

Code Obfuscation

Before we start learning about `deobfuscation`, we must first learn about `code obfuscation`. Without understanding how code is obfuscated, we may not be able to successfully deobfuscate the code, especially if it was obfuscated using a custom obfuscator.

What is obfuscation

Obfuscation is a technique used to make a script more difficult to read by humans but allows it to function the same from a technical point of view, though performance may be slower. This is usually achieved automatically by using an obfuscation tool, which takes code as an input, and attempts to re-write the code in a way that is much more difficult to read, depending on its design.

For example, code obfuscators often turn the code into a dictionary of all of the words and symbols used within the code and then attempt to rebuild the original code during execution by referring to each word and symbol from the dictionary. The following is an example of a simple JavaScript code being obfuscated:

```
console.log('HTB JavaScript Deobfuscation Module');
```

Normal

☒ Fast Decode

☐ Special Characters

Obfuscate

Clear

```
eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}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}('5.4(\\'3 2 1
0\\');',6,6,'Module|Deobfuscation|JavaScript|HTB|log|consol
e'.split('|'),0,{}))
```

Codes written in many languages are published and executed without being compiled in interpreted languages, such as Python, PHP, and JavaScript.

While Python and PHP usually reside on the server-side and hence are hidden from end-users, JavaScript is usually used within browsers at the client-side, and the code is sent to the user and executed in cleartext. This is why obfuscation is very often used with JavaScript.

Use Cases

There are many reasons why developers may consider obfuscating their code. One common reason is to hide the original code and its functions to prevent it from being reused or copied without the developer's permission, making it more difficult to reverse engineer the code's original functionality. Another reason is to provide a security layer when dealing with authentication or encryption to prevent attacks on vulnerabilities that may be found within the code.

It must be noted that doing authentication or encryption on the client-side is not recommended, as code is more prone to attacks this way.

The most common usage of obfuscation, however, is for malicious actions. It is common for attackers and malicious actors to obfuscate their malicious scripts to prevent Intrusion Detection and Prevention systems from detecting their scripts. In the next section, we will learn how to obfuscate a simple JavaScript code and attempt running it before and after obfuscation to note any differences.

Basic Obfuscation

Code obfuscation is usually not done manually, as there are many tools for various languages that do automated code obfuscation. Many online tools can be found to do so, though many malicious actors and professional developers develop their own obfuscation tools to make it more difficult to deobfuscate.

Running JavaScript code

Let us take the following line of code as an example and attempt to obfuscate it:

Code: javascript

```
console.log('HTB JavaScript Deobfuscation Module');
```

First, let us test running this code in cleartext, to see it work in action. We can go to [JSConsole](#), paste the code and hit enter, and see its output:

Use **:help** to show jsconsole commands
version: 2.1.2

```
> console.log('HTB JavaScript Deobfuscation Module');
```

HTB JavaScript Deobfuscation Module

⏪ undefined

We see that this line of code prints `HTB JavaScript Deobfuscation Module`, which is done using the `console.log()` function.

Minifying JavaScript code

A common way of reducing the readability of a snippet of JavaScript code while keeping it fully functional is JavaScript minification. `Code minification` means having the entire code in a single (often very long) line. `Code minification` is more useful for longer code, as if our code only consisted of a single line, it would not look much different when minified.

Many tools can help us minify JavaScript code, like [javascript-minifier](#). We simply copy our code, and click `Minify`, and we get the minified output on the right:

Input JavaScript

```
function log() {  
  console.log('HTB JavaScript Deobfuscation Module');  
}
```

[Minify](#) [Download as File](#) [RAW](#) [Clear](#)

Minified Output

```
function log(){console.log("HTB JavaScript Deobfuscation Module")}
```

[Copy to Clipboard](#) [Select All](#)

Once again, we can copy the minified code to [JSConsole](#), and run it, and we see that it runs as expected. Usually, minified JavaScript code is saved with the extension `.min.js`.

Note: Code minification is not exclusive to JavaScript, and can be applied to many other languages, as can be seen on [javascript-minifier](#).

Packing JavaScript code

Now, let us obfuscate our line of code to make it more obscure and difficult to read. First, we will try [BeautifyTools](#) to obfuscate our code:

Normal

☒ Fast Decode

☐ Special Characters

Obfuscate

Clear

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}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}('5.4\\'3 2 1 0\\');',6,6,'Module|Deobfuscation|JavaScript|HTB|log|console'.split('|'),0,{}))
```

Code: javascript

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}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}('5.4\\'3 2 1 0\\');',6,6,'Module|Deobfuscation|JavaScript|HTB|log|console'.split('|'),0,{}))
```

We see that our code became much more obfuscated and difficult to read. We can copy this code into <https://jsconsole.com>, to verify that it still does its main function:

Use **:help** to show jsconsole commands
version: 2.1.2

```
> eval(function(p,a,c,k,e,d){e=function(c){return c;if(''.replace(/\/,String))while(c--){d[c]=k[c]||c}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}('5.4(\3 2 1 0\');',6,6,'Module|Deobfuscation|JavaScript|HTB|log|console'.split('|'),0,{}))
```

HTB JavaScript Deobfuscation Module

We see that we get the same output.

Note: The above type of obfuscation is known as "packing", which is usually recognizable from the six function arguments used in the initial function "function(p,a,c,k,e,d)".

A **packer** obfuscation tool usually attempts to convert all words and symbols of the code into a list or a dictionary and then refer to them using the (p,a,c,k,e,d) function to re-build the original code during execution. The (p,a,c,k,e,d) can be different from one packer to another. However, it usually contains a certain order in which the words and symbols of the original code were packed to know how to order them during execution.

While a packer does a great job reducing the code's readability, we can still see its main strings written in cleartext, which may reveal some of its functionality. This is why we may want to look for better ways to obfuscate our code.

Advanced Obfuscation

So far, we have been able to make our code obfuscated and more difficult to read. However, the code still contains strings in cleartext, which may reveal its original functionality. In this section, we will try a couple of tools that should completely obfuscate the code and hide any remanence of its original functionality.

Obfuscator

Let's visit <https://obfuscator.io>. Before we click **obfuscate**, we will change **String Array Encoding** to **Base64**, as seen below:

The screenshot shows the Obfuscator.io web interface with the following settings:

- Reset options** button
- ☒ Compact code
- Identifier Names Generator**: hexadecimal
- Identifiers Dictionary**: foo
- ☒ String Array
- ☒ Rotate String Array
- ☒ Shuffle String Array
- String Array Encoding**: Base64
- String Array Threshold**: 0.8
- ☐ Disable Console Output
- ☐ Self Defending
- ☐ Debug Protection
- ☐ Debug Protection Interval
- Domain lock**: domain.com
- Sourcemaps**: Off
- Source Map Base URL**: http://localhost:3000
- Source Map File Name**: example
- Seed**

Now, we can paste our code and click `obfuscate` :

Copy & Paste JavaScript Code	Upload JavaScript File	Output
<pre>1 console.log('HTB JavaScript Deobfuscation Module');</pre>		
<div>Obfuscate</div>		

We get the following code:

Code: javascript

```
var _0x1ec6=['Bg9N','sfrciePHDMfty3jPChqGrgvVyMz1C2nHDgLVBIBnB2r1Bgu='];
(function(_0x13249d,_0x1ec6e5){var _0x14f83b=function(_0x3f720f){while(--_0x3f720f){_0x13249d['push'](_0x13249d['shift']());}};_0x14f83b(++_0x1ec6e5);}(_0x1ec6,0xb4));var
_0x14f8=function(_0x13249d,_0x1ec6e5){_0x13249d=_0x13249d-0x0;var
_0x14f83b=_0x1ec6[_0x13249d];if(_0x14f8['e0TqeL']===undefined){var
_0x3f720f=function(_0x32fbfd){var
_0x523045='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/,',_0x4f8a49=String(_0x32fbfd)['replace'](/=+$/,'');var _0x1171d4='';for(var
_0x44920a=0x0,_0x2a30c5,_0x443b2f,_0xcd142=0x0;_0x443b2f=_0x4f8a49['charAt'](_0xcd142++);~_0x443b2f&&(_0x2a30c5=_0x44920a%0x4?
_0x2a30c5*0x4+_0x443b2f:_0x443b2f,_0x44920a++%0x4)?
_0x1171d4+=String['fromCharCode'](_0xff&_0x2a30c5>>(-0x2*_0x44920a&0x6)):0x0){_0x443b2f=_0x523045['indexOf'](_0x443b2f);}return
_0x1171d4;};_0x14f8['oZLYBE']=function(_0x8f2071){var
_0x49af5e=_0x3f720f(_0x8f2071);var _0x52e65f=[];for(var
_0x1ed1cf=0x0,_0x79942e=_0x49af5e['length'];_0x1ed1cf<_0x79942e;_0x1ed1cf++){_0x52e65f+=('%'+('00'+_0x49af5e['charCodeAt'](_0x1ed1cf))['toString'])(0x10))
['slice'](-0x2);}return decodeURIComponent(_0x52e65f);},_0x14f8['qHtbNC']=
{,_0x14f8['e0TqeL']=!![];}_0x20247c=_0x14f8['qHtbNC'][_0x13249d];return
_0x20247c===undefined?(_0x14f83b=_0x14f8['oZLYBE']
(_0x14f83b),_0x14f8['qHtbNC']
[_0x13249d]=_0x14f83b):_0x14f83b=_0x20247c,_0x14f83b;};console[_0x14f8('0x0'
)](_0x14f8('0x1'));
```

This code is obviously more obfuscated, and we can't see any remnants of our original code.

We can now try running it in <https://jsconsole.com> to verify that it still performs its original

function. Try playing with the obfuscation settings in <https://obfuscator.io> to generate even more obfuscated code, and then try rerunning it in <https://jsconsole.com> to verify it still performs its original function.

More Obfuscation

Now we should have a clear idea of how code obfuscation works. There are still many variations of code obfuscation tools, each of which obfuscates the code differently. Take the following JavaScript code, for example:

Code: javascript

[illegible]

We can still run this code, and it would still perform its original function:

[illegible]

HTB JavaScript Deobfuscation Module

< undefined

Note: The above code was snipped as the full code is too long, but the full code should successfully run.

We can try obfuscating code using the same tool in [JSF](#), and then rerunning it. We will notice that the code may take some time to run, which shows how code obfuscation could affect the performance, as previously mentioned.

There are many other JavaScript obfuscators, like [JJ Encode](#) or [AA Encode](#). However, such obfuscators usually make code execution/compilation very slow, so it is not recommended to be used unless for an obvious reason, like bypassing web filters or restrictions.

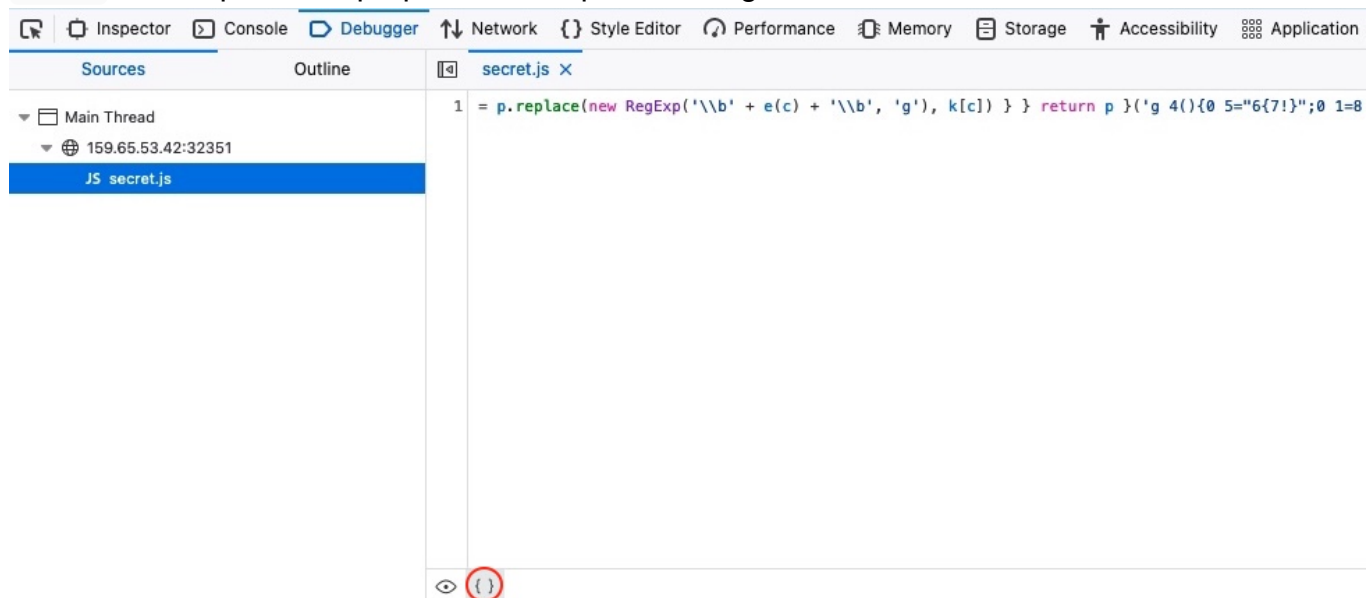
Deobfuscation

Now that we understand how code obfuscation works let's start our learning towards deobfuscation. Just as there are tools to obfuscate code automatically, there are tools to beautify and deobfuscate the code automatically.

Beautify

We see that the current code we have is all written in a single line. This is known as `Minified JavaScript` code. In order to properly format the code, we need to `Beautify` our code. The most basic method for doing so is through our `Browser Dev Tools`.

For example, if we were using Firefox, we can open the browser debugger with [`CTRL+SHIFT+Z`], and then click on our script `secret.js`. This will show the script in its original formatting, but we can click on the ' { } ' button at the bottom, which will `Pretty Print` the script into its proper JavaScript formatting:

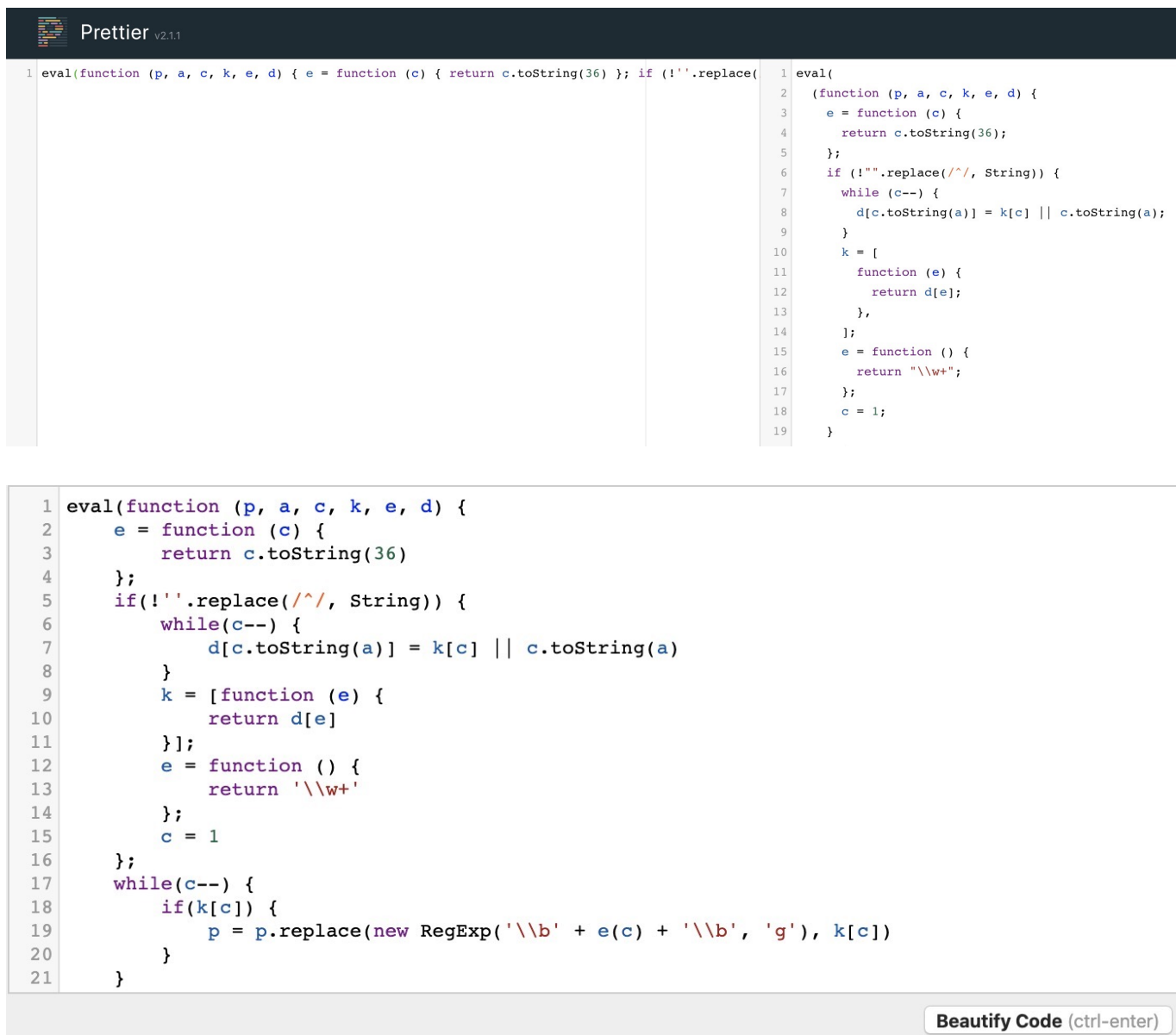


Furthermore, we can utilize many online tools or code editor plugins, like [Prettier](#) or [Beautifier](#). Let us copy the `secret.js` script:

Code: javascript

```
eval(function (p, a, c, k, e, d) { e = function (c) { return c.toString(36)
}; 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 }('g 4(){0 5="6{7!}";0 1=8 a();0
2="/9.c";1.d("e",2,f);1.b(3)}', 17, 17,
'var|xhr|url|null|generateSerial|flag|HTB|flag|new|serial|XMLHttpRequest|sen
d|php|open|POST|true|function'.split('|'), 0, {}))
```

We can see that both websites do a good job in formatting the code:



The screenshot shows the Prettier v2.1.1 interface with the JavaScript code formatted into two columns. The left column shows the code with line numbers 1 through 21. The right column shows the code with line numbers 1 through 19. The code is formatted with proper indentation and line wrapping.

```
1 eval(function (p, a, c, k, e, d) {
2   e = function (c) {
3     return c.toString(36)
4   };
5   if (!''.replace(/^/, String)) {
6     while (c--) {
7       d[c.toString(a)] = k[c] || c.toString(a)
8     }
9     k = [function (e) {
10       return d[e]
11     }];
12     e = function () {
13       return '\\w+'
14     };
15     c = 1
16   };
17   while (c--) {
18     if (k[c]) {
19       p = p.replace(new RegExp('\\b' + e(c) + '\\b', 'g'), k[c])
20     }
21   }
22 }
```

Beautify Code (ctrl-enter)

However, the code is still not very easy to read. This is because the code we are dealing with was not only minified but obfuscated as well. So, simply formatting or beautifying the code will not be enough. For that, we will require tools to deobfuscate the code.

Deobfuscate

We can find many good online tools to deobfuscate JavaScript code and turn it into something we can understand. One good tool is [UnPacker](#). Let's try copying our above-obfuscated code and run it in UnPacker by clicking the `UnPack` button.

Tip: Ensure you do not leave any empty lines before the script, as it may affect the deobfuscation process and give inaccurate results.

```
eval(function (p, a, c, k, e, d) { e = function (c) { return  
c.toString(36) }; 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 }('g 4(){0 5="6{7!}";0 1=8 a();0 2="/9.c";1.d("e",2,f);1.b(3)}',  
17, 17,  
'var|xhr|url|null|generateSerial|flag|HTB|flag|new|serial|XMLHttpRequest|  
send|php|open|POST|true|function'.split('|'), 0, {}))
```

UnPack Clear

```
function generateSerial()  
{  
    ...SNIP...  
    var xhr=new XMLHttpRequest();  
    var url="/serial.php";  
    xhr.open("POST",url,true);  
    xhr.send(null)  
}
```

We can see that this tool does a much better job in deobfuscating the JavaScript code and gave us an output we can understand:

Code: javascript

```
function generateSerial() {  
    ... SNIP ...  
    var xhr = new XMLHttpRequest;  
    var url = "/serial.php";  
    xhr.open("POST", url, true);  
    xhr.send(null);  
};
```

As previously mentioned, the above-used method of obfuscation is `packing`. Another way of `unpacking` such code is to find the `return` value at the end and use `console.log` to print it instead of executing it.

Reverse Engineering

Though these tools are doing a good job so far in clearing up the code into something we can understand, once the code becomes more obfuscated and encoded, it would become much more difficult for automated tools to clean it up. This is especially true if the code was obfuscated using a custom obfuscation tool.

We would need to manually reverse engineer the code to understand how it was obfuscated and its functionality for such cases. If you are interested in knowing more about advanced JavaScript Deobfuscation and Reverse Engineering, you can check out the [Secure Coding 101](#) module, which should thoroughly cover this topic.