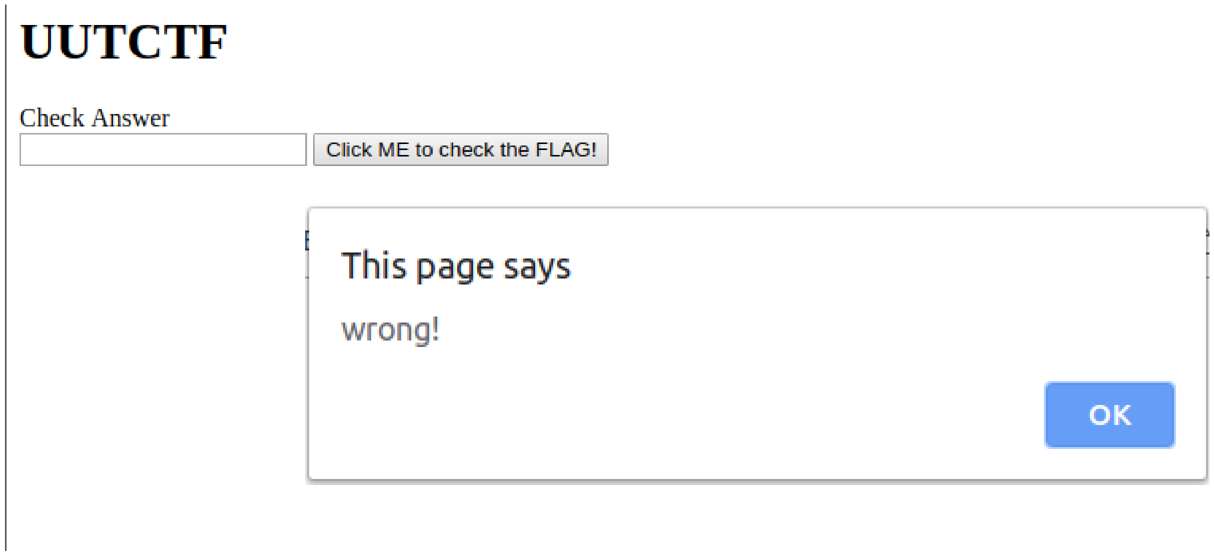
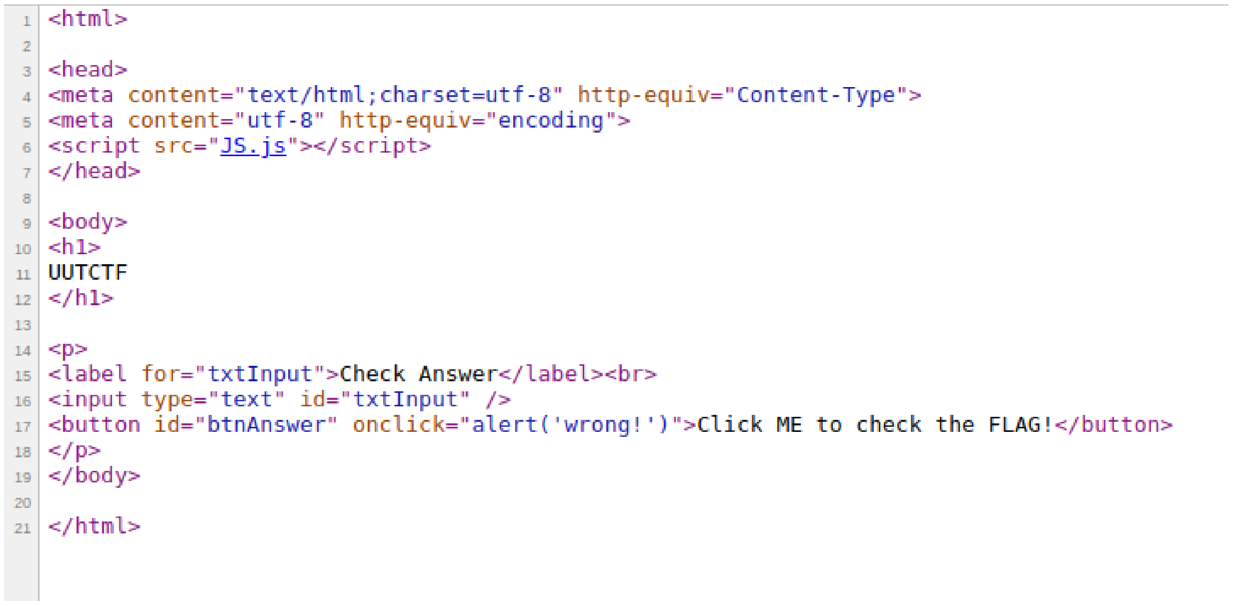
Bad JS (100)

“There is a bad JS which hides flag inside. Capture it.”

First thing is to download and extract the 7-zip archive into the directory of your choosing. Once extracted, open up index.html (I am using Chrome). We are greeted with a simple flag check application.

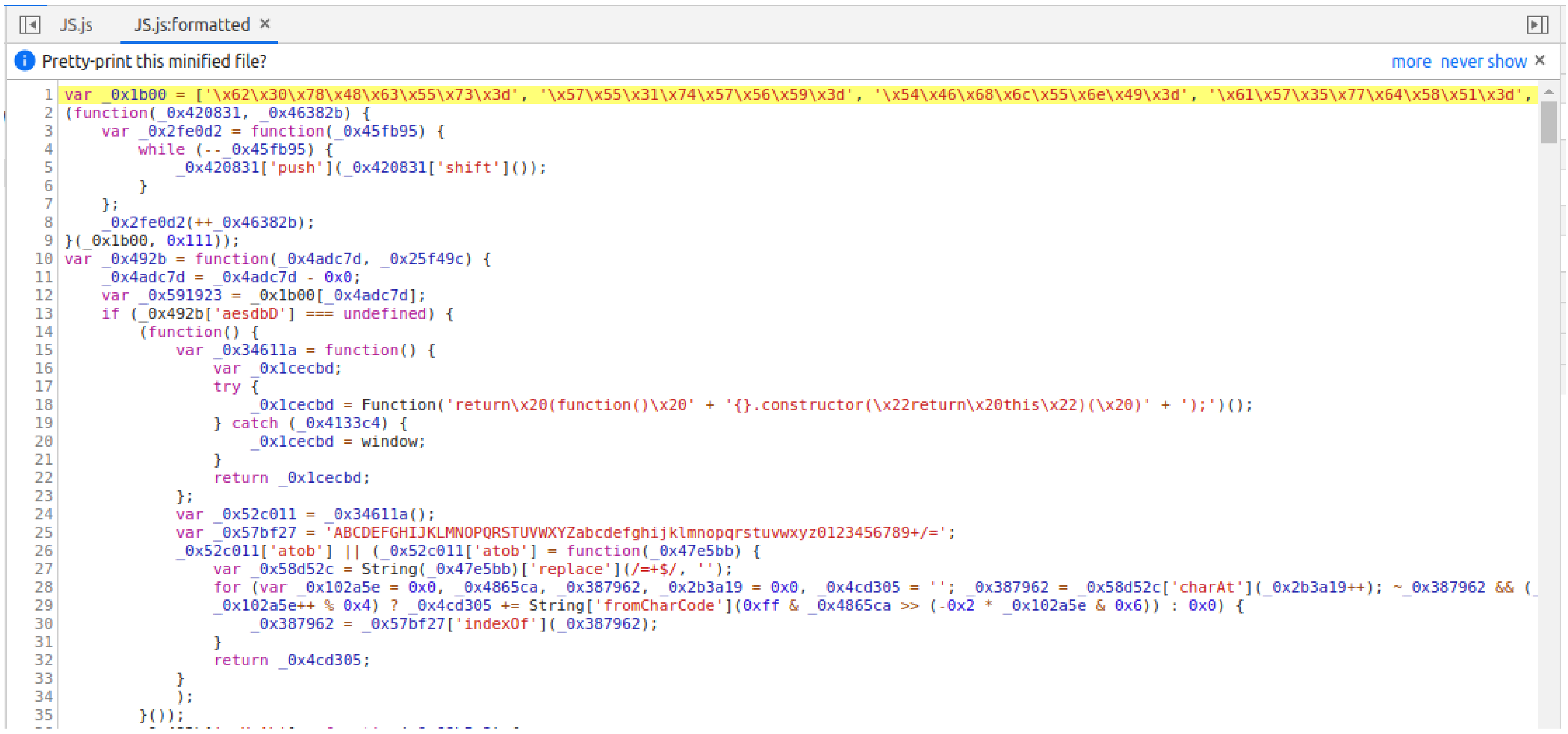


No matter what I entered, the result was the same (An alert box with the phrase “wrong!”). So let's take a quick look at the source code.



```html
<html>

<head>
<meta content="text/html;charset=utf-8" http-equiv="Content-Type">
<meta content="utf-8" http-equiv="encoding">
<script src="JS.js"></script>
</head>

<body>
<h1>
UUTCTF
</h1>

<p>
<label for="txtInput">Check Answer</label><br>
<input type="text" id="txtInput" />
<button id="btnAnswer" onclick="alert('wrong!')">Click ME to check the FLAG!</button>
</p>
</body>

</html>
```

We now know why we always get the same alert, no matter what we enter. The form input is not actually checked against anything. The next logical place to look would be the javascript source file JS.js.



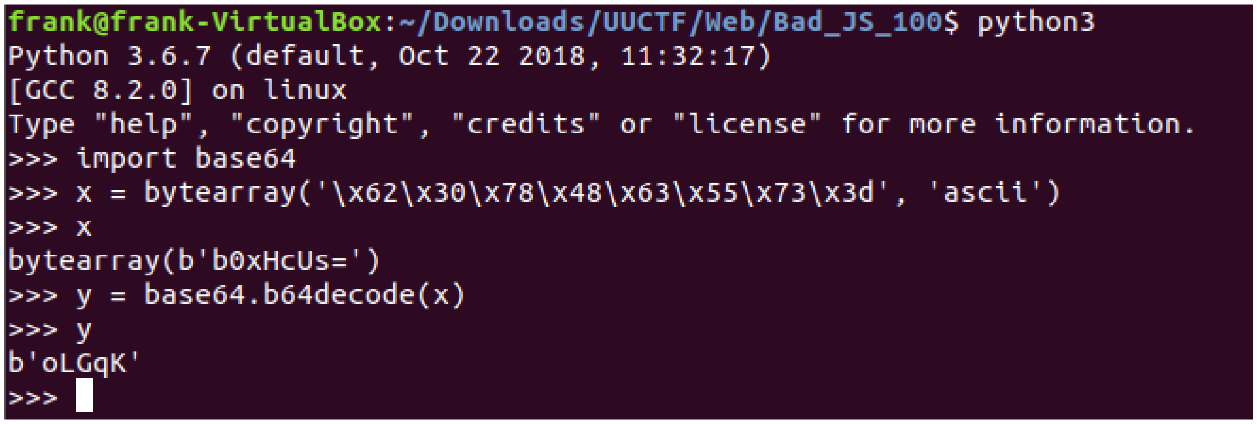
JS.js JS.js:formatted ×

Pretty-print this minified file? more never show ×

```javascript
var _0x1b00 = ['\x62\x30\x78\x48\x63\x55\x73\x3d', '\x57\x55\x31\x74\x57\x56\x59\x3d', '\x54\x46\x68\x6c\x55\x6e\x49\x3d', '\x61\x57\x35\x77\x64\x58\x51\x3d',
(function(_0x420831, _0x46382b) {
    var _0x2fe0d2 = function(_0x45fb95) {
        while (--_0x45fb95) {
            _0x420831['push'](_0x420831['shift']());
        }
    };
    _0x2fe0d2(++_0x46382b);
}(_0x1b00, 0x111));
var _0x492b = function(_0x4adc7d, _0x25f49c) {
    _0x4adc7d = _0x4adc7d - 0x0;
    var _0x591923 = _0x1b00[_0x4adc7d];
    if (_0x492b['aesdbD'] === undefined) {
        (function() {
            var _0x34611a = function() {
                var _0x1cecbd;
                try {
                    _0x1cecbd = Function('return\x20(function()\x20' + '{}.constructor(\x22return\x20this\x22)(\x20)' + ');')();
                } catch (_0x4133c4) {
                    _0x1cecbd = window;
                }
                return _0x1cecbd;
            };
            var _0x52c011 = _0x34611a();
            var _0x57bf27 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';
            _0x52c011['atob'] || (_0x52c011['atob'] = function(_0x47e5bb) {
                var _0x58d52c = String(_0x47e5bb)['replace'](/=+$/, '');
                for (var _0x102a5e = 0x0, _0x4865ca, _0x387962, _0x2b3a19 = 0x0, _0x4cd305 = ''; _0x387962 = _0x58d52c['charAt'](_0x2b3a19++); ~_0x387962 && (_
                _0x102a5e++ % 0x4) ? _0x4cd305 += String['fromCharCode'](0xff & _0x4865ca >> (-0x2 * _0x102a5e & 0x6)) : 0x0) {
                    _0x387962 = _0x57bf27['indexOf'](_0x387962);
                }
                return _0x4cd305;
            }
            );
        }());
```

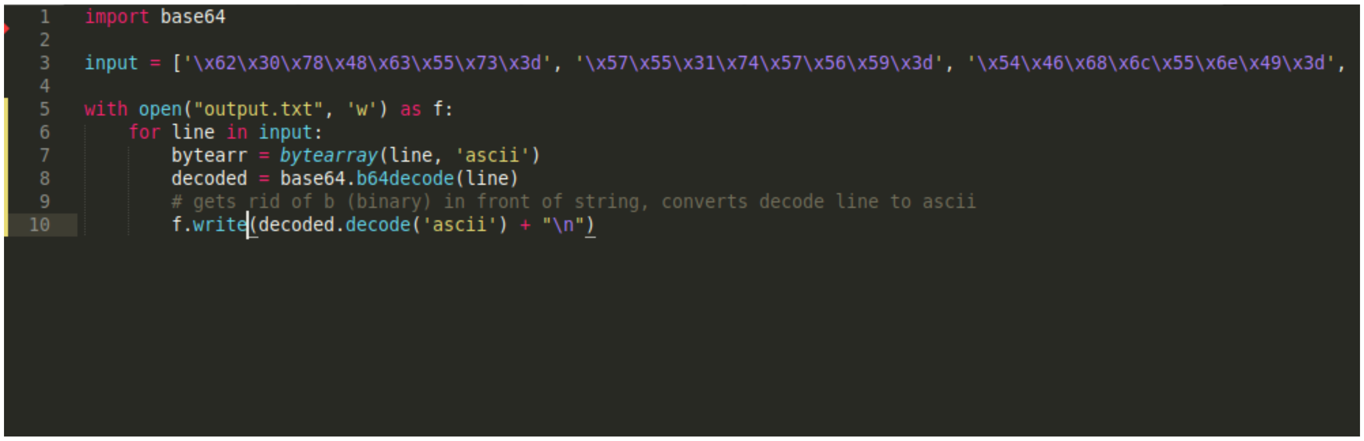
The entire file seems to be stripped of symbols. After a quick and painful look through, I had no leads on how to deal with it, but I had a feeling the solution was a little easier than reversing stripped javascript. That is when I noticed the first line of the file, **var_0x1b00**. The odd thing was, that every bytes string in the array ended in \x3d (which I knew was hex for '='). That may mean that the byte strings were base64 encoded, then hex encoded, so all we would have to do is decode them.

Here we can see that is is the case. That the strings are base64 encoded.



```
frank@frank-VirtualBox:~/Downloads/UUCTF/Web/Bad_JS_100$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import base64
>>> x = bytearray('\x62\x30\x78\x48\x63\x55\x73\x3d', 'ascii')
>>> x
bytearray(b'b0xHcUs=')
>>> y = base64.b64decode(x)
>>> y
b'oLGqK'
>>> 
```

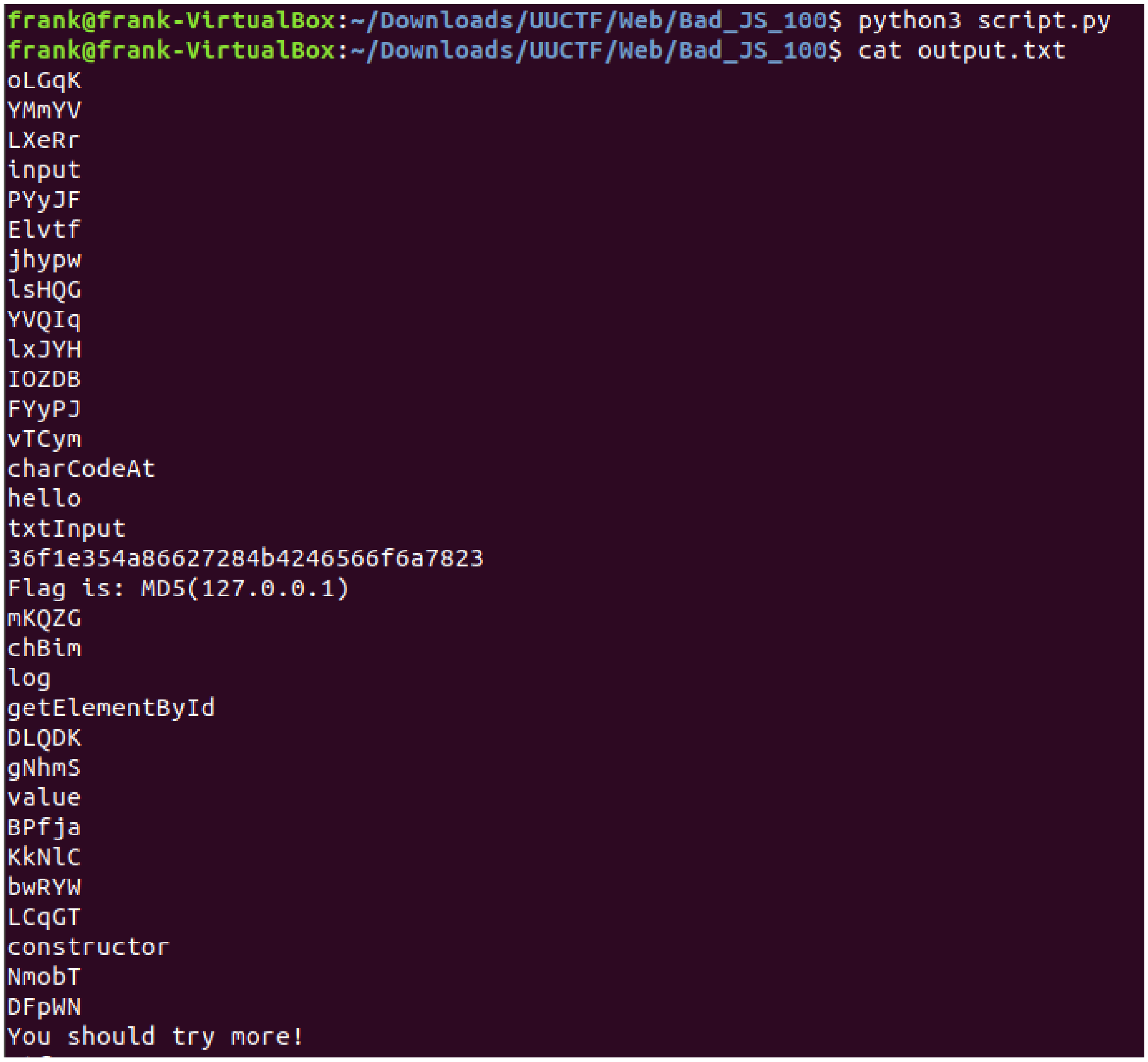
I wrote a python script that takes the byte array and converts each string in it into plain text. I wrote these plain text values to a file for reading.



```python
import base64

input = ['\x62\x30\x78\x48\x63\x55\x73\x3d', '\x57\x55\x31\x74\x57\x56\x59\x3d', '\x54\x46\x68\x6c\x55\x6e\x49\x3d',

with open("output.txt", 'w') as f:
    for line in input:
        bytearr = bytearray(line, 'ascii')
        decoded = base64.b64decode(line)
        # gets rid of b (binary) in front of string, converts decode line to ascii
        f.write(decoded.decode('ascii') + "\n")
```

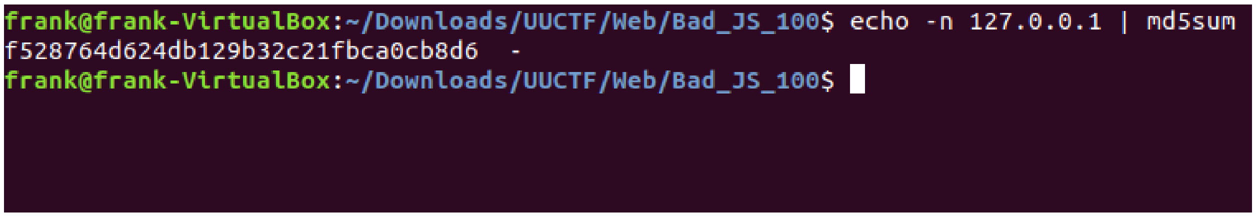
The results were just as I hoped for. It seemed that the byte strings were one of two things. 1.) A missing symbol from the javascript source. 2.) Random strings of 5 characters

Output File:



```
frank@frank-VirtualBox:~/Downloads/UUCTF/Web/Bad_JS_100$ python3 script.py
frank@frank-VirtualBox:~/Downloads/UUCTF/Web/Bad_JS_100$ cat output.txt
oLGqK
YMmYV
LXeRr
input
PYyJF
Elvtf
jhypw
lsHQG
YVQIq
lxJYH
IOZDB
FYyPJ
vTCym
charCodeAt
hello
txtInput
36f1e354a86627284b4246566f6a7823
Flag is: MD5(127.0.0.1)
mKQZG
chBim
log
getElementById
DLQDK
gNhmS
value
BPfja
KkNlC
bwRYW
LCqGT
constructor
NmobT
DFpWN
You should try more!
```

Now we know that the flag is the MD5 hash of 127.0.0.1. Plenty of websites can find this hash in a jiffy, but I opted for bash.



```
frank@frank-VirtualBox:~/Downloads/UUCTF/Web/Bad_JS_100$ echo -n 127.0.0.1 | md5sum
f528764d624db129b32c21fbca0cb8d6  -
frank@frank-VirtualBox:~/Downloads/UUCTF/Web/Bad_JS_100$
```

The -n flag is needed, otherwise md5 will think the \n is part of the string. That hash (without spaces and - at the end) wrapped inside UUTCTF{} is the flag. I was happy I thought to decode those strings early on, or I may still be looking at that stripped JS code at this moment. This is my first writeup I tried to format for this site, so I hope it does not look like complete trash.