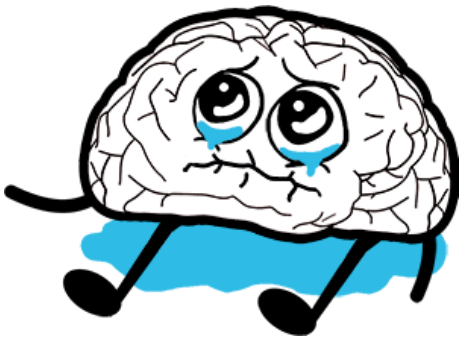
JavaScript Obfuscator Tool

A free and efficient obfuscator for JavaScript (including ES2017). Make your code harder to copy and prevent people from stealing your work. This tool is a Web UI to the excellent (and open source) `javascript-obfuscator@0.18.1` created by Timofey Kachalov.

Star Watch



What is this?

This tool transforms your original JavaScript source code into a new representation that's harder to understand, copy, re-use and modify without authorization. The obfuscated result will have the exact functionality of the original code.

So, like UglifyJS, Closure Compiler, etc?

Yes and no. While UglifyJS (and others minifiers) does make the output code harder to understand (compressed and ugly), it can be easily be transformed into something readable using a JS Beautifier.

This tool prevents that by using various transformations and "traps", such as **self-defending** and **debug protection**.

How does the obfuscation works?

Through a series of transformations, such as variable / function / arguments renaming, strings removal, and others, your source code is transformed into something unreadable, while working exatcly as before.

Read more in the FAQ...

Sounds great!

Just paste your code or upload it below and click on "obfuscate".

Also, be sure to read about all the options to understand all the trade-offs between code protection and code size / speed.

Copy & Paste JavaScript CodeUpload JavaScript FileOutput

```
[ log , Hello\x20wo
rld!'];
(function(_0x16f6f0
  0x40111c){var
```

Download obfuscated code

- [ ] Evaluate
- [x] Compact code

Identifier Names Generator

hexadecimal

hexadecimal

mangled

Identifiers Prefix

- [ ] Rename Globals
- [ ] Self Defending

Control Flow Flatteing

Control Flow Flattening Threshold

0.75

Dead Code Injection

Dead Code Injection Threshold

0.4

String Array

Rotate String Array

String Array Encoding

Off

Off

Base64

RC4

String Array Threshold

0.8

Transform Object Keys

Unicode Escape Sequence

Disable Console Output

Debug Protection

Debug Protection Interval

Domain lock

domain.com

Reserved Names

^someVariable *or *RegExp

Reserved Strings

^some *string *or RegExp

Sourcemaps

Off

Off

Inline

Separate

Source Map Base URL

http://localhost:3000

Source Map File Name

example

Seed

0

Target

Browser

Browser

Browser No Eval

Node

Compact Code	Removes line breaks from the output obfuscated code.
Identifier Names Generator	Use this option to control how identifiers (variable names, functions names, etc) will be obfuscated. **hexadecimal** Generates random identifier names using a hexadecimal pattern (e.g: `0xabc123`) **mangled** Uses short identifier names (e.g: `a`, `b`, `c`, etc.)
Identifiers Prefix	This options makes all global identifiers have a specific prefix.

Use this option when obfuscating multiple files that are loaded on the same page. This option helps to avoid conflicts between global identifiers of these files. Use a different prefix for each file.

Rename Globals

This option can break your code. Only enable it if you know what it does.

Enables the obfuscation of global variables and function names with declaration.

Self Defending

This option makes the output code resilient against formating and variable renaming. If one tries to use a JavaScript beautifier on the obfuscated code, the code won't work anymore, making it harder to understand and modify it.

requires the **Compact Code** setting.

Control Flow Flattening

This option greatly affects the performance up to 1.5x slower runtime speed.

Enables code control flow flattening. Control flow flattening is a structure transformation of the source code that hinders program comprehension. See the docs on JavaScript's obfuscator GH page for an example of how the transformation works.

Control Flow Flattening Threshold	You can use this setting to adjust the probability (from 0 to 1) that a `controlFlowFlattening` transformation will be applied to a node. In larger codebases it's advised to lower this value, because larger amounts of control flow transformations can increase the size of your code and slow it down.

Dead Code Injection

This option increases the size of the obfuscated code greatly (up to 200%).

This feature adds random blocks of dead code (i.e: code that won't be executed) to the obfuscated output, making it harder to be reverserd-engineered. See the docs on JavaScript Obfuscator's GH page for an example of how this feature works.

Dead Code Injection Threshold	You can use this setting to adjust the probability (from 0 to 1) that a node will be affected by the `deadCodeInjection` option.

String Array

requires the **String Array** option.

Removes string literals and place them in a special array. For instance the string "Hello World" in `var m = "Hello World";` will be replaced to a call to a function that will retrieve its value at runtime, e.g: `var m = _0xb0c3('0x1');`

See the options below on how to configure this feature be more or less resilient.

Rotate String Array	Shift the `stringArray` array by a fixed and random (generated at the code obfuscation) places. This makes it harder to match the order of the removed strings to their original place. This option is recommended if your original source code isn't small, as the helper function can attract attention.
Encode String Literals	This option can slightly slow down your script. Encode all string literals of the `stringArray` using either Base64 or RC4 and inserts a special function that it's used to decode it back at runtime. Beware that the RC4 option is about 30-35% slower than the Base64 option, but it's more difficult to retrieve the strings back.
String Array Threshold	You can use this setting to adjust the probability (from 0 to 1) that a string literal will be inserted into the `stringArray`.

This setting is useful in large codebases as repeatdely calls to the `stringArray` function can slow down your code.

Transform Object Keys

Transforms (obfuscates) object keys.

For instance, this code `var a = {enabled: true};` when obfuscated with this option will hide the `enabled` object key: `var a = {}; a[_0x2ae0[('0x0')] = true;.`

See the official documentation of the JavaScript Obfuscator on GitHub for a full example.

ideally used with the **String Array** setting.

Escape Unicode Sequence

Converts all the strings to their unicode representation. For instance, the string `"Hello World!"` will be converted to `"'\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64\x21".`

This convertion is pretty easy to revert, and will increase the obfuscated code size greatly. It's not recommended on larger code bases.

Disable Console Output

Disables the use of `console.log`, `console.info`, `console.error` and `console.warn` by replacing them with empty functions. This makes the use of the debugger harder.

Debug Protection

Can freeze your browser if you open the Developer Tools.

This option makes it almost impossible to use the **Console** tab of the Developer Tools (both on Google Chrome and Mozilla Firefox).

Debug Protection Interval	If checked, an interval is used to force the debug mode on the **Console** tab, making it harder to use other features of the Developer Tools. How does it works? A special code that calls `debugger;` repeatedly is inserted throughout the obfuscated source code.

Domain Lock

Locks the obfuscated source code so it only runs on specific domains and/or sub-domains. This makes really hard for someone just copy and paste your source code and run elsewhere.

Multiple domains and sub-domains

It's possible to lock your code to more than one domain or sub-domain. For instance, to lock it so the code only runs on **www.example.com** add `www.example.com`, to make it work on **any sub-domain from example.com**, use `.example.com`.

Reserved Names

Disables obfuscation and generation of identifiers, which being matched by passed RegExp patterns.

For instance, if you add `^someName`, the obfuscator will ensure that all variables, function names and function arguments that starts with **someName** will not get mangled.

Reserved Strings

Disables transformation of string literals, which being matched by passed RegExp patterns.

For instance, if you add `^some *string`, the obfuscator will ensure that all strings that starts with **some string** will not get moved to the `stringArray`.

Source Map

Be sure not to upload the obfuscated source code with the inline source map embedded on it, as it contains your original source code.

Source maps can be useful to help you debug your obfuscated Java Script source code. If you want or need to debug in production, you can upload the separate source map file to a secret location and then point your browser there. Read more about source maps on the Google Chrome Developer Tools website.

Inline Source Map

This embeds the source map of your source in the result of the obfuscated code. Useful if you just want to debug locally on your machine.

Separate Source Map

This generates a separate file with the source map. Useful to debug code in production, as this enables you to upload the source map to a secret location on your server and then point your browser to use it.

Use the **Source Map Base URL** and **Source Map File Name** to customize the `sourceMappingURL` property that will get appended to the end of your obfuscated code.

For instance, if you set the **Base URL** to `"http://localhost:9000"` and **File Name** to `"example"`, you'll get: `//# sourceMappingURL=http://localhost:9000/example.js.map`. appended to the end of your obfuscated code.

Seed

By default (`seed = 0`), each time you obfuscate your code you'll get a new result (i.e: different variable names, different variables inserted into the `stringArray`, etc). If you want repeatable results, set the `seed` to a specific integer.

Target

You can set the target environment of the obfuscated code to one of the following:

- **Browser**
- **Browser No Eval**
- **Node**

Currently the output of `browser` and `node` is identical.

FAQ

Why would I want to obfuscate my JavaScript code?

There're a numerous reasons why it's a good idea to protect your code, such as:

- Prevent anyone from simply copy/pasting your work. This is specially important on 100% client side projects, such as HTML5 games;
- Removal of comments and whitespace that aren't needed. Making it faster to load and harder to understand;
- Protection of work that hasn't been paid yet. You can show your work to the client knowing that they won't have the source code until the invoice has been paid.

Is this obfuscator absolutely foolproof?

No, while it's impossible to recover the exact original source code, someone with the time, knowledge and patience can reverse-engineer it.

Since the JavaScript runs on the browser, the browser's JavaScript engine must be able to read and interpret it, so there's no way to prevent that. And any tool that promises that is not being honest.

Why my obfuscated code is larger than my original source?

Because the obfuscator introduces new pieces of code that are meant to protect and defend against debugging and reverse-engineering. Also strings are converted to `\xAB` hexadecimal code to make things a little bit harder to understand. You don't have to worry too much about code size because since there're a lot of repetition, the obfuscated code will be compressed extremely well by your webserver (if you have GZIP compression enabled on your server, which most do nowadays).

Can I run a minifier such as UglifyJS or Google Closure Compiler on the obfuscated output?

No, it's not recommended and in some cases it'll break the code (such as if you enable **self-defending**). You can run your code through a minifier before to make sure that it removes dead code and do other optimizations, though.

Do you store my source code?

No. The source is processed by our application server, then to the obfuscator and back to the browser, so it only stays on our server memory for a brief period of time (usually milliseconds).

Can I recover the original source code from the obfuscated one?

No, it's impossible to revert the obfuscated code back to your original code, so keep the original safe.

Does this tool works with Node.js source code?

Yes.

I want to run the obfuscator on my own server/machine. Is it possible?

Sure. This tool uses a free and open source (BSD-2-Clause licensed) obfuscator written in TypeScript. You can go to its GitHub page and read more there.

There are also a number of plugins, such as: webpack-obfuscator, gulp-javascript-obfuscator and grunt-contrib-obfuscator.

Also, this web app is open-source as well. Check out our GitHub.

What are other similar tools?

If you're interested in just uglyfing and compressing your code, I suggest JSCompress.com.

Support project:

Also:
(Bitcoin) 14yhtZxLNp6ekZAgmEmPJqEKUP2VtUxQK6
(Ether) 0x5Df9eBcFB2D0f3315d03Ac112104b9023C409dc1
Big thanks to all supporters!