

Labs - JavaScript Deobfuscation

Preliminary knowledge

- `eval()`: In JavaScript, it executes commands given in texts and poses security risk.
- Obfuscation: In this context, it's about making code harder to read.
- `Console log()`: Logs input to the console
- `XMLHttpRequest (xhr)`: Object that enables browser to update web page without reloading the page (can send and receive data from server).

```
- xhr. open():  
    - Initialises a newly-created request.  
    - The third parameter stands for async and ensures the method  
    uses listeners, it causes the send() method to return as soon as the  
    request is send and the result is delivered using events.  
  
- xhr.send():  
    - Sends the request that is intialised by open() to the  
    server.  
    - It handles the request using events if request is async  
    (e.g. using xhr.onreadystatechange).  
  
- xhr.onreadystatechange:  
    - This is an event that is fired when readyState property of  
    XMLHttpRequest changes.  
  
- xhr: readyState propert  
    - This is a state that the XHR class/object/client is in.  
    - They can be illustrated in the following group:
```

The `XMLHttpRequest.readyState` property returns the state an XMLHttpRequest client is in. An XHR client exists in one of the following states:

Value	State	Description
0	UNSENT	Client has been created. <code>open()</code> not called yet.
1	OPENED	<code>open()</code> has been called.
2	HEADERS_RECEIVED	<code>send()</code> has been called, and headers and status are available.
3	LOADING	Downloading; <code>responseText</code> holds partial data.
4	DONE	The operation is complete.

-> With examples of state shown below:

Example

JS

```
const xhr = new XMLHttpRequest();
console.log("UNSENT", xhr.readyState); // readyState will be 0

xhr.open("GET", "/api", true);
console.log("OPENED", xhr.readyState); // readyState will be 1

xhr.onprogress = () => {
  console.log("LOADING", xhr.readyState); // readyState will be 3
};

xhr.onload = () => {
  console.log("DONE", xhr.readyState); // readyState will be 4
};

xhr.send(null);
```

- Trace command: Pushes 13 characters forward with this pattern (e.g A becomes N).

```
cat somefile | tr 'A-Za-z' 'N-ZA-Mn-za-m'
```

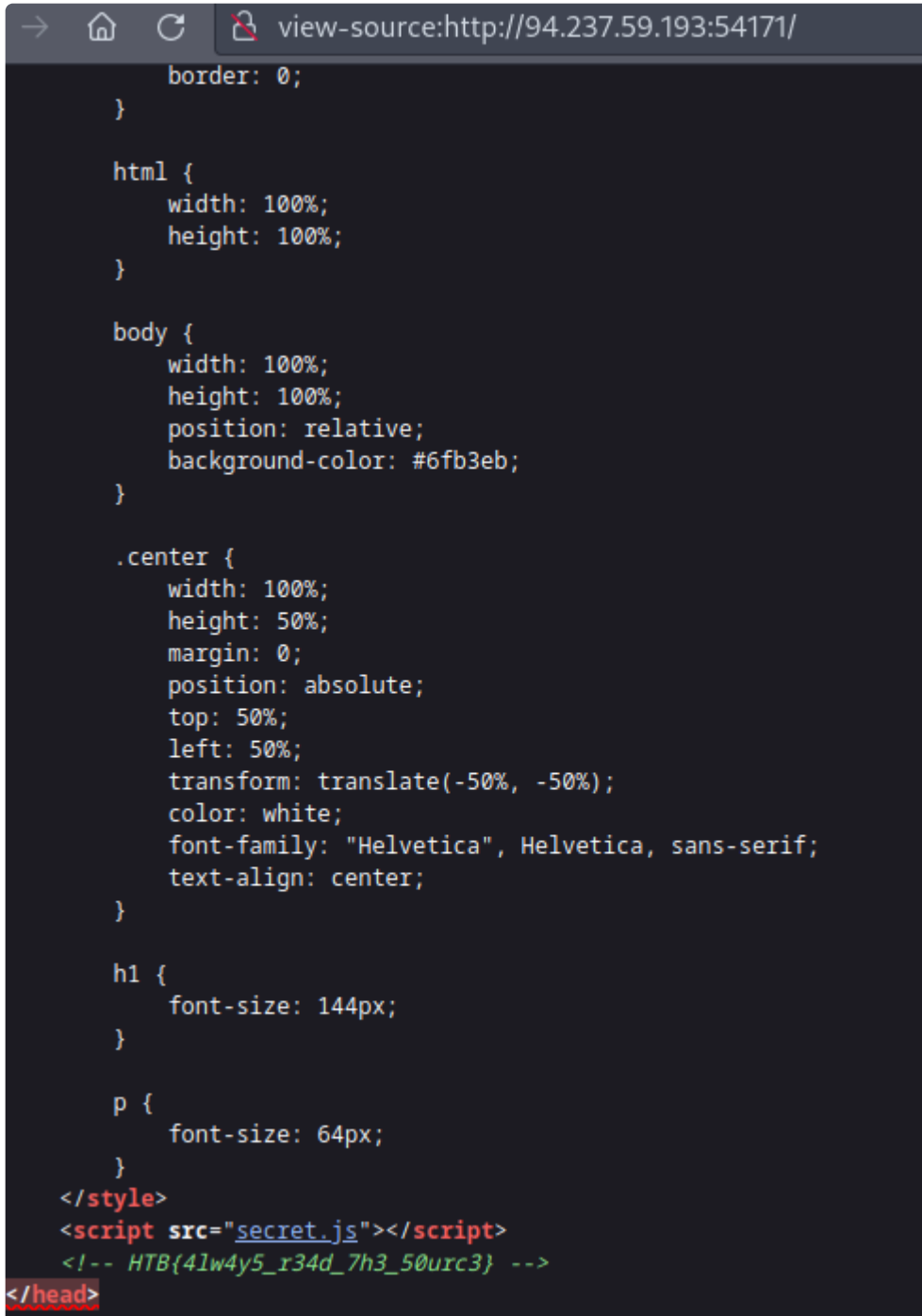
-> We can also see a symmetrical property that applying that same operations twice gives the same output, due to the property of alphabets (26 alphabets in total).

Introduction

Source Code

Question

- Repeat what you learned in this section, and you should find a secret flag, what is it?
-> We look at the source code and we get the flag:



```
→ ↩ ↺ 🔒 view-source:http://94.237.59.193:54171/

border: 0;
}

html {
  width: 100%;
  height: 100%;
}

body {
  width: 100%;
  height: 100%;
  position: relative;
  background-color: #6fb3eb;
}

.center {
  width: 100%;
  height: 50%;
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  color: white;
  font-family: "Helvetica", Helvetica, sans-serif;
  text-align: center;
}

h1 {
  font-size: 144px;
}

p {
  font-size: 64px;
}
</style>
<script src="secret.js"></script>
<!-- HTB{4lw4y5_r34d_7h3_50urc3} -->
</head>
```

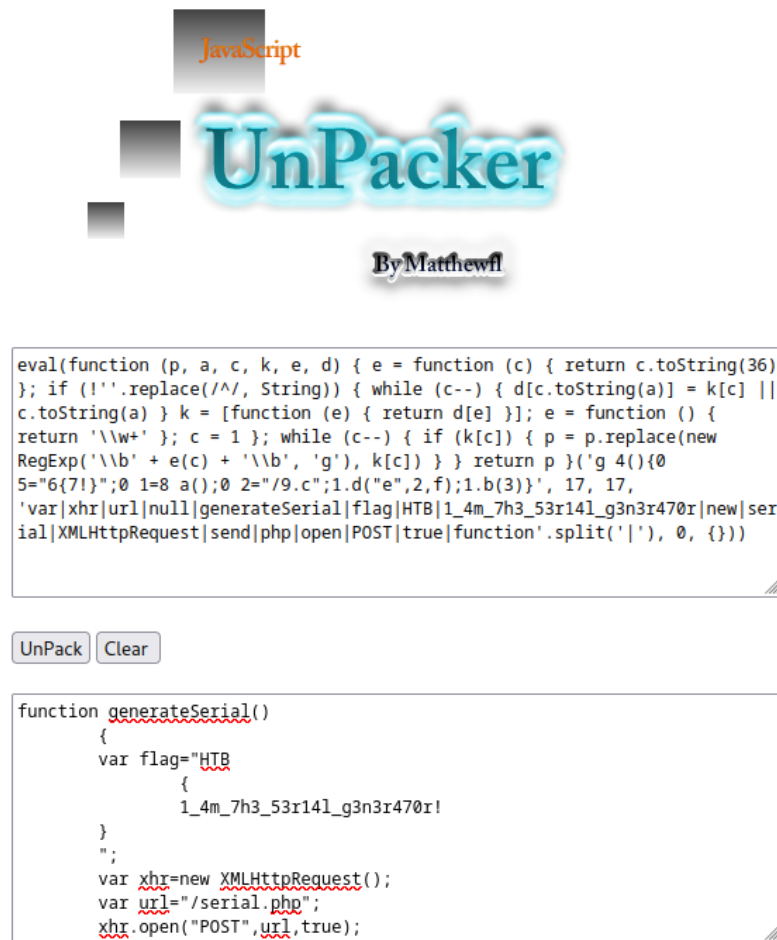
Obfuscation

Deobfuscation

Question

- Using what you learned in this section, try to deobfuscate 'secret.js' in order to get the content of the flag. What is the flag?

-> We use the <https://matthewfl.com/unPacker.html> unpacker tool and copy the javascript code into it as follows:



-> And we see the flag.

Deobfuscation Examples

HTTP Requests

Question

- Try applying what you learned in this section by sending a 'POST' request to '/serial.php'. What is the response you get?
-> We send a post request as follows

```
curl -s http://94.237.55.105:44248/serial.php -X POST
```

```
└─ [★]$ curl -s http://94.237.55.105:44248/serial.php -X POST  
N2gxNV8xNV9hX3MzY3IzN19tMzU1NGcz└─[academy-regular]─[10.10.16.9]
```

-> We get the serial value, which seems to be some secret value of strings.

Decoding

Question

- Using what you learned in this section, determine the type of encoding used in the string you got at previous exercise, and decode it. To get the flag, you can send a 'POST' request to 'serial.php', and set the data as "serial=YOUR_DECODED_OUTPUT".
-> We request for the secret serial:

```
curl -s http://83.136.254.167:58896/serial.php -X POST
```

```
-> N2gxNV8xNV9hX3MzY3IzN19tMzU1NGcz
```

-> Using Cipher Identifier, we obtained that:



-> The encoding is Base64

-> We decode the message and pass it through a post request as follows

```
echo 'N2gxNV8xNV9hX3MzY3IzN19tMzU1NGcz' | base64 -d
```

```
-> 7h15_15_a_s3cr37_m3554g3
```

```
curl -s http://83.136.254.167:58896/serial.php -X POST -d  
"serial=7h15_15_a_s3cr37_m3554g3"
```

```
[academy-regular]-[10.10.16.9]-[eric@parrot]-[~/Desktop/htb]  
└─[*]$ curl -s http://83.136.254.167:58896/serial.php -X POST  
N2gxNV8xNV9hX3MzY3IzN19tMzU1NGcz└─[academy-regular]-[10.10.16.9]-[eric@parrot]-[  
~/Desktop/htb]  
└─[*]$ echo 'N2gxNV8xNV9hX3MzY3IzN19tMzU1NGcz' | base64 -d  
7h15_15_a_s3cr37_m3554g3└─[academy-regular]-[10.10.16.9]-[eric@parrot]-[~/Desкто  
p/htb]  
└─[*]$ curl -s http://83.136.254.167:58896/serial.php -X POST -d "serial=7h15  
_15_a_s3cr37_m3554g3"  
HTB{ju57_4n07h3r_r4nd0m_53r14l}└─[academy-regular]-[10.10.16.9]-[eric@parrot]-[~
```

Skills Assessment

Scenario

- During our Penetration Test, we came across a web server that contains JavaScript and APIs.
 - We need to determine their functionality to understand how it can negatively affect our customer.

Question

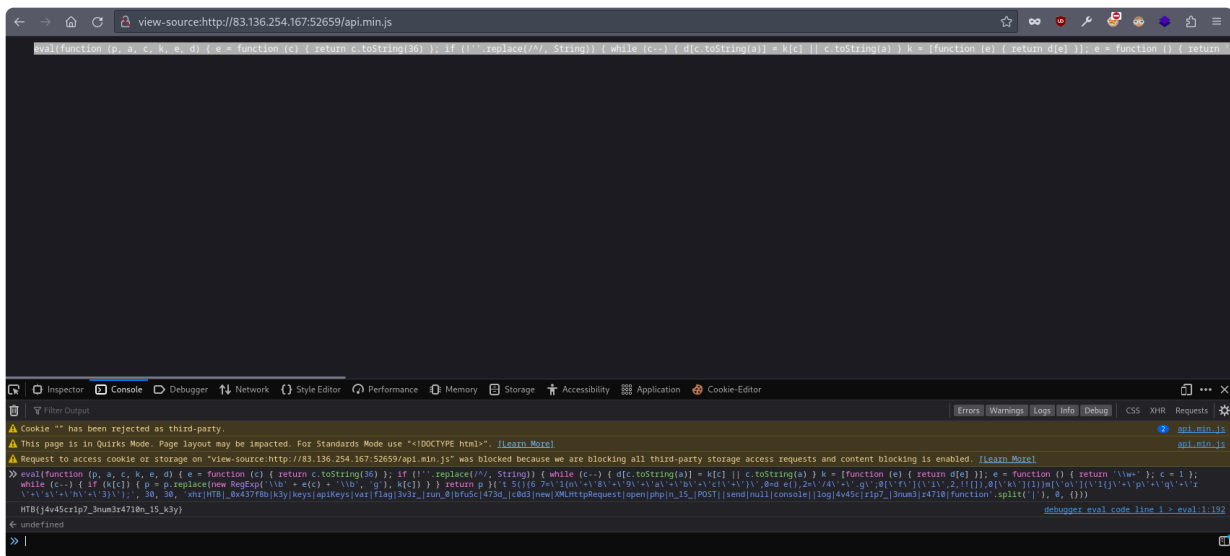
- Try to study the HTML code of the webpage, and identify used JavaScript code within it. What is the name of the JavaScript file being used?
-> We go the source page and found that

```
view-source:http://83.136.254.167:52659/
10     padding: 0;
11     border: 0;
12 }
13
14 html {
15     width: 100%;
16     height: 100%;
17 }
18
19 body {
20     width: 100%;
21     height: 100%;
22     position: relative;
23     background-color: #6fb3eb;
24 }
25
26 .center {
27     width: 100%;
28     height: 50%;
29     margin: 0;
30     position: absolute;
31     top: 50%;
32     left: 50%;
33     transform: translate(-50%, -50%);
34     color: white;
35     font-family: "Helvetica", Helvetica, sans-serif;
36     text-align: center;
37 }
38
39 h1 {
40     font-size: 144px;
41 }
42
43 p {
44     font-size: 64px;
45 }
46 </style>
47 <script src="api.min.js"></script>
48 </head>
49
50 <body>
51     <div class="center">
52         <h1>API Keys</h1>
53         <p>API Keys control panel</p>
54     </div>
55 </body>
56
57 </html>
```

-> So it is api.min.js being used.

- Once you find the JavaScript code, try to run it to see if it does any interesting functions. Did you get something in return?

-> We ran the script and we see the following:



-> HTB{j4v45cr1p7_3num3r4710n_15_k3y}

- As you may have noticed, the JavaScript code is obfuscated. Try applying the skills you learned in this module to deobfuscate the code, and retrieve the 'flag' variable.

-> We used unpacker and we see that:



```
}; if (!''.replace(/\/, String)) { while (c--) { d[c.toString(a)] = k[c] || c.toString(a) } k = [function (e) { return d[e] }]; e = function () { return '\\w+' }; c = 1 }; while (c--) { if (k[c]) { p = p.replace(new RegExp('\\b' + e(c) + '\\b', 'g'), k[c]) } } return p }('t 5() {6 7=\\1{n \\'+\\8\\'+\\9\\'+\\a\\'+\\b\\'+\\c!\\'+\\}\\',0=d e(),2=\\'/4\\'+\\'g\\';0[\\'f \\'] (\\'i\\',2,!![]),0[\\'k\\'](l)m[\\'o\\'] (\\'1{j\\'+\\'p\\'+\\'q\\'+\\'r\\'+\\'s\\'+\\'h \\'+\\'3\\'\\')', 30, 30, 'xhrlHTB|_0x437f8b|k3y|keys|apiKeys|var|flag|3v3r_|run_0|bfu5c|473d_|c0d3|r ew|XMLHttpRequest|open|php|n_15_|POST|send|null|console|log|4v45c|r1p7_|3 num3|r4710|function'.split('|'), 0, {}))
```

UnPack Clear

```
function apiKeys()
{
    var flag='HTB
    {
        n'+ '3v3r_' + 'run_0' + 'bfu5c' + '473d_' + 'c0d3!' + '
    }
    ,xhrl=new XMLHttpRequest(),_0x437f8b='/keys+'.php';
    xhrl['open']('POST',_0x437f8b,!![]),xhrl['send'](null)
}
console['log']('HTB
```

-> So we obtained the flag variable as HTB{n3v3r_run_0bfu5c473d_c0d3!} after removing the + and spaces above.

- Try to Analyze the deobfuscated JavaScript code, and understand its main functionality. Once you do, try to replicate what it's doing to get a secret key. What is the key?

-> The deobfuscated code is as follows

```
function apiKeys()  
{  
  var flag='HTB'  
    {  
      n'+ '3v3r_'+'run_0'+ 'bfu5c'+ '473d_'+'c0d3!'+'  
    }  
    ',xhr=new XMLHttpRequest(),_0x437f8b='/keys'+'.php';  
    xhr['open']('POST',_0x437f8b,!![]),xhr['send'](null)  
}  
console['log']('HTB'  
  {  
    j'+ '4v45c'+ 'r1p7_'+'3num3'+ 'r4710'+ 'n_15_'+'k3y'  
  }  
  ');
```

-> It's functionality is to open a new request and sending it to /keys.php.

-> So, we will replicate it and obtained the key

```
curl -s http://83.136.254.167:52659//keys.php -X POST  
  
# -> 4150495f70336e5f37333537316e365f31355f66756e
```

```
[*]$ curl -s http://83.136.254.167:52659//keys.php -X POST  
4150495f70336e5f37333537316e365f31355f66756e [academy-regular]-[10.10.16.9]-[er
```

- Once you have the secret key, try to decide it's encoding method, and decode it. Then send a 'POST' request to the same previous page with the decoded key as "key=DECODED_KEY". What is the flag you got?
-> We first identify what encoding is being used using Cipher Identifier (though it looks like hex encoding):

https://www.boxentriq.com/code-breaking/cipher-identifier

BOXENTRIQ

Analysis Results

4150495f70336e5f37333537316e365f31355f66756e

Your ciphertext is likely of this type:

Hexadecimal Code (click to read more)

-> Hence, we decode it as follows

```
echo '4150495f70336e5f37333537316e365f31355f66756e' | xxd -p -r
```

```
# -> API_p3n_73571n6_15_fun
```

```
[*]$ echo '4150495f70336e5f37333537316e365f31355f66756e' | xxd -p -r
API_p3n_73571n6_15_fun [academy-regular]-[10.10.16.9]-[eric@parrot]-[~/Desktop/
```

-> Next, we send a post request using this secret key:

```
curl -s http://83.136.254.167:52659/keys.php -X POST -d
"key=API_p3n_73571n6_15_fun"
```

```
[*]$ curl -s http://83.136.254.167:52659/keys.php -X POST -d "key=API_p3n_7
3571n6_15_fun"
HTB{r34dy_70_h4ck_my_w4y_1n_2_HTB} [academy-regular]-[10.10.16.9]-[eric@parrot]
[~/Desktop/htb]
```