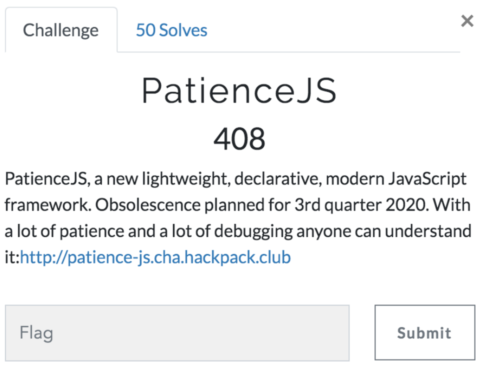
HackPackCTF 2020 PatienceJS

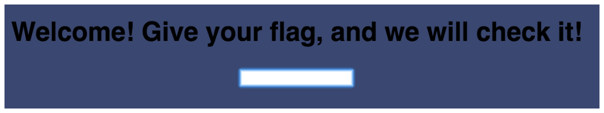
Saturday, March 7, 2020 12:16 PM



PatienceJS

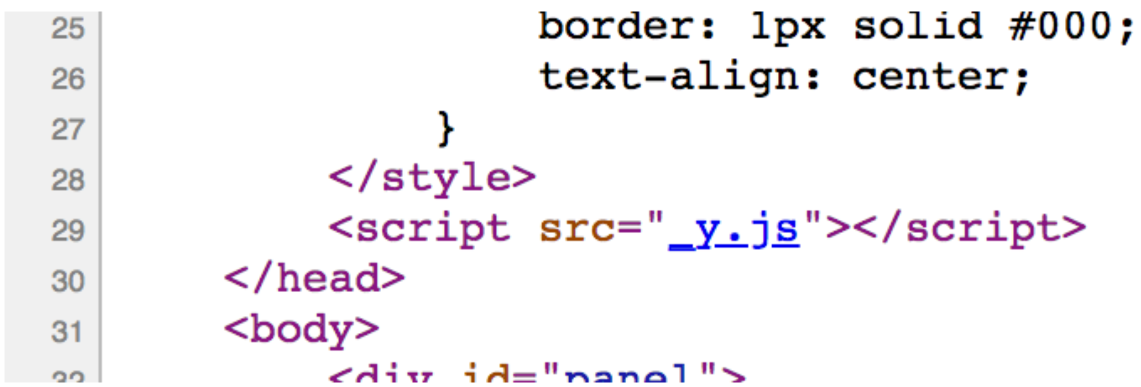
408

PatienceJS, a new lightweight, declarative, modern JavaScript framework. Obsolescence planned for 3rd quarter 2020. With a lot of patience and a lot of debugging anyone can understand it:http://patience-js.cha.hackpack.club



Welcome! Give your flag, and we will check it!

View Source:



```
                border: 1px solid #000;
                text-align: center;
            }
        </style>
        <script src="_y.js"></script>
    </head>
    <body>
        <div id="panel">
```

Clicking the .js link:

```
const _0x_0x5c0a=['join','Congrats!','pop','charCodeAt','getElementById','No!','0xm','from'];(function(_0x9c7ba2,_0x5c0aaf){const _0x11cd2d=function(_0x5629ed){while(--
_0x5629ed){_0x9c7ba2['push'](_0x9c7ba2['shift']());}};_0x11cd2d(++_0x5c0aaf);}(_0x_0x5c0a,0x8d));const _0x_0x11cd=function(_0x9c7ba2,_0x5c0aaf){_0x9c7ba2=_0x9c7ba2-0x0;let
_0x11cd2d=_0x_0x5c0a[_0x9c7ba2];return _0x11cd2d;};function _0x_0x4ab1f1(_0x5315fb){for(let _0x576c1c=0x42;_0x576c1c>0x0;)_0x5315fb['unshift'](_0x5315fb[_0x_0x11cd('0x5')]
()),--_0x576c1c;return _0x5315fb[_0x_0x11cd('0x3')]('');}function _0x_0x2123f4(){return _0x_0x5f4a94()-0x2;}function _0x_0x22a1c0(){return _0x_0x2422b7()+0x2;}function
_0x_0x4cbea8(){return _0x_0x556af7()+0x2;}function _0x_0x3dcfaa(){return _0x_0x20633a()-0x3;}function _0x_0x2c6478(){return _0x_0x26ed44()+0x4;}function _0x_0x420a5d()
{return _0x_0xf6deb3()+0x2;}function _0x_0x556af7(){return _0x_0x466208()-0x3;}function _0x_0x500299(){return _0x_0x3ace04()-0x7;}function _0x_0x46fcc6(){return
_0x_0x4173f1()-0x3;}function _0x_0x27c07f(){return _0x_0x8df3cd()+0x9;}function _0x_0x3a5c14(){return _0x_0x26ed44()+0x2;}function _0x_0x4173f1(){return
_0x_0x20633a()-0x2;}function _0x_0x466208(){return _0x_0x26ed44()-0x2;}function _0x_0x26ed44(){return _0x_0x2422b7()-0x6;}function _0x_0x45ef23(){return
_0x_0x5f4a94()-0xa;}function _0x_0x2422b7(){return _0x_0x3ace04()-0x4;}function _0x_0x5f4a94(){return _0x_0x20633a()+0x6;}function _0x_0x8df3cd(){return
_0x_0xf6deb3()-0xa;}function _0x_0x3ace04(){return _0x_0x20633a()-0x9;}function _0x_0xf6deb3(){return _0x_0x20633a()-0x46;}function _0x_0x20633a(){return 0xa0-
0x29;}function _0x_0x44e371(_0x2de50b){try{let _0x5e0bfc=_0x_0x4ab1f1(Array[_0x_0x11cd('0x2')](_0x2de50b));for(let _0x2d45f5=0x1b;_0x2d45f5>0x0;_0x2d45f5--)
{if(_0x2d45f5===0x17||_0x2d45f5===0xd){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x500299()){return-0x1;}}else if(_0x2d45f5===0x1){if(_0x5e0bfc[_0x_0x11cd('0x6')]
(_0x2d45f5)!==_0x_0x3dcfaa()){return-0x1;}}else if(_0x2d45f5===0x19||_0x2d45f5===0xa){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x3a5c14()){return-0x1;}}else
if(_0x2d45f5===0x3||_0x2d45f5===0x14){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x420a5d()){return-0x1;}}else if(_0x2d45f5===0x16){if(_0x5e0bfc[_0x_0x11cd('0x6')]
(_0x2d45f5)!==_0x_0x4173f1()){return-0x1;}}else if(_0x2d45f5===0x8||_0x2d45f5===0xf){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x8df3cd()){return-0x1;}}else
if(_0x2d45f5===0xc){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x4cbea8()){return-0x1;}}else if(_0x2d45f5===0x15){if(_0x5e0bfc['charCodeAt']
(_0x2d45f5)!==_0x_0x466208()){return-0x1;}}else if(_0x2d45f5===0x10){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x2422b7()){return-0x1;}}else
if(_0x2d45f5===0x0||_0x2d45f5===0x18||_0x2d45f5===0x12||_0x2d45f5===0x4){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x556af7()){return-0x1;}}else if(_0x2d45f5===0x1a)
{if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x27c07f()){return-0x1;}}else if(_0x2d45f5===0x5){if(_0x5e0bfc['charCodeAt'](_0x2d45f5)!==_0x_0x20633a()){return-
0x1;}}else if(_0x2d45f5===0x11){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x45ef23()){return-0x1;}}else if(_0x2d45f5===0x7){if(_0x5e0bfc[_0x_0x11cd('0x6')]
(_0x2d45f5)!==_0x_0x3ace04()){return-0x1;}}else if(_0x2d45f5===0x2){if(_0x5e0bfc['charCodeAt'](_0x2d45f5)!==_0x_0x2c6478()){return-0x1;}}else if(_0x2d45f5===0x1b)
{if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x46fcc6()){return-0x1;}}else if(_0x2d45f5===0x9){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x5f4a94()){return-
0x1;}}else if(_0x2d45f5===0x13){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x26ed44()){return-0x1;}}else if(_0x2d45f5===0xe){if(_0x5e0bfc[_0x_0x11cd('0x6')]
(_0x2d45f5)!==_0x_0x2123f4()){return-0x1;}}else if(_0x2d45f5===0x6){if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0xf6deb3()){return-0x1;}}else if(_0x2d45f5===0xb)
{if(_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5)!==_0x_0x22a1c0()){return-0x1;}}}return 0x0;}catch(_0x1e3b66){}return-0x1;}function _0x_0x5d47c2(){let
_0xb92bef=document[_0x_0x11cd('0x7')]('0xs')['value'],_0x28923e=document[_0x_0x11cd('0x7')](_0x_0x11cd('0x1'));_0x28923e['innerHTML']=_0x_0x44e371(_0xb92bef)?
_0x_0x11cd('0x0'):_0x_0x11cd('0x4');};
```

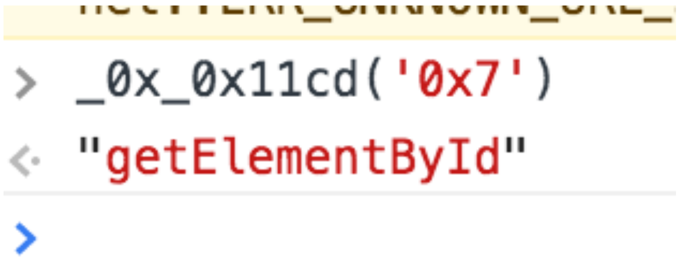
Copied the js code and pasted into an online beautifier.

Then pasted into code.js and opened in Visual Studio Code.

This has some js smarts (goto definition, rename variable/function, etc...).

Noticed lots of calls like this: _0x_0x11cd('0x7')

Used the browser console window to evaluate them:



```
> _0x_0x11cd('0x7')
<· "getElementById"
>
```

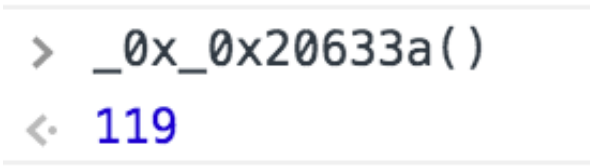
Then I search/replaced the call with the string. Repeated until they were all gone.

Noticed a bunch of functions like this:

```
function _0x_0xf6deb3() {
    return _0x_0x20633a() - 0x46;
}

function _0x_0x20633a() {
    return 0xa0 - 0x29;
}
```

Evaluated them all (20 or so)and replaced calls to them with the numeric value.



```
> _0x_0x20633a()
<· 119
```

The last fn at the bottom is looks like:

```
function _0x_0x5d47c2() {
    let _0xb92bef = document[_0x_0x11cd('0x7')]('0xs')['value'],
        _0x28923e = document[_0x_0x11cd('0x7')](_0x_0x11cd('0x1'));
    _0x28923e['innerHTML'] = _0x_0x44e371(_0xb92bef) ? _0x_0x11cd('0x0') : _0x_0x11cd('0x4');
};
```

With the above search/replace and using VSCode to rename some vars we get:

```
function main() {
    let input = document["getElementById"]('0xs')['value'],
        outputElement = document["getElementById"]("0xm");
    outputElement['innerHTML'] = testInput(input) ? "No!" : "Congrats!";
};
```

Looking at the html there is one element with id="0xs".
It is an input element. This code is getting what you type in that input element and sending it to a fn I renamed as testInput(). If that returns 0, then you win!

The top of the testInput function started out like this:

```
function _0x_0x44e371(_0x2de50b) {
    try {
        let _0x5e0bfc = _0x_0x4ab1f1(Array[_0x_0x11cd('0x2')](_0x2de50b));
```

This calls this fn:

```
function _0x_0x4ab1f1(_0x5315fb) {
    for (let _0x576c1c = 0x42; _0x576c1c > 0x0;) _0x5315fb['unshift'](_0x5315fb[_0x_0x11cd('0x5')]()), --_0x576c1c;
    return _0x5315fb[_0x_0x11cd('0x3')]('');
}
```

After my above search/replace and some renaming and massaging , it looks like:

```
function rotateRightBy66(inputArray) {
    for (let i = 0x42; i > 0x0;) {
        inputArray.unshift(inputArray.pop());
        --i;
    }
    return inputArray["join"]('');
}
```

If you study javascript array unshift and pop, you can see this is taking an array and rotating it to the right 66 times.

[7, 3, 4, 4, 2, 9] rotated once to the right would be [9, 7, 3, 4, 4, 2], then [2, 9, 7, 3, 4, 4], etc...

The calling function, cleaned up, now starts with:

```
function testInput(input) {
    try {
        let rotatedInput = rotateRightBy66(Array["from"](input));
```

The Array["from"] is really Array.from(input) and just creates an array from each char of the given input string.
The rotate function then shift/rotates it and returns it back as a string.

This function now has a big for loop that starts as:

```
        for (let _0x2d45f5 = 0x1b; _0x2d45f5 > 0x0; _0x2d45f5--) {
            if (_0x2d45f5 === 0x17 || _0x2d45f5 === 0xd) {
                if (_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5) !== _0x_0x500299()) {
                    return -0x1;
                }
            } else if (_0x2d45f5 === 0x1) {
```

```
            if (_0x5e0bfc[_0x_0x11cd('0x6')](_0x2d45f5) !== _0x_0x3dcfaa()) {
                return -0x1;
            }
```

After the above cleanup:

```
        for (let index = 0x1b; index > 0x0; index--) {
            if (index === 0x17 || index === 0xd) {
                if (rotatedInput.charCodeAt(index) !== 103) {
                    return -0x1;
                }
            } else if (index === 0x1) {
                if (rotatedInput.charCodeAt(index) !== 116) {
                    return -0x1;

                }
```

This is starting at the right of the rotatedInput and making assertions about character values at certain indices.

The first one says that at index 0x17 and 0xd, the char value better be 103.
There are a bunch of these if statements making assertions.

I wanted a way to turn this loop to my advantage so I made a copy of it and did some search/replace to change it FROM "checking" input TO creating input.

For example, the first if statement:

```
            if (index === 0x17 || index === 0xd) {
                if (rotatedInput.charCodeAt(index) !== 103) {
                    return -0x1;
                }
```

becomes:

```
            if (index === 0x17 || index === 0xd) {
                input[index]= 103;
```

I ended up with a function I called createInput():

```
function createInput() {

    let input = [];
    try {
        for (let index = 0x1b; index >= 0x0; index--) {
            if (index === 0x17 || index === 0xd) {
                input[index]= 103;
            } else if (index === 0x1) {
                input[index] =  116;
            } else if (index === 0x19 || index === 0xa) {
                input[index] =  102;
// leaving out the middle part to save space
```

```
        } else if (index === 0x6) {
            input[index] =  49;
        } else if (index === 0xb) {
            input[index] =  108;
        }
    }

    console.log(input.map(x => String.fromCharCode(x))["join"](''))
    return 0x0;
  } catch (_0x1e3b66) {}
  return -0x1;
}
```

At the end of the for loop, I have input[] with a bunch of numbers. I need to turn that into a string:

Array has a map function that lets you create a new array by modifying each element:

```
    console.log(input.map(x => String.fromCharCode(x))["join"](''))
```

Then join turns it all back into a string so we can log it.

I then ran this (can do it in Chrome devtools Source snippets OR save to a crack.js file and run with: **node crack.js**

This output:

```
th3_w1n'}flag{'js_d3bug_f0r
```

I knew it was rotated, so I rearranged this by hand:

```
flag{'js_d3bug_f0rth3_w1n'}
```

Turns out that isn't quite right. The for loop has > 0. I changed it to >= 0 and then it output:

```
_th3_w1n'}flag{'js_d3bug_f0r
```

rearranged to get the winning flag!

flag{'js_d3bug_f0r_th3_w1n'}