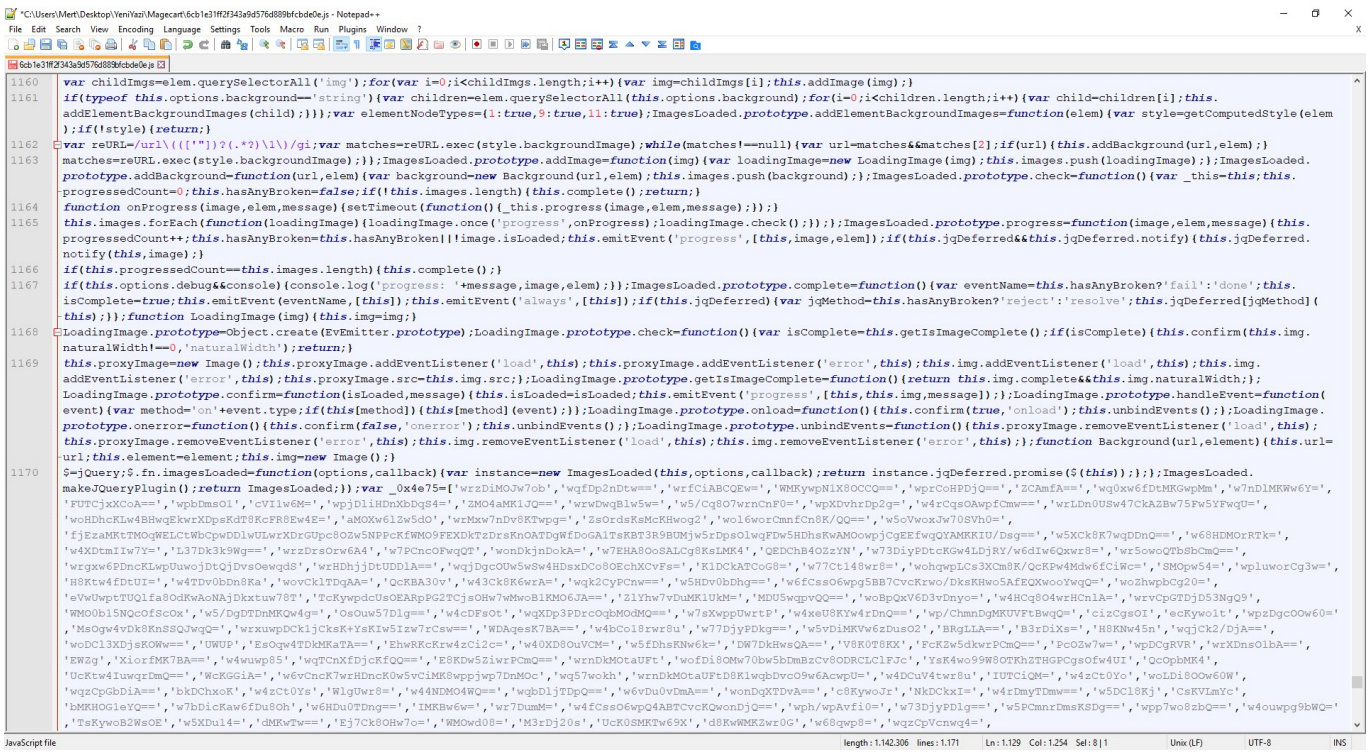
Magecart Analysis

written by Mert SARICA | 4 April 2020

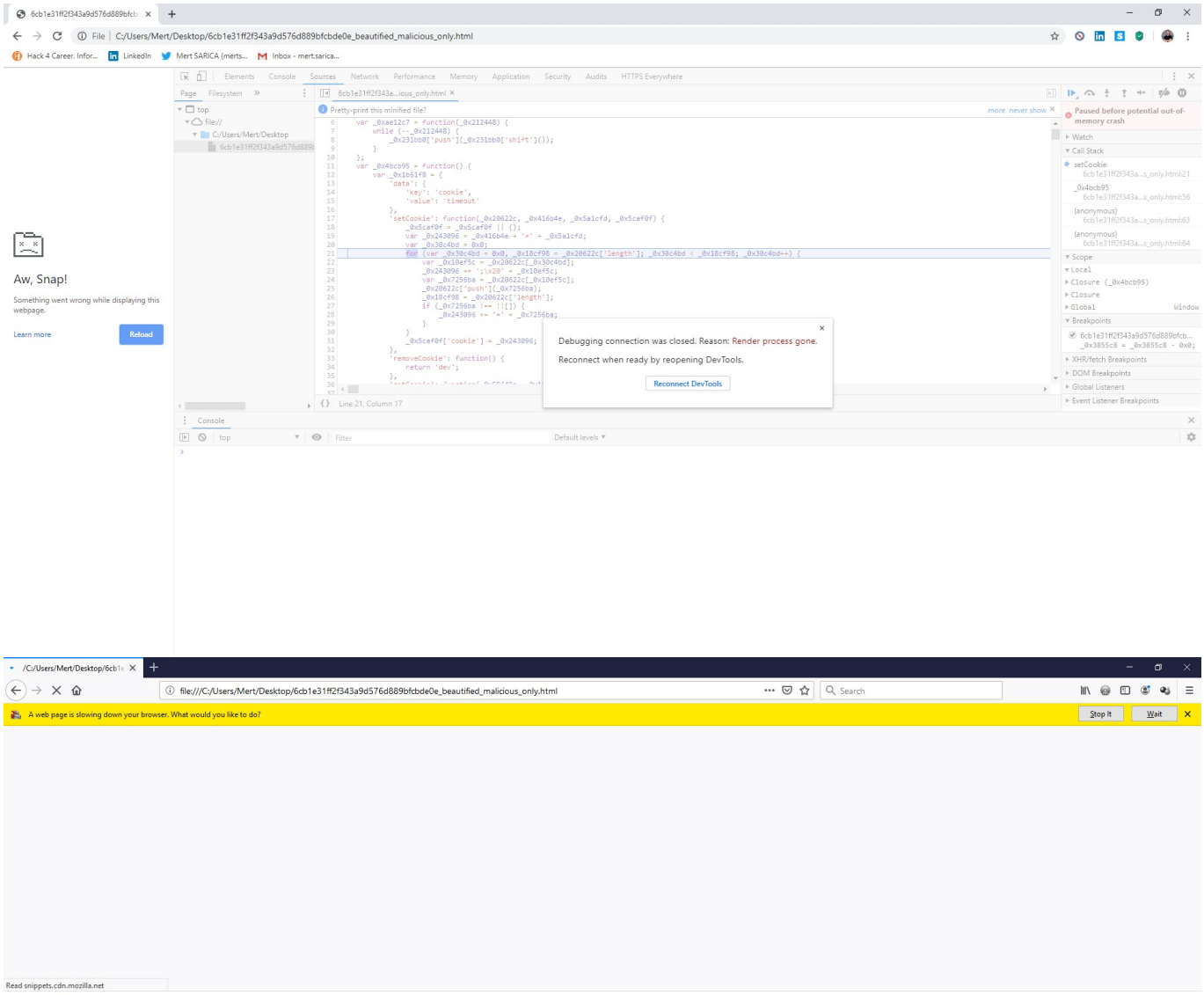
As you may remember, in my blog post “Fighting Against Magecart” I mentioned that I would cover the analysis of malicious JavaScript code in another article. Until now, I have analyzed malicious JavaScript code many times, and about 3 years ago, I also wrote a blog post titled “Malicious JavaScript Analysis“. Of course, as years passed, the methods used by threat actors changed and the work of cybersecurity analysts and researchers became increasingly more difficult.

When I first came across the malicious JavaScript code (6cb1e31ff2f343a9d576d889bfcbde0e.js) developed by the Magecart group, I immediately noticed that the code had been made too complex to easily understand, it was likely that one of the JavaScript Obfuscator or JavaScript Obfuscator Tool tools had been used. I thought that I could easily overcome this complexity by using tools like de4js and IlluminateJs and at worst, I could reach a happy ending by doing dynamic code analysis (debugging). But things didn’t turn out as planned. :)



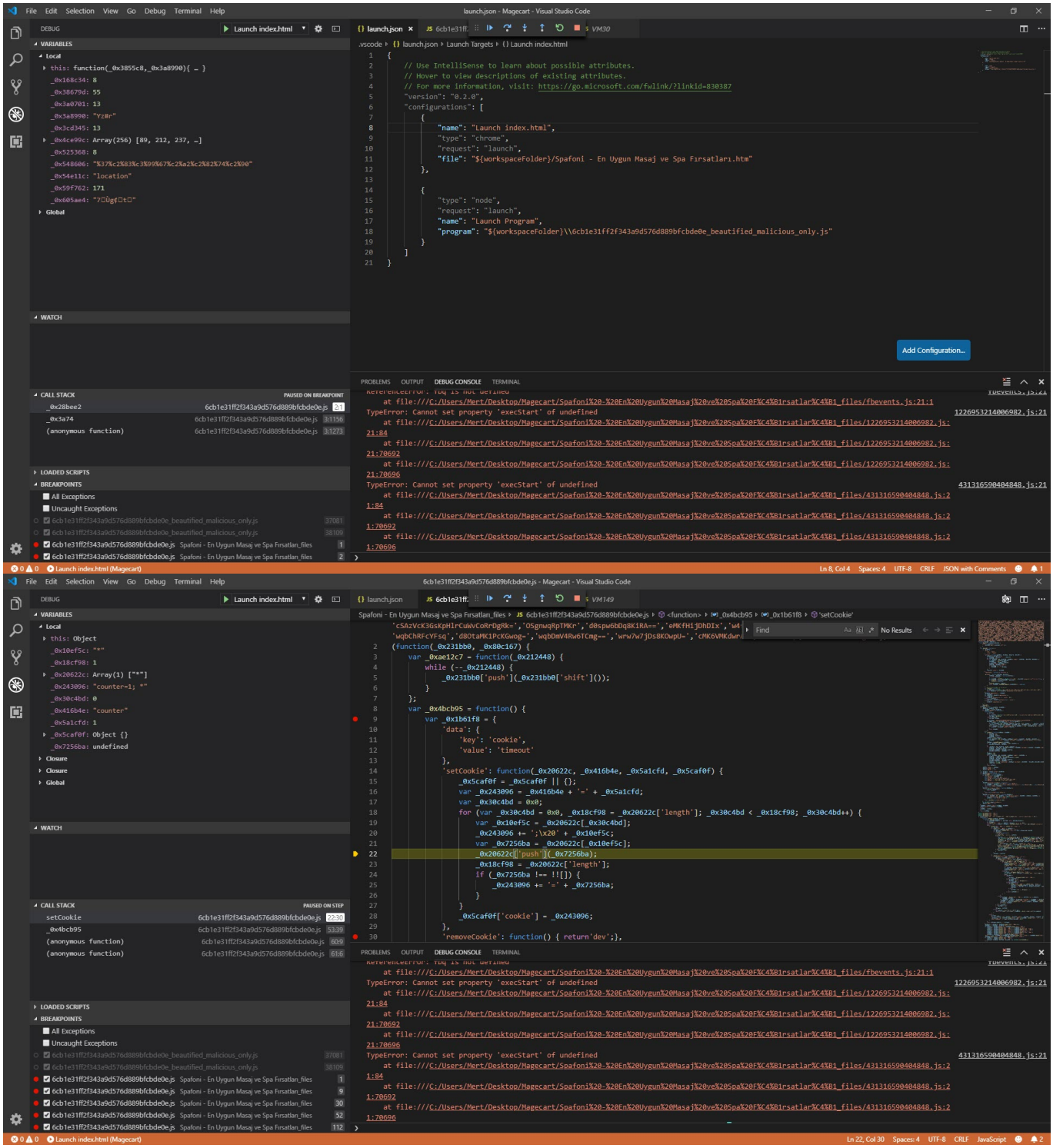
First, I used the JavaScript Beautifier website to make the malicious code block readable. Then, I tried to use the de4js and IlluminateJs tools in succession to make the code more understandable, but I failed. I started

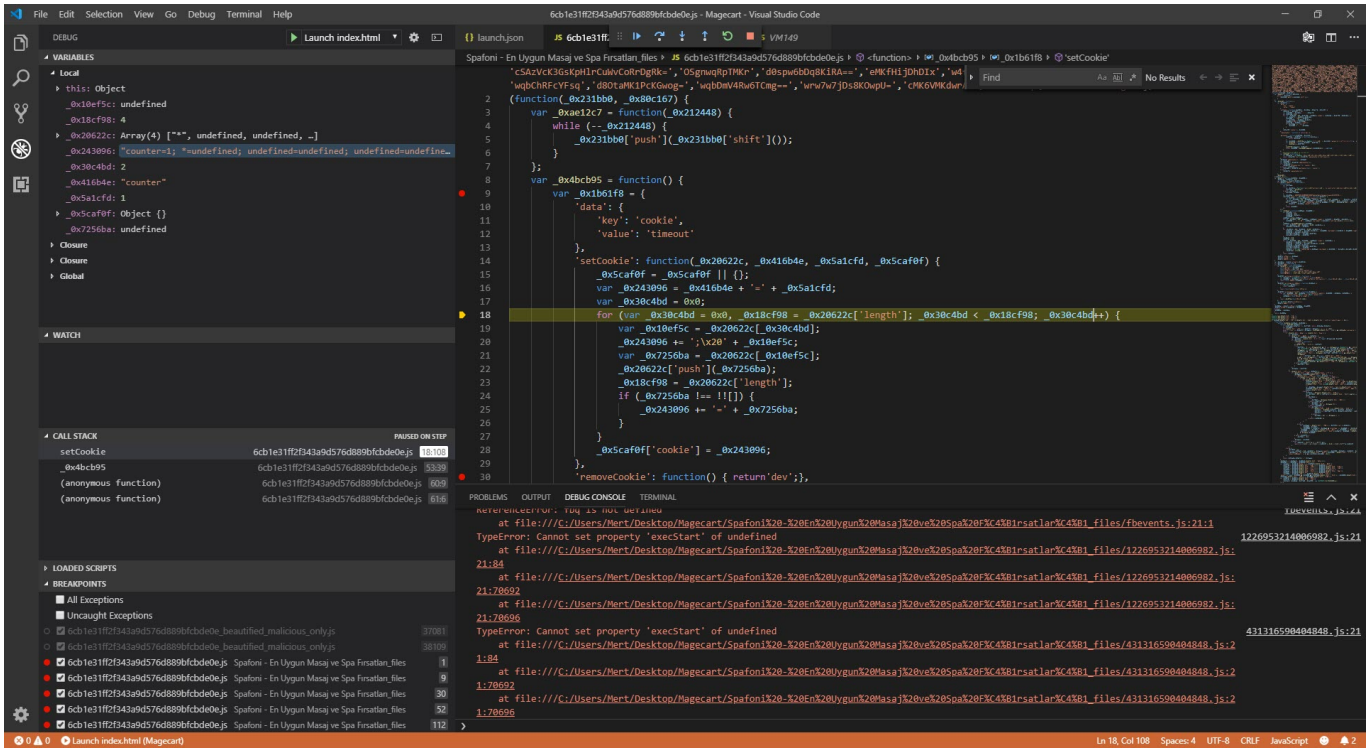
analyzing the malicious JavaScript code with the Chrome DevTools by doing debugging and soon realized that things were not going well. I thought that the problem might be caused by Chrome and decided to try my luck with Firefox, but it also gave a warning that something was not right.



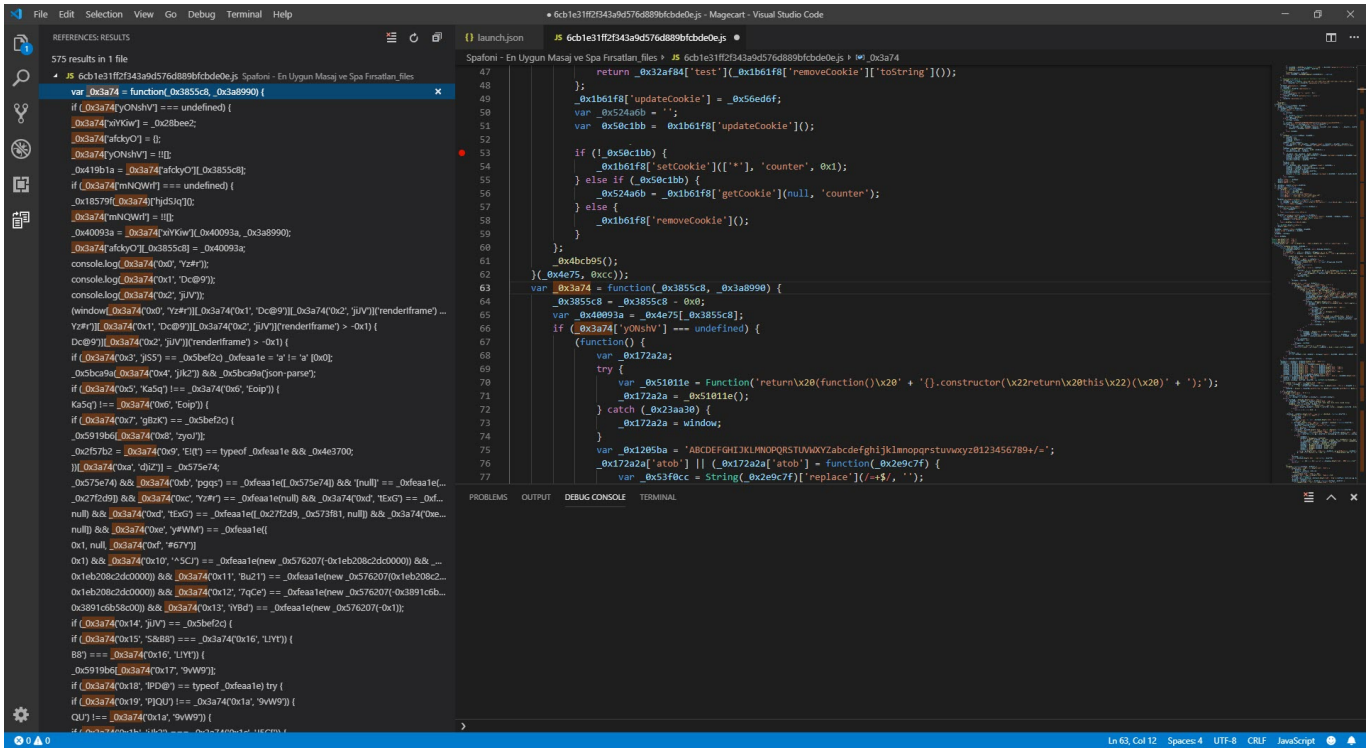
As I was thinking about what to do, I started researching the possibility of debugging with a different tool instead of a web browser and came across the Visual Studio Code source code editor. When I started debugging with this editor, which allows for the analysis of HTML and JavaScript code in the

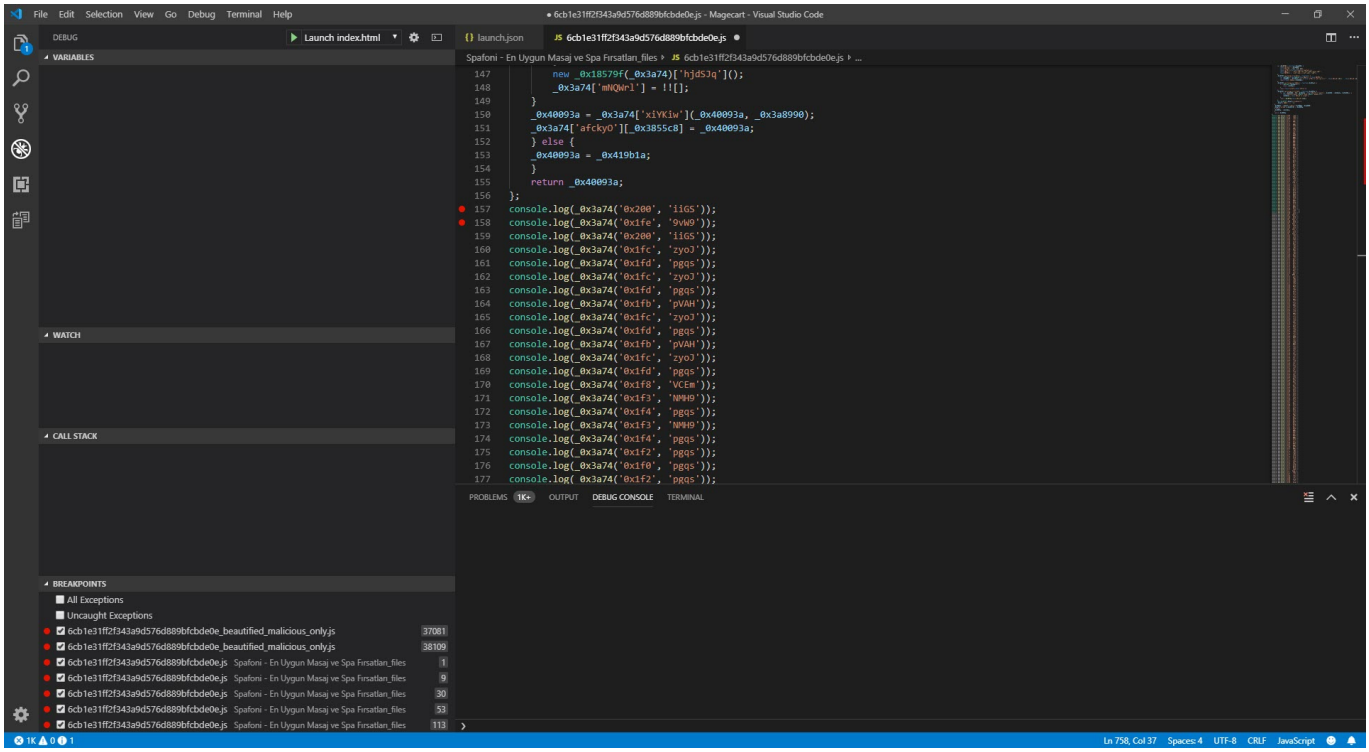
background through the Chrome debugging extension and has many plugins, I saw that the function associated with SetCookie was creating many arrays, consuming the available space in memory, and making the debugging ineffective (self-defending).



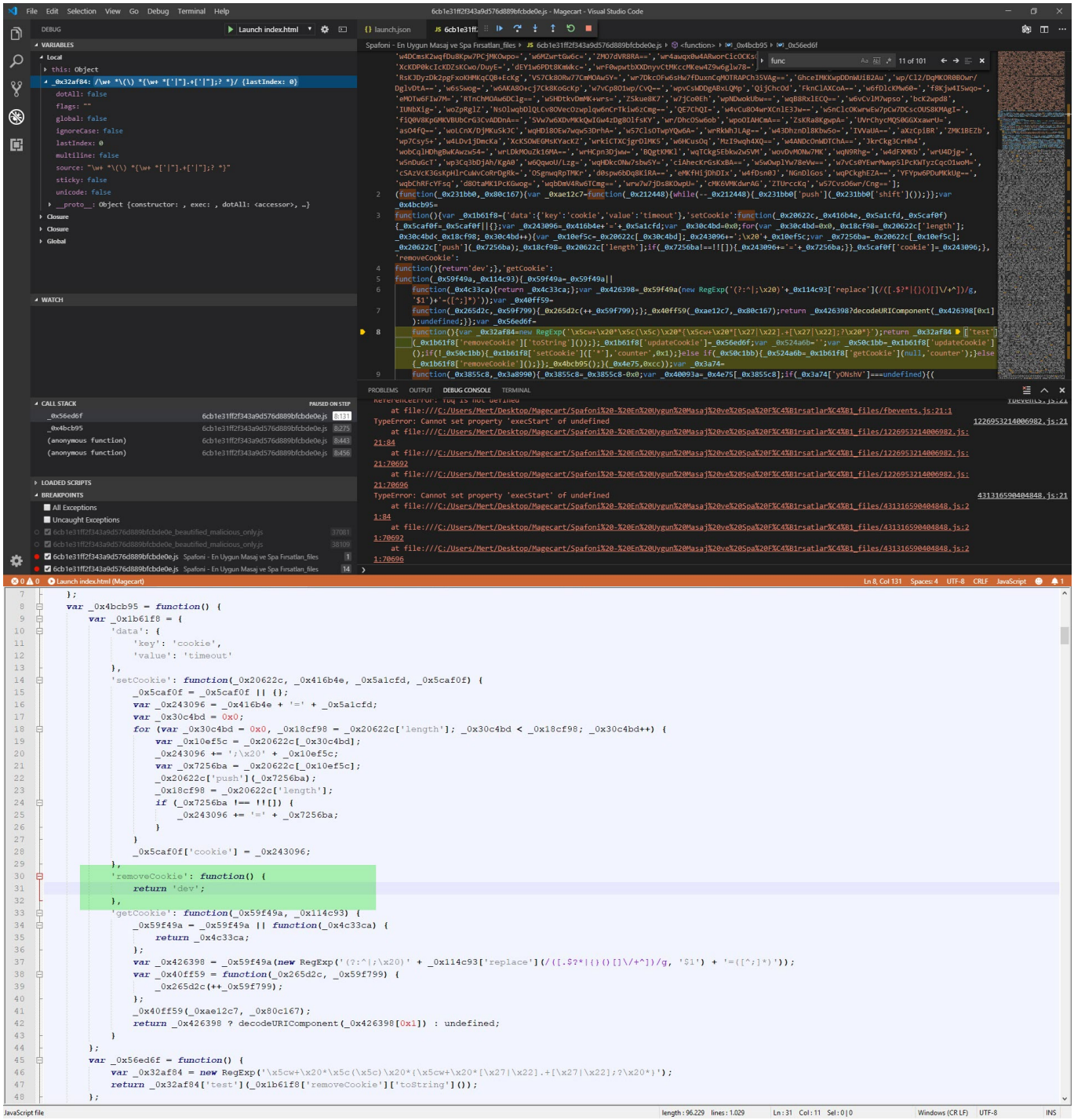


I assumed that because malicious actors intended for this code to work seamlessly in the web browser, there were controls in the code for debugging, and began analyzing each function step by step. My ultimate goal was not to analyze the code dynamically from beginning to end, but to find out which website the stolen information was sent to and to decode the hidden character strings. So, I progressed by starting from the _0x3a74 function used to decode the hidden strings.

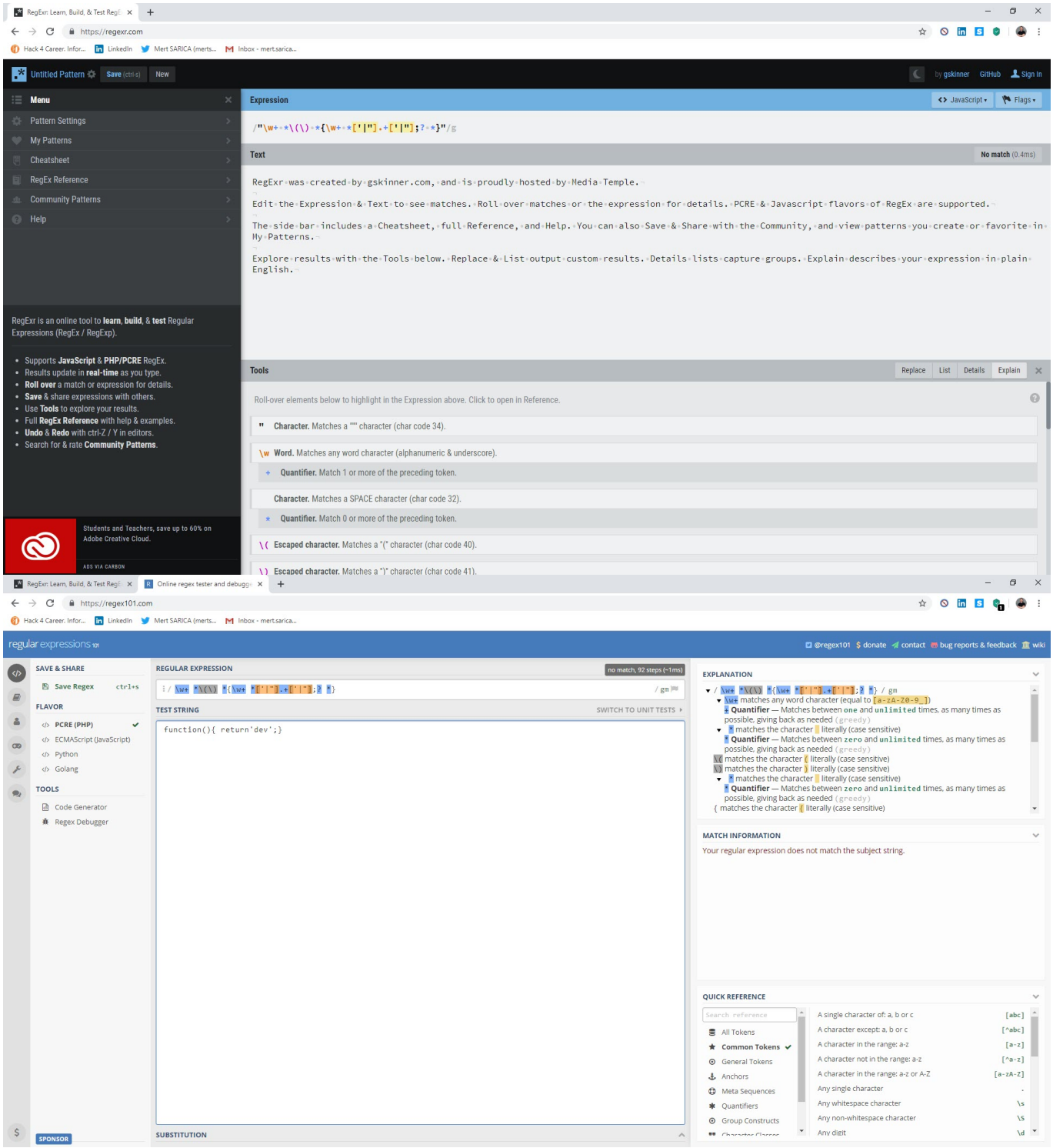


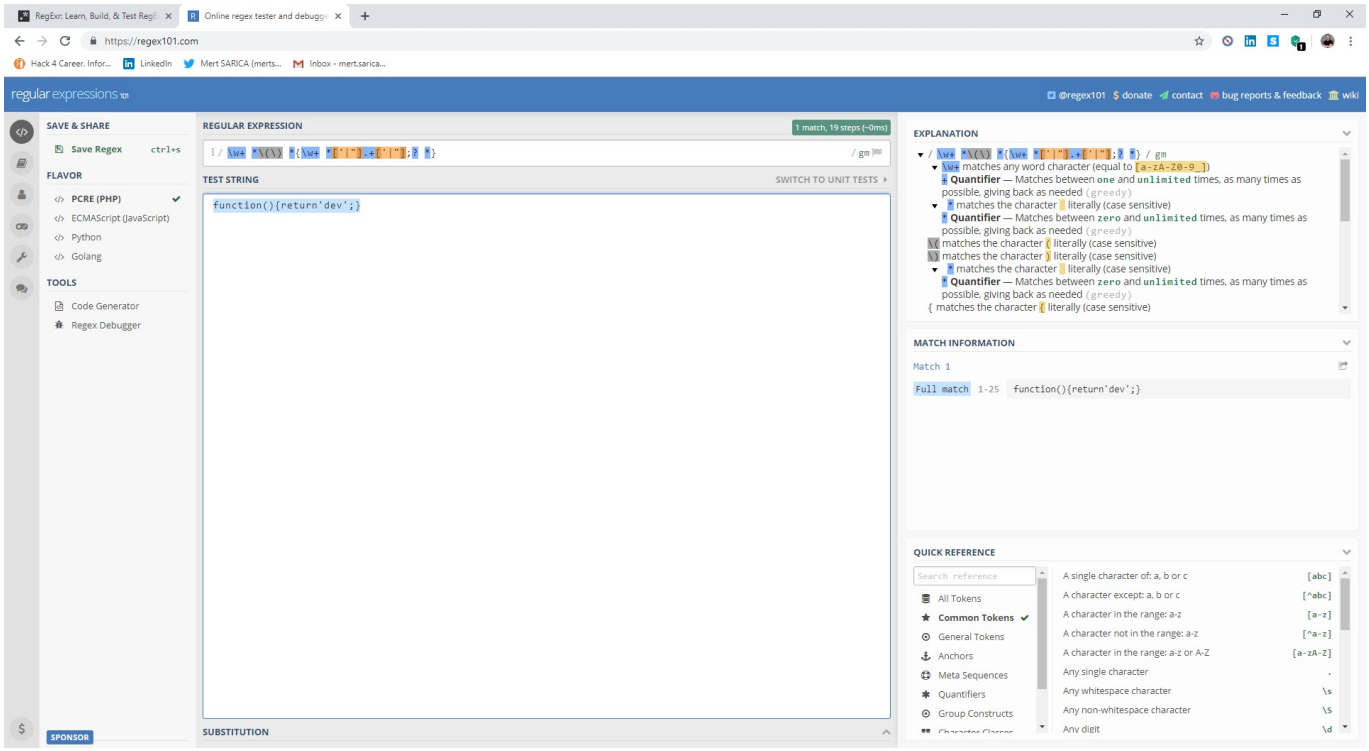


While analyzing, I noticed that a check for a space between the { sign and the return keyword was being made with Regex in the removeCookie value. When a space character was detected, the code flow would proceed to the function that was causing problems by creating many arrays, as mentioned above. So why did the malicious developer put such a control? When analysts encounter such complex, unreadable codes, the first thing they do is to use tools (like JavaScript Beautifier) to make the code readable and properly formatted, these tools automatically insert spaces and this creates a great detection mechanism for the malicious actors that code is being analyzed.

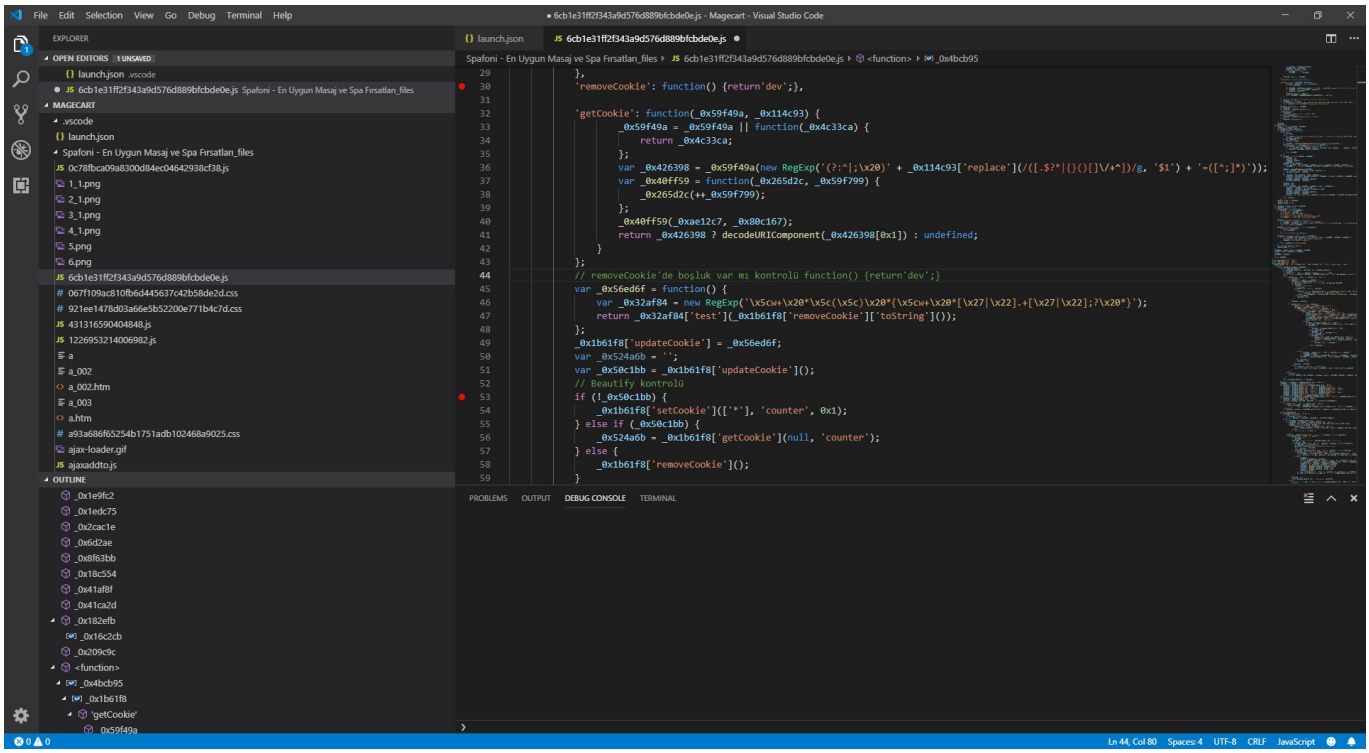


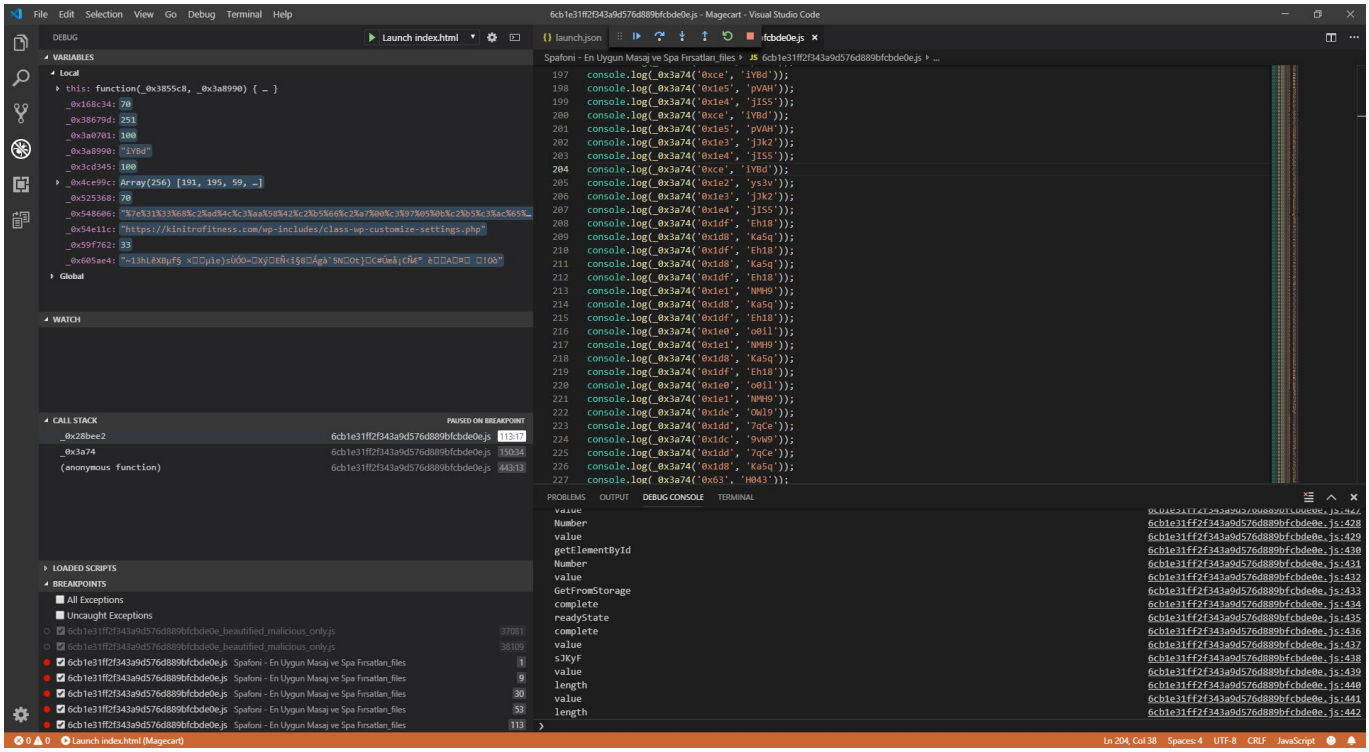

```
};
var _0x4bcb95 = function() {
    var _0x1b61f8 = {
        'data': {
            'key': 'cookie',
            'value': 'timeout'
        },
        'setCookie': function(_0x20622c, _0x416b4e, _0x5a1cfd, _0x5caf0f) {
            _0x5caf0f = _0x5caf0f || {};
            var _0x243096 = _0x416b4e + '=' + _0x5a1cfd;
            var _0x30c4bd = 0x0;
            for (var _0x30c4bd = 0x0, _0x18cf98 = _0x20622c['length']; _0x30c4bd < _0x18cf98; _0x30c4bd++) {
                var _0x10ef5c = _0x20622c[_0x30c4bd];
                _0x243096 += ';\x20' + _0x10ef5c;
                var _0x7256ba = _0x20622c[_0x10ef5c];
                _0x20622c['push'](_0x7256ba);
                _0x18cf98 = _0x20622c['length'];
                if (_0x7256ba !== !![]) {
                    _0x243096 += '=' + _0x7256ba;
                }
            }
            _0x5caf0f['cookie'] = _0x243096;
        },
        'removeCookie': function() {
            return 'dev';
        },
        'getCookie': function(_0x59f49a, _0x114c93) {
            _0x59f49a = _0x59f49a || function(_0x4c33ca) {
                return _0x4c33ca;
            };
            var _0x426398 = _0x59f49a(new RegExp('(?:^|;\x20)' + _0x114c93['replace'](/([.$?*|{}()[]\/+^])/g, '$1') + '=([^;]*)'));
            var _0x40ff59 = function(_0x265d2c, _0x59f799) {
                _0x265d2c(++_0x59f799);
            };
            _0x40ff59(_0xae12c7, _0x80c167);
            return _0x426398 ? decodeURIComponent(_0x426398[0x1]) : undefined;
        }
    };
    var _0x56ed6f = function() {
        var _0x32af84 = new RegExp('\x5cw+\x20*\x5c(\x5c)\x20*{\x5cw+\x20*[\x27|\x22].+[\x27|\x22];?\x20*}');
        return _0x32af84['test'](_0x1b61f8['removeCookie']['toString']());
    };
```


RegExr: Learn, Build, & Test Regex
https://regexr.com
Untitled Pattern
Save (ctrl-s)
New
by gskinner
GitHub
Sign In
Menu
Pattern Settings
My Patterns
Cheatsheet
RegEx Reference
Community Patterns
Help
Expression
JavaScript
Flags
/"\w+=\(\)*{\w+*['|"].+['|"];?*}"/g
Text
No match (0.4ms)
RegExr was created by gskinner.com, and is proudly hosted by Media Temple.
Edit the Expression & Text to see matches. Roll over matches or the expression for details. PCRE & Javascript flavors of RegEx are supported.
The side bar includes a Cheatsheet, full Reference, and Help. You can also Save & Share with the Community, and view patterns you create or favorite in My Patterns.
Explore results with the Tools below. Replace & List output custom results. Details lists capture groups. Explain describes your expression in plain English.
RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp).
Supports JavaScript & PHP/PCRE RegEx.
Results update in real-time as you type.
Roll over a match or expression for details.
Save & share expressions with others.
Use Tools to explore your results.
Full RegEx Reference with help & examples.
Undo & Redo with ctrl-Z / Y in editors.
Search for & rate Community Patterns.
Students and Teachers, save up to 60% on Adobe Creative Cloud.
ADS VIA CARBON
Tools
Replace
List
Details
Explain
Roll-over elements below to highlight in the Expression above. Click to open in Reference.
" Character. Matches a "" character (char code 34).
\w Word. Matches any word character (alphanumeric & underscore).
+ Quantifier. Match 1 or more of the preceding token.
Character. Matches a SPACE character (char code 32).
* Quantifier. Match 0 or more of the preceding token.
\(Escaped character. Matches a "(" character (char code 40).
\) Escaped character. Matches a ")" character (char code 41).
RegExr: Learn, Build, & Test Regex
Online regex tester and debugger
https://regex101.com
regular expressions 101
@regex101
donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Save Regex ctrl+s
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION
no match, 92 steps (~1ms)
/ \w+ *\(\) *{\w+ *['|"].+['|"];? *} / gm
TEST STRING
SWITCH TO UNIT TESTS
function(){ return'dev';}
SUBSTITUTION
EXPLANATION
/ \w+ *\(\) *{\w+ *['|"].+['|"];? *} / gm
\w+ matches any word character (equal to [a-zA-Z0-9_])
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
* matches the character literally (case sensitive)
* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
\(matches the character (literally (case sensitive)
\) matches the character) literally (case sensitive)
* matches the character literally (case sensitive)
* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
{ matches the character { literally (case sensitive)
MATCH INFORMATION
Your regular expression does not match the subject string.
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
Quantifiers
Group Constructs
A single character of: a, b or c [abc]
A character except: a, b or c [^abc]
A character in the range: a-z [a-z]
A character not in the range: a-z [^a-z]
A character in the range: a-z or A-Z [a-zA-Z]
Any single character .
Any whitespace character \s
Any non-whitespace character \S
Any digit \d



I have detected that credit card information (CVV, Holder, ccexpiry, ccnumber, cvc, fullname) is being stolen and sent to the address https://kinitrofitness[.]com/wp-includes/class-wp-customize-settings.php by editing it without spaces and solving hidden character strings through Regex control and static and dynamic code analysis.





Hope to see you in the following articles.

Note:

1. This article also contains the solution for the Pi Hediyem Var #19 cybersecurity game.