Introduction

Welcome to the JavaScript Deobfuscation module!

Code deobfuscation is an important skill to learn if we want to be skilled in code analysis and reverse engineering. During red/blue team exercises, we often come across obfuscated code that wants to hide certain functionalities, like malware that utilizes obfuscated JavaScript code to retrieve its main payload. Without understanding what this code is doing, we may not know what exactly the code is doing, and hence may not be able to complete the red/blue team exercise.

In this module, we start by learning the general structure of an HTML page and then will locate JavaScript code within it. Once we do that, we will learn what obfuscation is, how it is done, and where it is used and follow that by learning how to deobfuscate such code. Once the code is deobfuscated, we will attempt to understand its general usage to replicate its functionality and uncover what it does manually.

The following topics will be discussed:

- Locating JavaScript code
- Intro to Code Obfuscation
- How to Deobfuscate JavaScript code
- How to decode encoded messages
- Basic Code Analysis
- Sending basic HTTP requests

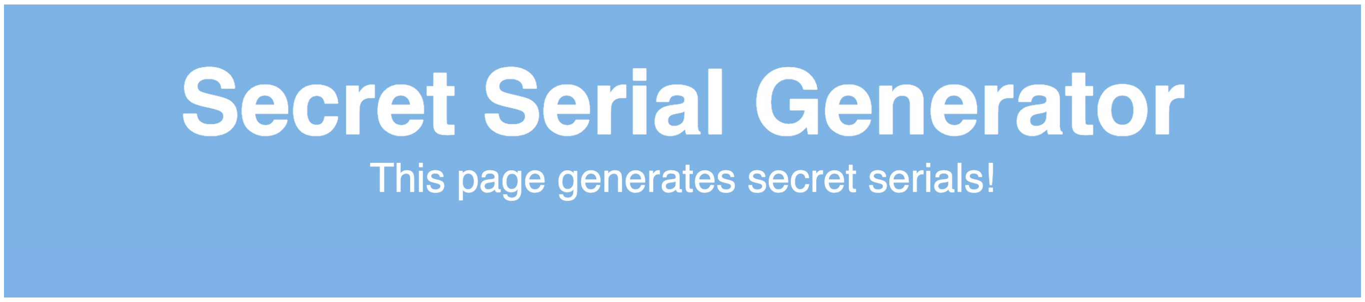
Source Code

Most websites nowadays utilize JavaScript to perform their functions. While `HTML` is used to determine the website's main fields and parameters, and `CSS` is used to determine its design, `JavaScript` is used to perform any functions necessary to run the website. This happens in the background, and we only see the pretty front-end of the website and interact with it.

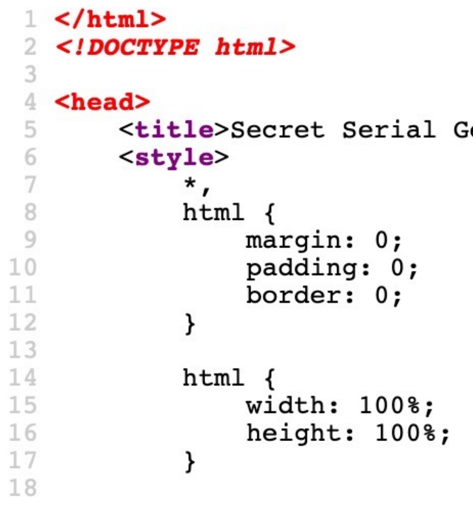
Even though all of this source code is available at the client-side, it is rendered by our browsers, so we do not often pay attention to the HTML source code. However, if we wanted to understand a certain page's client-side functionalities, we usually start by taking a look at the page's source code. This section will show how we can uncover the source code that contains all of this and understand its general usage.

HTML

We will start by starting the exercise below, open Firefox in our PwnBox, and visit the url shown in the question:



As we can see, the website says `Secret Serial Generator`, without having any input fields or showing any clear functionality. So, our next step is to peak at its source code. We can do that by pressing `[CTRL + U]`, which should open the source view of the website:



```
</html>
<!DOCTYPE html>

<head>
    <title>Secret Serial Generator</title>
    <style>
        *,
        html {
            margin: 0;
            padding: 0;
            border: 0;
        }

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

```

As we can see, we can view the `HTML` source code of the website.

CSS

`CSS` code is either defined `internally` within the same `HTML` file between `<style>` elements, or defined `externally` in a separate `.css` file and referenced within the `HTML` code.

In this case, we see that the `CSS` is internally defined, as seen in the code snippet below:

Code: html

```
<style>
    *,
    html {
        margin: 0;
        padding: 0;
        border: 0;
    }
    ...SNIP...
    h1 {
        font-size: 144px;
    }
    p {
        font-size: 64px;
    }
</style>
```

If a page `CSS` style is externally defined, the external `.css` file is referred to with the `<link>` tag within the HTML head, as follows:

Code: html

```html
<head>
    <link rel="stylesheet" href="style.css">
</head>
```

JavaScript

The same concept applies to `JavaScript`. It can be internally written between `<script>` elements or written into a separate `.js` file and referenced within the `HTML` code.

We can see in our `HTML` source that the `.js` file is referenced externally:

Code: html

```html
<script src="secret.js"></script>
```

We can check out the script by clicking on `secret.js`, which should take us directly into the script. When we visit it, we see that the code is very complicated and cannot be comprehended:

Code: javascript

```javascript
eval(function (p, a, c, k, e, d) { e = function (c) { '...SNIP...
|true|function'.split('|'), 0, {}))
```

The reason behind this is `code obfuscation`. What is it? How is it done? Where is it used?