ+ ﾟｰﾟ)[o^_^o -(ﾟΘﾟ)];(oﾟｰﾟo)=(ﾟωﾟﾉ +'_')[c^_^o];(ﾟ
Дﾟ) [ﾟoﾟ]='\"';(ﾟДﾟ) ['_'] ( (ﾟДﾟ) ['_'] (ﾟε+(ﾟДﾟ)[ﾟoﾟ]+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ (ﾟΘﾟ)+
(ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟ
εﾟ]+(ﾟΘﾟ)+ ((o^_^o) +(o^_^o))+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((o^_^o) +
(o^_^o))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟΘﾟ))+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟｰﾟ)+ ((o^_^o) - (ﾟ
Θﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟ
Θﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟｰﾟ)+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟΘﾟ)+ (ﾟДﾟ)[ﾟoﾟ])
(ﾟΘﾟ)) ('_');
```

Believe or not, the code above works perfectly, you can try execute it in the browser's console and get familiar with it as that piece of obfuscated code will about to be broke down.

Let beautify our cinderella

To increase readability, I used an extremely useful website called jsbeautifier to somewhat deobfuscate the code. After using jsbeautifier, there was only a mild improvement in readability as all of the variables are UTF8 character.

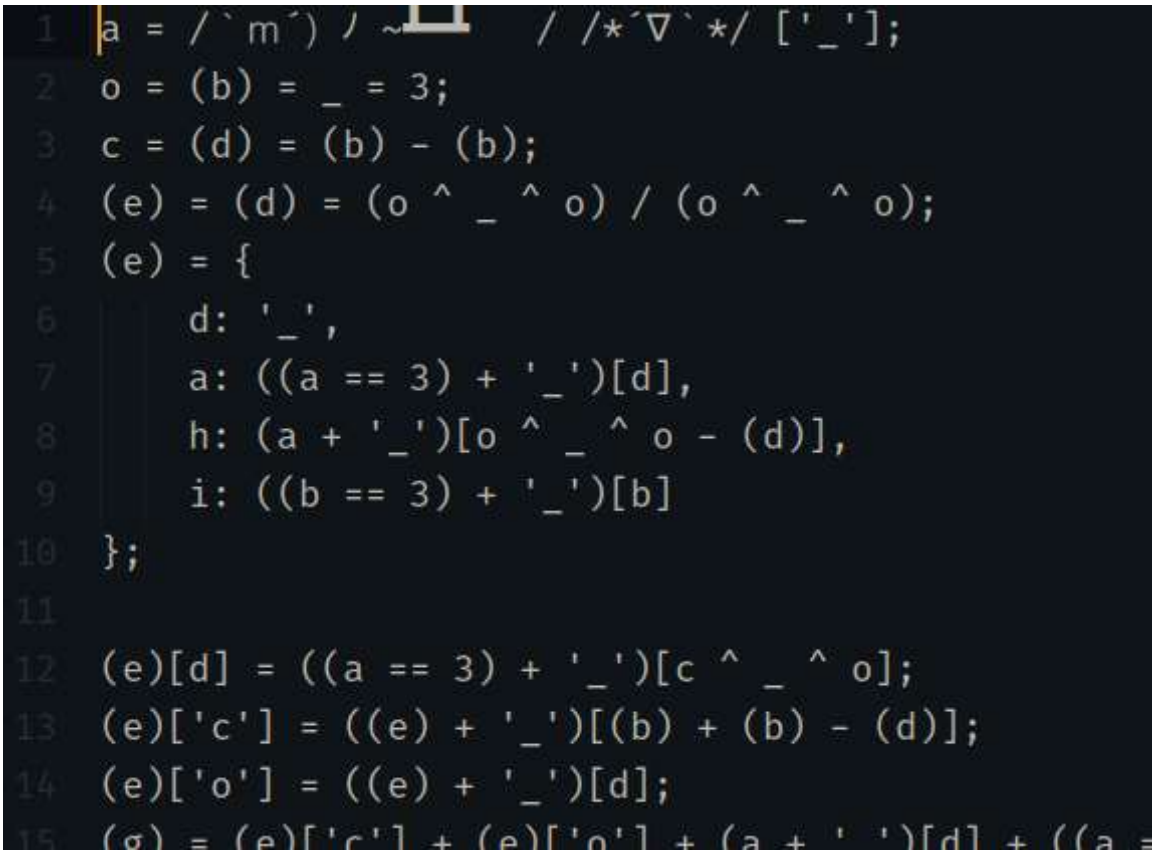
```
ﾟωﾟﾉ = /｀ｍ´）ﾉ ~┻━┻   / /*´∇｀*/ ['_'];
o = (ﾟｰﾟ) =_= 3;
c = (ﾟΘﾟ) = (ﾟｰﾟ) - (ﾟｰﾟ);
(ﾟДﾟ) = (ﾟΘﾟ) = (o^_^o) / (o^_^o);
(ﾟДﾟ) = {
    ﾟΘﾟ: '_',
    ﾟωﾟﾉ: ((ﾟωﾟﾉ == 3) + '_')[ﾟΘﾟ],
    ﾟｰﾟﾉ: (ﾟωﾟﾉ + '_')[o^_^o - (ﾟΘﾟ)],
    ﾟДﾟﾉ: ((ﾟｰﾟ == 3) + '_')[ﾟｰﾟ]
};
(ﾟДﾟ)[ﾟΘﾟ] = ((ﾟωﾟﾉ == 3) + '_')[c^_^o];
(ﾟДﾟ)['c'] = ((ﾟДﾟ) + '_')[(ﾟｰﾟ) + (ﾟｰﾟ) - (ﾟΘﾟ)];
(ﾟДﾟ)['o'] = ((ﾟДﾟ) + '_')[ﾟΘﾟ];
(ﾟoﾟ) = (ﾟДﾟ)['c'] + (ﾟДﾟ)['o'] + (ﾟωﾟﾉ + '_')[ﾟΘﾟ] + ((ﾟωﾟﾉ == 3) + '_')[ﾟｰﾟ] + ((ﾟДﾟ) + '_')[(ﾟｰﾟ
(ﾟДﾟ)['_'] = (o^_^o)[ﾟoﾟ][ﾟoﾟ];
(ﾟεﾟ) = ((ﾟｰﾟ == 3) + '_')[ﾟΘﾟ] + (ﾟДﾟ).ﾟДﾟﾉ + ((ﾟДﾟ) + '_')[(ﾟｰﾟ) + (ﾟｰﾟ)] + ((ﾟｰﾟ == 3) + '_')[o^
(ﾟｰﾟ) += (ﾟΘﾟ);
(ﾟДﾟ)[ﾟεﾟ] = '\\';
(ﾟДﾟ).ﾟΘﾟﾉ = (ﾟДﾟ + ﾟｰﾟ)[o^_^o - (ﾟΘﾟ)];
(oﾟｰﾟo) = (ﾟωﾟﾉ + '_')[c^_^o];
(ﾟДﾟ)[ﾟoﾟ] = '\"';
(ﾟДﾟ)['_']((ﾟДﾟ)['_'](ﾟε + (ﾟДﾟ)[ﾟoﾟ] + (ﾟДﾟ)[ﾟεﾟ] + (ﾟΘﾟ) + (ﾟｰﾟ) + (ﾟΘﾟ) + (ﾟДﾟ)[ﾟεﾟ] + (ﾟΘﾟ) + (
```

The result after using jsbeautifier

JS.

```
var ಠ_ಠ = "I'm done with JS";
alert(ಠ_ಠ);
```

So how do we solve this problem? There is only 1 way, is to replace all the UTF8 variable names with English alphabet letters. This process will take quite some time but with the replace all feature of sublime text I was able to finish the task in just a few minutes.



```
a = /`m´) / ~┻┻   / /*´∇`*/ ['_'];
o = (b) = _ = 3;
c = (d) = (b) - (b);
(e) = (d) = (o ^ _ ^ o) / (o ^ _ ^ o);
(e) = {
    d: '_',
    a: ((a == 3) + '_')[d],
    h: (a + '_')[o ^ _ ^ o - (d)],
    i: ((b == 3) + '_')[b]
};

(e)[d] = ((a == 3) + '_')[c ^ _ ^ o];
(e)['c'] = ((e) + '_')[(b) + (b) - (d)];
(e)['o'] = ((e) + '_')[d];
(g) = (e)['c'] + (e)['o'] + (a + ' ')[d] + ((a =
```

Deeper into the world of code

Let's first look at the first 10 lines.

```
a = /`m´) / ~┻┻   / /*´∇`*/ ['_'];
o = (b) = _ = 3;
c = (d) = (b) - (b);
(e) = (d) = (o ^ _ ^ o) / (o ^ _ ^ o);
(e) = {
    d: '_',
    a: ((a == 3) + '_')[d],
    h: (a + '_')[o ^ _ ^ o - (d)],
    i: ((b == 3) + ' ')[b]
```

First of all, we can see that the code is assigning some dark magic characters to the variable “a”. However, despite its weird looking, its value is actually just `undefined` because the value of `a` consists 3 parts

```
/`m´) / ~┻┻   / -> regular expression
/*´∇`*/ -> comment
['_'] -> object property selector
```

So the value of `a` is the value of the property `_` in the regex and because the `_` property is not defined so it will return `undefined` .

Next, the value of `o` , `b` and `_` is set to 3. Afterward, the variables `c` and `d` are created and assigned the value of `b-b` which is `3-3 = 0` . The variable `d` is then shares it value with `e` which is a newly created variable and both of them contain a new value which is smile face divided smile face.

Haha, that actually not a smile face, it’s a combination bettween the `o` and the `_` variable using the xor operator and the divide operator. The actual value of a smile face is.

```
(o ^ _ ^ o) = (3 ^ 3 ^ 3) = 3
```

So we can calculate the value of `e` and `d` is now equals to `3 / 3 = 1` .

Then `e` changed its value to an object, that means only `d` is now carry the value of `1` . The value of e is now easy to resolve as it’s now just basic JavaScript.

```
(e) = {
    d: '_',
    a: ((a == 3) + '_')[d],
    h: (a + '_')[o ^ _ ^ o - (d)],
    i: ((b == 3) + '_')[b]
};
```

Let’s take a look at the `a` property, its value is `((a == 3) + '_')[d]` . Now we all know that `a` is equals to `undefined` so `a==3` will definitely return `false` , then `false` is concatenated with the

Same thing with `h` ,property, we will have

```
(a + '_')[o ^ _ ^ o - (d)] = 'undefined_'[3 - 1] = 'd'
```

and `i` property will possess the value of

```
((b == 3) + '_')[b] = (true + '_')[3] = 'true_'[3] = 'e'
```

Let rewrite our code with the new value that we have successfully calculated.

```
a = undefined;
o = (b) = _ = 3;
c = (d) = 0;
(e) = (d) = 1;
(e) = {
    d: '_',
    a: 'a',
    h: 'd',
    i: 'e'
};
```

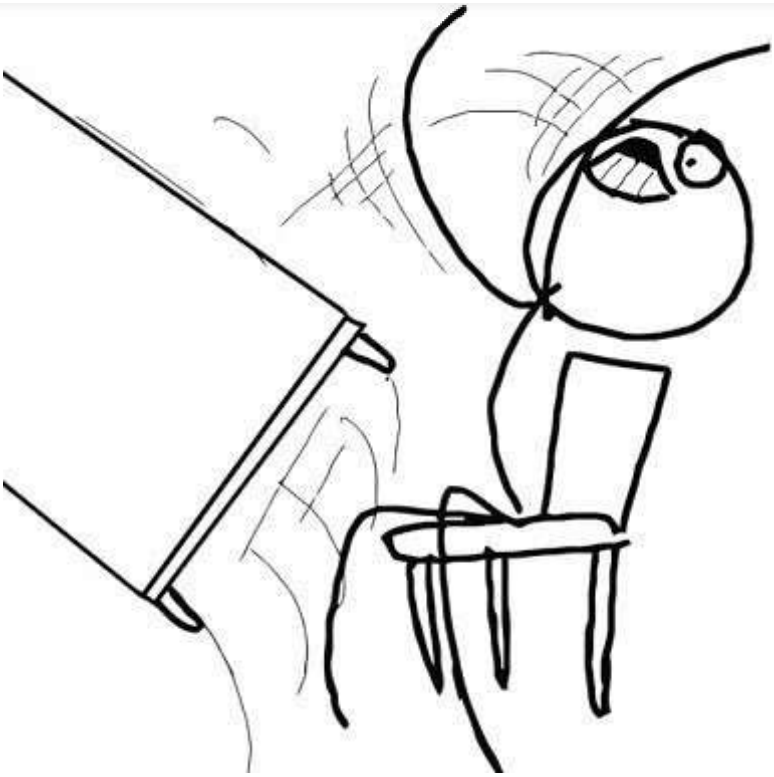
Move on to the next 3 lines.

```
(e)[d] = ((a == 3) + '_')[c ^ _ ^ o];
(e)['c'] = ((e) + '_')[(b) + (b) - (d)];
(e)['o'] = ((e) + '_')[d];
```

The first line in those 3 lines is seem to be assigning a new property to the `e` object variable. Because `d` is equals to `1` and there is no property named `1` in `e` so **JavaScript will create a new property if you assign some value to a undefined property.** That means `e` is now own another property with value of `'false_'[0 ^ 3 ^ 3] = 'f'` .

Same thing will happen with the next 2 lines. But something different appeared in those 2 lines. They concatenating `e` with `'_'` ? How can an object combine with a character? Well, **if you try to concatenate an object with a character or string, JavaScript will not return any error but**



Here is the result of it.

```
> e.toString()
<- "[object Object]"
> e + '_'
<- "[object Object]_"
>
```

Thus, the value of the last 2 lines will be

```
(e)['c'] = ((e) + '_')[(b) + (b) - (d)] = '[object Object]_'[5] = 'c'
(e)['o'] = ((e) + '_')[d] = '[object Object]_'[1] = 'o';
```

Again let rewrite the `e` variable with the newly added property.

```
a = undefined;
o = (b) = _ = 3;
c = (d) = 0;
(e) = (d) = 1;
(e) = {
    d: '_',
    a: 'a',
    h: 'd',
    i: 'e',
    1: 'f',
    c: 'c',
    o: 'o'
};
```

Take a break !

You should now take a break because we have just finished our work with the stage 1 and you won't survive the stage 2 if you don't take a break. Who said JavaScript is easier than calculus?

More complicated code

Now we will move on to the hardest part of our analyzing process. Here is a part of the rest of our obfuscated code and in this section we will analyze the part from the beginning of the code to the line 22.

```
(g) = (e)['c'] + (e)['o'] + (a + '_')[d] + ((a == 3) + '_')[b] + ((e) + '_')[(b) + (b)] + ((b == 3) + '_')
(e)['_'] = (o ^ _ ^ o)[g][g];
(f) = ((b == 3) + '_')[d] + (e).i + ((e) + '_')[(b) + (b)] + ((b == 3) + '_')[o ^ _ ^ o - d] + ((b == 3) +
(b) += (d);
(e)[f] = '\\';
(e).j = (e + b)[o ^ _ ^ o - (d)];
(obo) = (a + '_')[c ^ _ ^ o];
(e)[g] = '\"';

(e)['_']((e)['_'](f + (e)[g] + (e)[f] + (d) + (b) + (d) + (e)[f] + (d) + ((b) + (d)) + (b) + (e)[f] + (d)
```

Let's start with the first line

```
(g) = (e)['c'] + (e)['o'] + (a + '_')[d] + ((a == 3) + '_')[b] + ((e) + '_')
[(b) + (b)] + ((b == 3) + '_')[d] + ((b == 3) + '_')[(b) - (d)] + (e)['c'] +
```

```
(g) =
 (e)['c'] +
 (e)['o'] +
 (a + '_')[d] +
 ((a == 3) + '_')[b] +
 ((e) + '_')[(b) + (b)] +
 ((b == 3) + '_')[d] +
 ((b == 3) + '_')[(b) - (d)] +
 (e)['c'] + ((e) + '_')[(b) + (b)] +
 (e)['o'] + ((b == 3) + '_')[d];
```

Look like this statement is trying to build a string because we can see that the first thing that they concatenate is `e['c']` which is a character `'c'` . Thus, we can replace all the variable and obtain this result.

```
(g) =
 (e)['c'] +                              // 'c'
 (e)['o'] +                              // 'o'
 (a + '_')[d] +                          // 'n'
 ((a == 3) + '_')[b] +                   // 's'
 ((e) + '_')[(b) + (b)] +                // 't'
 ((b == 3) + '_')[d] +                   // 'r'
 ((b == 3) + '_')[(b) - (d)] +           // 'u'
 (e)['c'] +                              // 'c'
 ((e) + '_')[(b) + (b)] +                // 't'
 (e)['o'] +                              // 'o'
 ((b == 3) + '_')[d];                    // 'r'

// 'c' + 'o' + 'n' + 's' + 't' + 'r' + 'u' + 'c' + 't' + 'o' + 'r'
// 'constructor'
```

So this line of code just build the stirng `'constructor'` and then assign it to a new variable called `'g'` .

Move on to the next statement, we have a very short but interesting line of code.

```
(e)['_'] = (o ^ _ ^ o)[g][g];
```

So we know that the property `'_'` is not exists in the object variable `e` so we can conclude that this line of code is creating and assigning a new value to that property. But that's a weird value

```
(e)['_'] = 3['constructor']['constructor']
```

So what's constructor? According to this link, `object['constructor']` will

> *Returns a reference to the `Object` constructor function that created the instance object. Note that the value of this property is a reference to the function itself, not a string containing the function's name. The value is only read-only for primitive values such as `1`, `true` and `"test"`.*

So basically, you can use `object['constructor']` to create a new instance of that object type.

```
> function Book(title) {
      this.title = title;
  }
<- undefined
> b = new Book("math book");
<- ▶Book {title: "math book"}
> c = new b["constructor"]("English book")
<- ▶Book {title: "English book"}
>
```

Now an intriguing question to ask is "what is the constructor of an object constructor?"

Every constructor of a class in JavaScript is a function.

Thus, the constructor of the constructor of a object is a constructor of a function and **every function in JavaScript is a `Function` object**. See the first sentence in the description section in this site to know more.

A `Function` object is a global object and according to MDN web docs

> *The **`Function` constructor** creates a new `Function` object. Calling the constructor directly can create functions dynamically, but suffers from security and performance issues similar to `eval`.*

With some furthur reading, you will discover that this is a bit like the `eval` function except it will create an anonymous function but not to invoke it.

```
> a = new Function("console.log('abc');")
<- f anonymous() {
   console.log('abc');
   }
> a()
   abc
<- undefined
>
```

The a function will only execute if we call it

Back to the main reason why we conduct all of these researchs — our short line of code.

```
(e)['_'] = (o ^ _ ^ o)[g][g];
```

That means the property `_` of object variable `e` is now contains a reference to the `Function` object constructor.

Our next line of code is a bit shorter than the first one but don't worry this is just another string building code.

```
(f) = ((b == 3) + '_')[d] + (e).i + ((e) + '_')[(b) + (b)] + ((b == 3) +
'_')[o ^ _ ^ o - d] + ((b == 3) + '_')[d] + (a + '_')[d];
```

The final result will be

```
(f) =
 ((b == 3) + '_')[d] +                    // 'r'
 (e).i +                                  // 'e'
 ((e) + '_')[(b) + (b)] +                 // 't'
 ((b == 3) + '_')[o ^ _ ^ o - d] +        // 'u'
 ((b == 3) + '_')[d] +                    // 'r'
 (a + '_')[d];                            // 'n'

// 'r' + 'e' + 't' + 'u' + 'r' + 'n'
// 'return'
```

```
(b) += (d);
(e)[f] = '\\';
(e).j = (e + b)[o ^ _ ^ o - (d)];
(obo) = (a + '_')[c ^ _ ^ o];
(e)[g] = '\"';
```

It's clear that the first line will increase the variable `b` by the value of `d` which is 1. Now `b` will hold the value of 4.

The next line will take the string that we obtained in the previous step which is `'return'` to use as a property object `e` and the value of it is a string `'\\'` or more precise, a `\` character.

In this line the code continue to add another property to the object `e` and its value is

```
(e).j = (e + b)[o ^ _ ^ o - (d)]; // -> (e).j = '[object Object]4'[2] = 'b'
```

Next it create a new variable named `obo` to store a character `'u'`.

```
(obo) = (a + '_')[c ^ _ ^ o]; // -> (obo) = 'undefined_'[0] = 'u'
```

Then `e` is continue to be added a new property in this line.

```
(e)[g] = '\"'; // -> (e)['constructor'] = '\"'
```

And that's the end of our stage 2. We can now rewrite our code and here is how it look after rewriting.




```
a = undefined;
o = (b) = _ = 3;
c = (d) = 0;
(e) = (d) = 1;
(e) = {
    d: '_',
    a: 'a',
    h: 'd',
    i: 'e',
    1: 'f',
    c: 'c',
    o: 'o',
    return: '\\',
    _: Function,
    constructor: "\"",
    j: 'b'
};

(g) = 'constructor';
(f) = 'return';
(b) = 4;

(e)['_']((e)['_'](f + (e)[g] + (e)[f] + (d) + (b) + (d
```

Last line standing

We are now at the final stage of our analyzing process. Only 1 last line of code which is also the most crucial line of all.

```
(e)['_']((e)['_'](f + (e)[g] + (e)[f] + (d) + (b) + (d) + (e)[f] + (d) +
((b) + (d)) + (b) + (e)[f] + (d) + (b) + ((b) + (d)) + (e)[f] + (d) + ((o ^
_ ^ o) + (o ^ _ ^ o)) + ((o ^ _ ^ o) - (d)) + (e)[f] + (d) + ((o ^ _ ^ o) +
(o ^ _ ^ o)) + (b) + (e)[f] + ((b) + (d)) + (c ^ _ ^ o) + (e)[f] + (b) + ((o
^ _ ^ o) - (d)) + (e)[f] + (d) + ((b) + (d)) + (c ^ _ ^ o) + (e)[f] + (d) +
((b) + (d)) + (d) + (e)[f] + (b) + ((o ^ _ ^ o) - (d)) + (e)[f] + ((b) +
(d)) + (d) + (e)[g])(d))('_');
```

First, let's separate them into parts.

```
(e)['_'](
 (e)['_'](

  f + (e)[g] + (e)[f] + (d) + (b) + (d) + (e)[f] + (d) + ((b) + (d)) + (b) +
(e)[f] + (d) + (b) + ((b) + (d)) + (e)[f] + (d) + ((o ^ _ ^ o) + (o ^ _ ^
o)) + ((o ^ _ ^ o) - (d)) + (e)[f] + (d) + ((o ^ _ ^ o) + (o ^ _ ^ o)) + (b)
+ (e)[f] + ((b) + (d)) + (c ^ _ ^ o) + (e)[f] + (b) + ((o ^ _ ^ o) - (d)) +
(e)[f] + (d) + ((b) + (d)) + (c ^ _ ^ o) + (e)[f] + (d) + ((b) + (d)) + (d)
+ (e)[f] + (b) + ((o ^ _ ^ o) - (d)) + (e)[f] + ((b) + (d)) + (d) + (e)[g]

  )(d))
('_');
```

As you can see, the `e['_']` is wrapping all the code inside it and `e['_']` is the `Function` object constructor so we know this line is trying to dynamically make a function and execute it. We also know that `Function` object constructor take a string to use it as the function body, so the code within it must be building a string. Using the browser console we can see that the inner code is building the string

```
return"\141\154\145\162\164\50\42\150\151\42\51"
```

```
> f + (e)[g] + (e)[f] + (d) + (b) + (d) + (e)[f] + (d) + ((b) +
  (d)) + (b) + (e)[f] + (d) + (b) + ((b) + (d)) + (e)[f] + (d) +
  ((o ^ _ ^ o) + (o ^ _ ^ o)) + ((o ^ _ ^ o) - (d)) + (e)[f] + (d)
  + ((o ^ _ ^ o) + (o ^ _ ^ o)) + (b) + (e)[f] + ((b) + (d)) + (c
  ^ _ ^ o) + (e)[f] + (b) + ((o ^ _ ^ o) - (d)) + (e)[f] + (d) +
  ((b) + (d)) + (c ^ _ ^ o) + (e)[f] + (d) + ((b) + (d)) + (d) +
  (e)[f] + (b) + ((o ^ _ ^ o) - (d)) + (e)[f] + ((b) + (d)) + (d)
  + (e)[g]
< "return"\141\154\145\162\164\50\42\150\151\42\51""
```

What does `"\141\154\145\162\50\42\150\151\42\51"` means?

It actually is a octal escape string. By pasting it directly to the console we can obtain its true value, which is

```
alert("hi")
```

```
> "\141\154\145\162\164\50\42\150\151\42\51"
< "alert("hi")"
```

Let rebuild the code

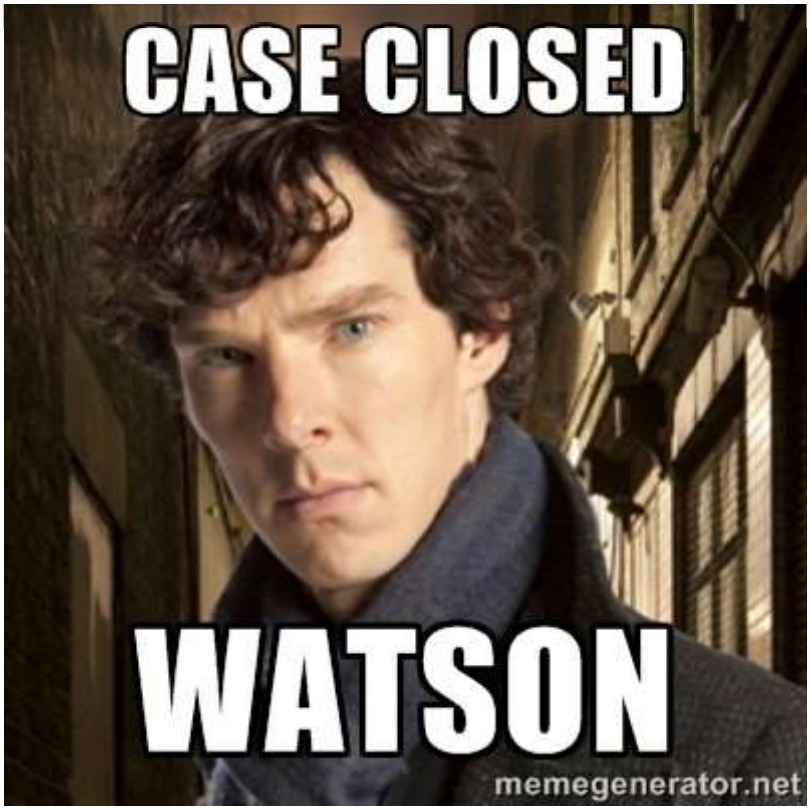
```
(e)['_']( (e)['_']( "return \"alert(\"hi\")\" ) (d)) ('_');
```

We can rewrite the inner function to this

```
(function(){ return "alert(\"hi\")"; })(d)
```

So after the function is built it is then invoked with a parameter `d` and string `alert(\"hi\")` will be passed to the outter `Function` constructor which will make a function and use that string as the function body. After constructing the function it will invoke the function using the `'_'` character and that's how our code is executed.



That's how aaencode JS obfuscator works and it will take a significant amount of time to analyze it, however, nothing is impossible. Also, I want to thank Yosuke Hasegawa for create this amazing obfuscator and I hope after reading this, you guys will find some interesting things to do with JS and please drop a comment if you have any question or compliments :))

Goodbye and Merry Christmas.