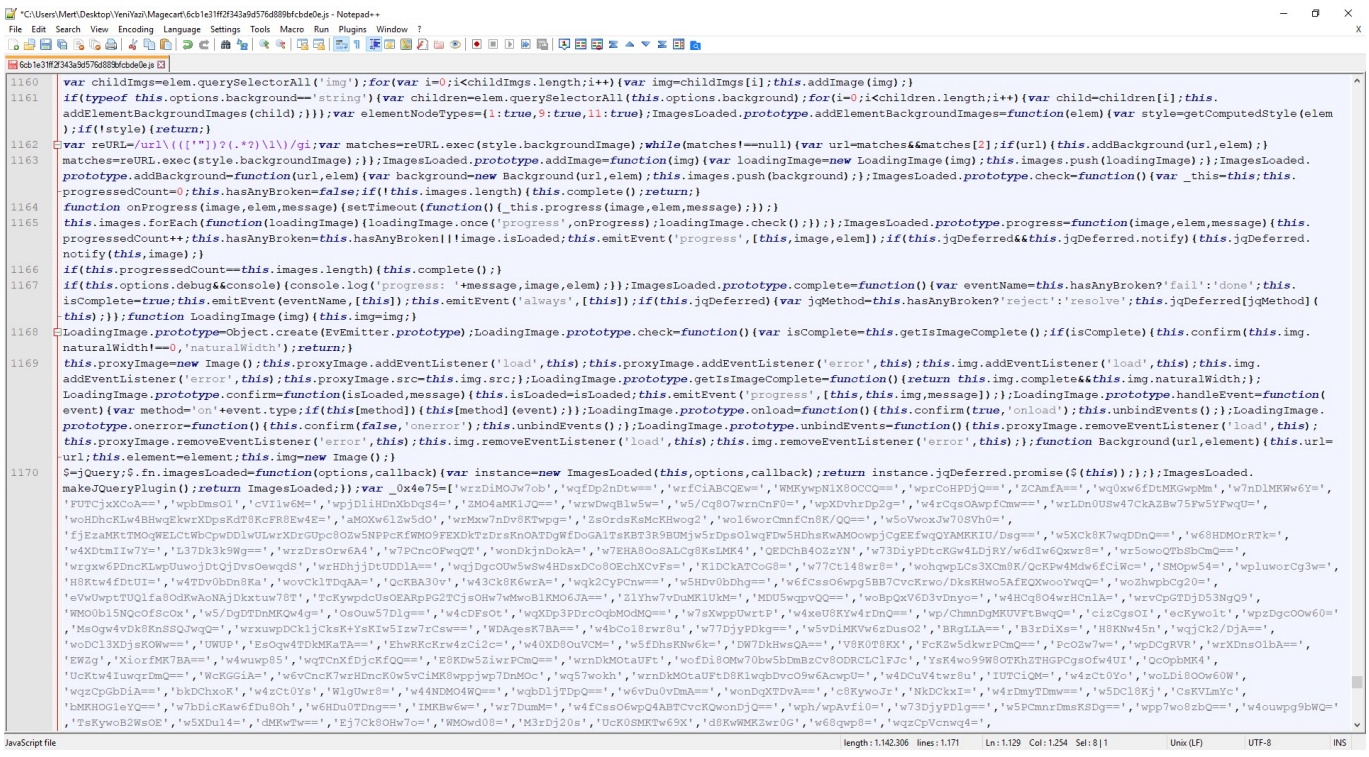
Magecart Analizi

written by Mert SARICA | 4 April 2020

If you are looking for an English version of this article, please visit here.

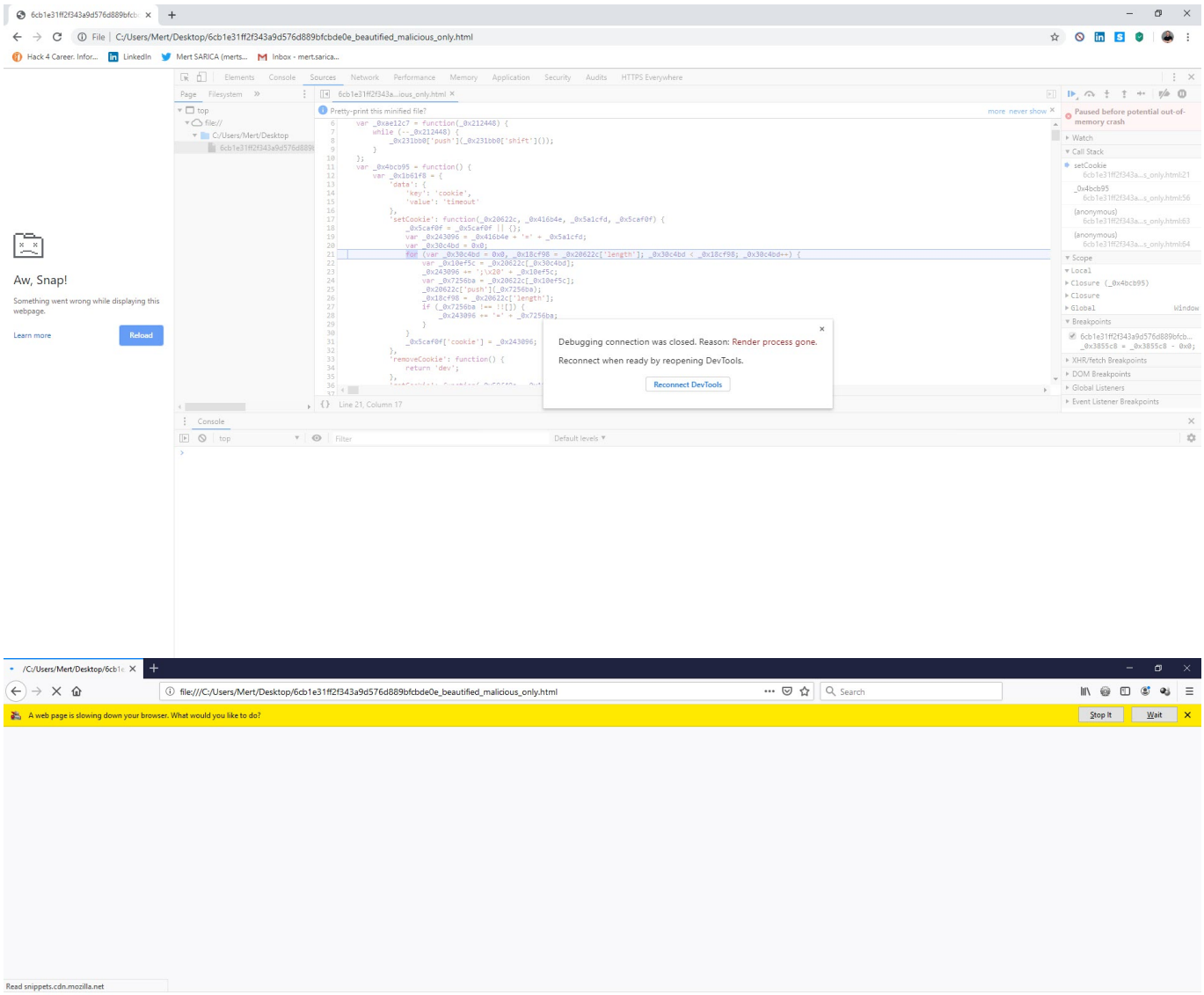
Hatırlarsanız Magecart ile Mücadele başlıklı blog yazımda zararlı JavaScript kodunun analizine başka bir yazımda yer vereceğimi belirtmiştim. Bu zamana dek çok defa zararlı javascript kodu analiz etmiş ve nasıl analiz edilebileceğine dair yaklaşık 3 yıl önce Zararlı JavaScript Analizi başlıklı bir blog yazısı da yazmıştım. Tabii yıllar geçtikçe tehdit aktörlerinin kullandığı yöntemler değişmeye ve siber güvenlik analistlerinin, araştırmacılarının işlerini git gide daha da zorlaştırmaya başladı.

Magecart grubunun geliştirdiği zararlı JavaScript kodu (6cb1e31ff2f343a9d576d889bfcbde0e.js) ile ilk karşılaştığımda kodun kolay bir şekilde anlaşılamayacak kadar karmaşıklaştırılmış (obfuscated) olduğu JavaScript Obfuscator veya JavaScript Obfuscator Tool araçlarından biri kullanılmış olabilir.) hemen dikkatimi çekmişti. de4js ve IlluminateJs gibi araçların da varlığına güvenerek karmaşıklaştırılmış kodu kolay bir şekilde alt edeceğimi, en kötü dinamik kod analizi yaparak (debugging) mutlu sona ulaşabileceğimi düşünüyordum fakat evdeki hesap çarşıya uymadı. :)



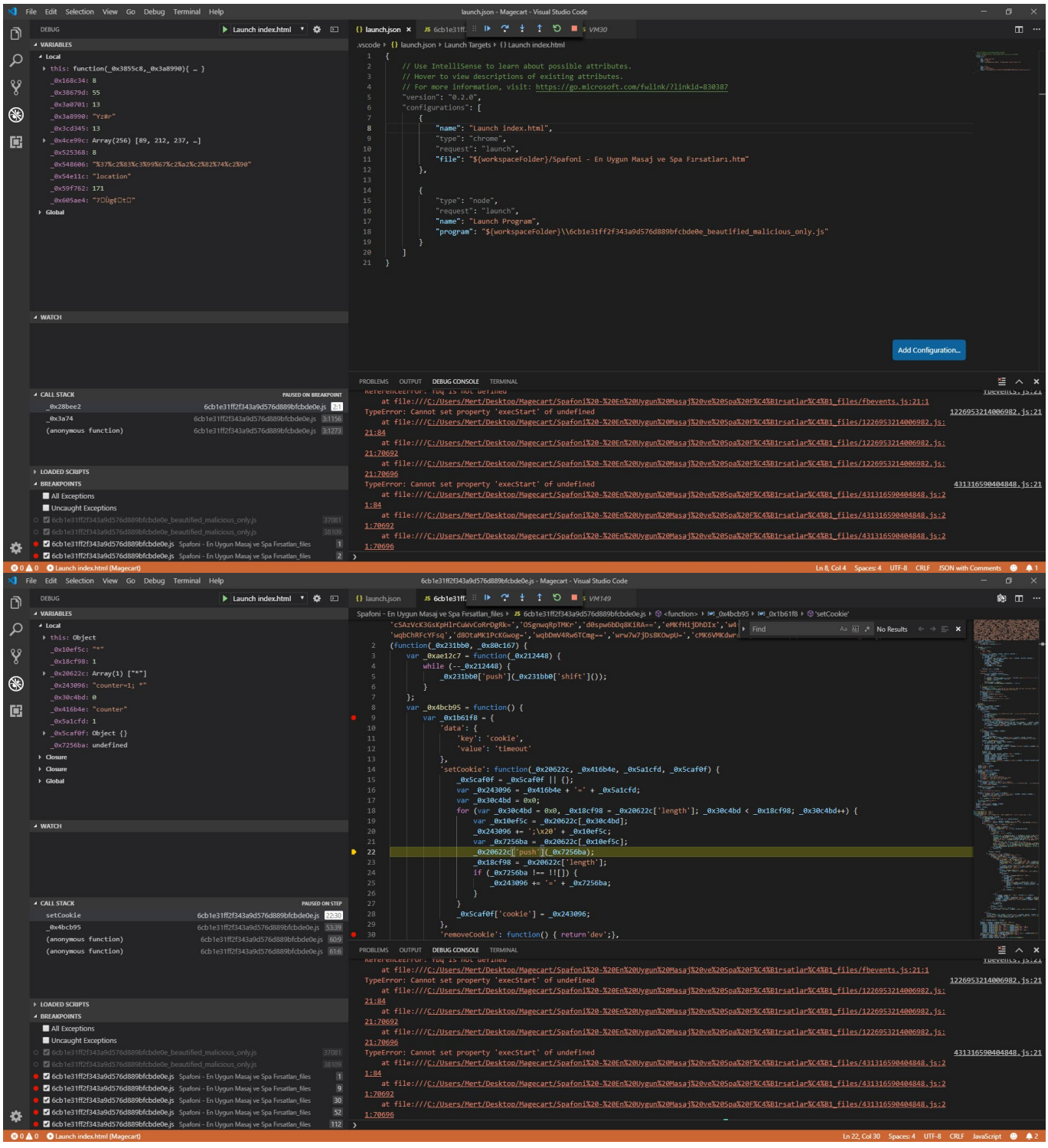
İlk iş olarak JavaScript Beautifier web sitesinden faydalanarak zararlı kod bloğunu okunaklı hale getirdim. Ardından kodu anlaşılır hale getirmek için

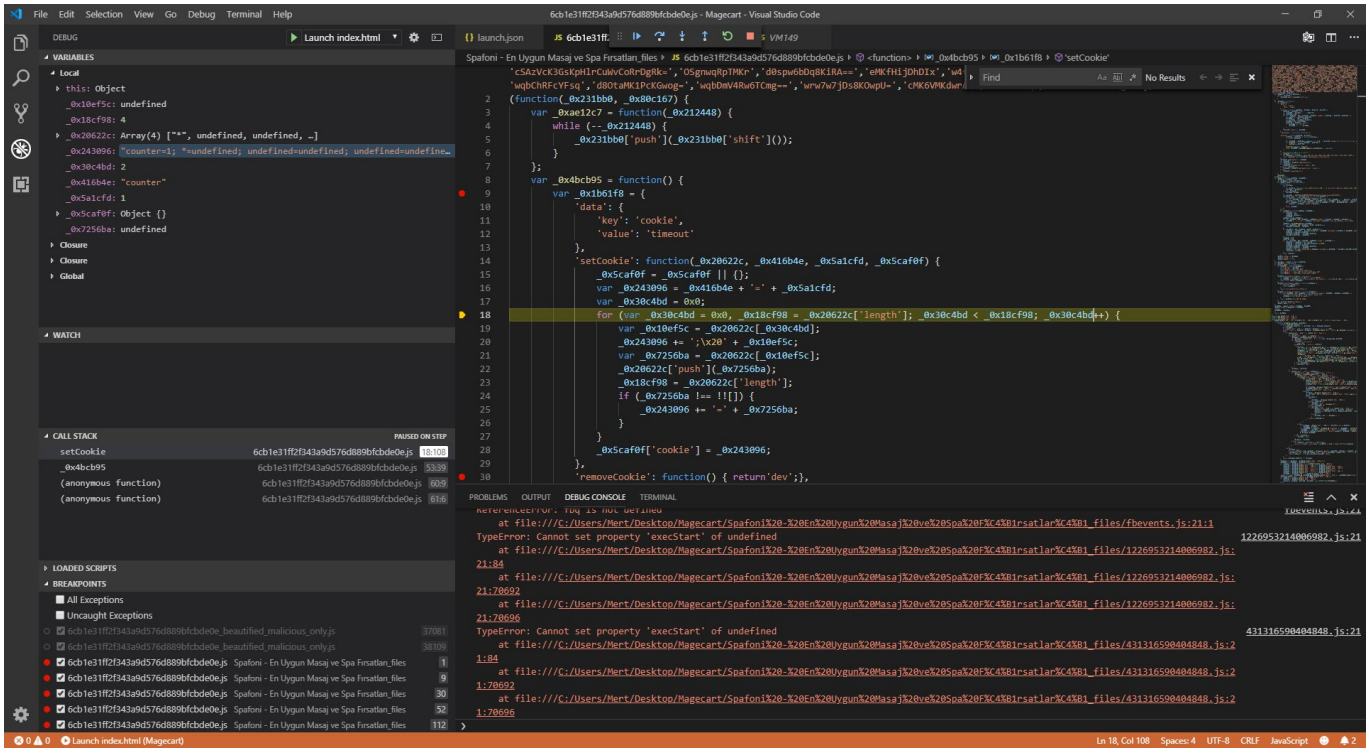
sırasıyla de4js ve IlluminateJs araçlarından faydalanmaya çalıştım ancak başarısız oldum. Chrome DevTools ile zararlı JavaScript kodunu hata ayıklaması (debugging) yaparak analiz etmeye başladım ve çok geçmeden bir zaman sonra işlerin yolunda gitmediğini farkettim. Chrome'dan kaynaklanan bir problem olabileceğini düşünerek şansımı Firefox ile denemeye karar verdim fakat o da birşeylerin yolunda gitmediğine dair uyarı verdi.



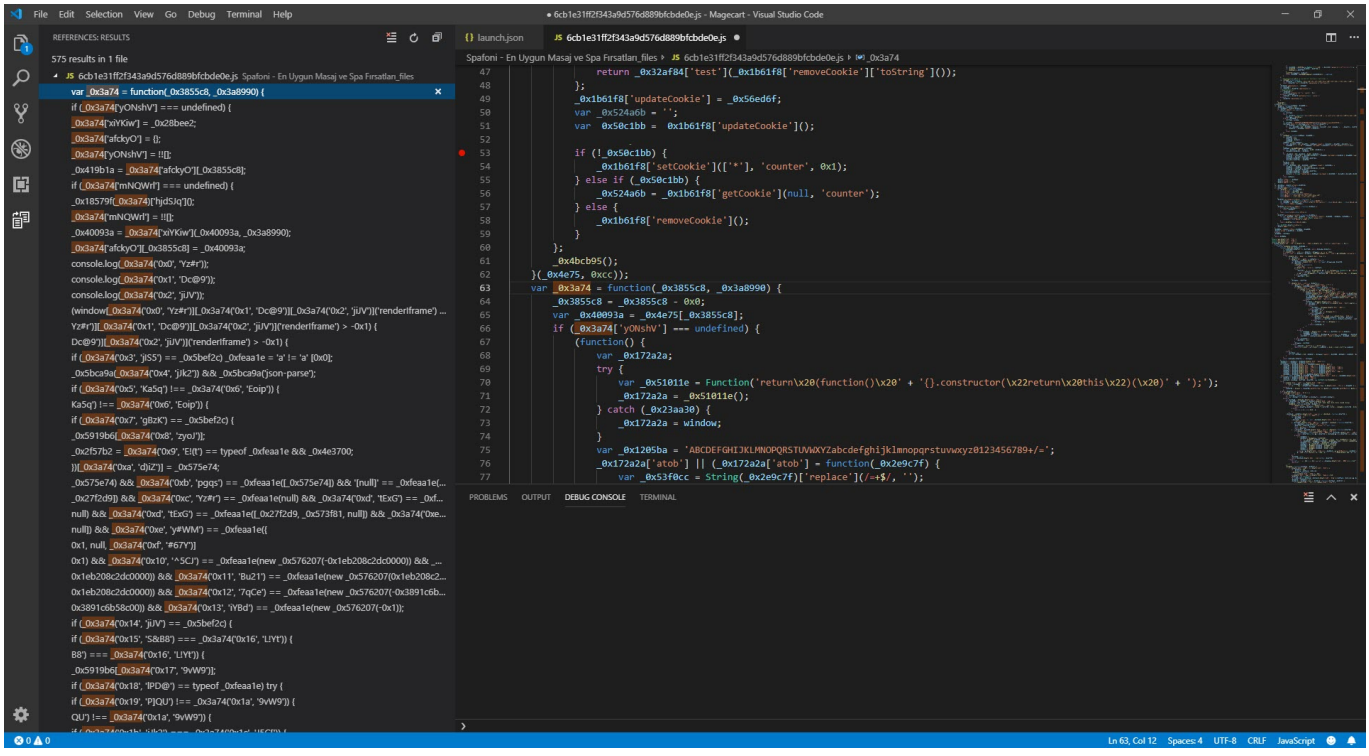
Ne yapabileceğime dair hindi gibi düşünüp dururken internet tarayıcısı yerine farklı bir araç ile hata ayıklaması yapabilmek için araştırma yapmaya

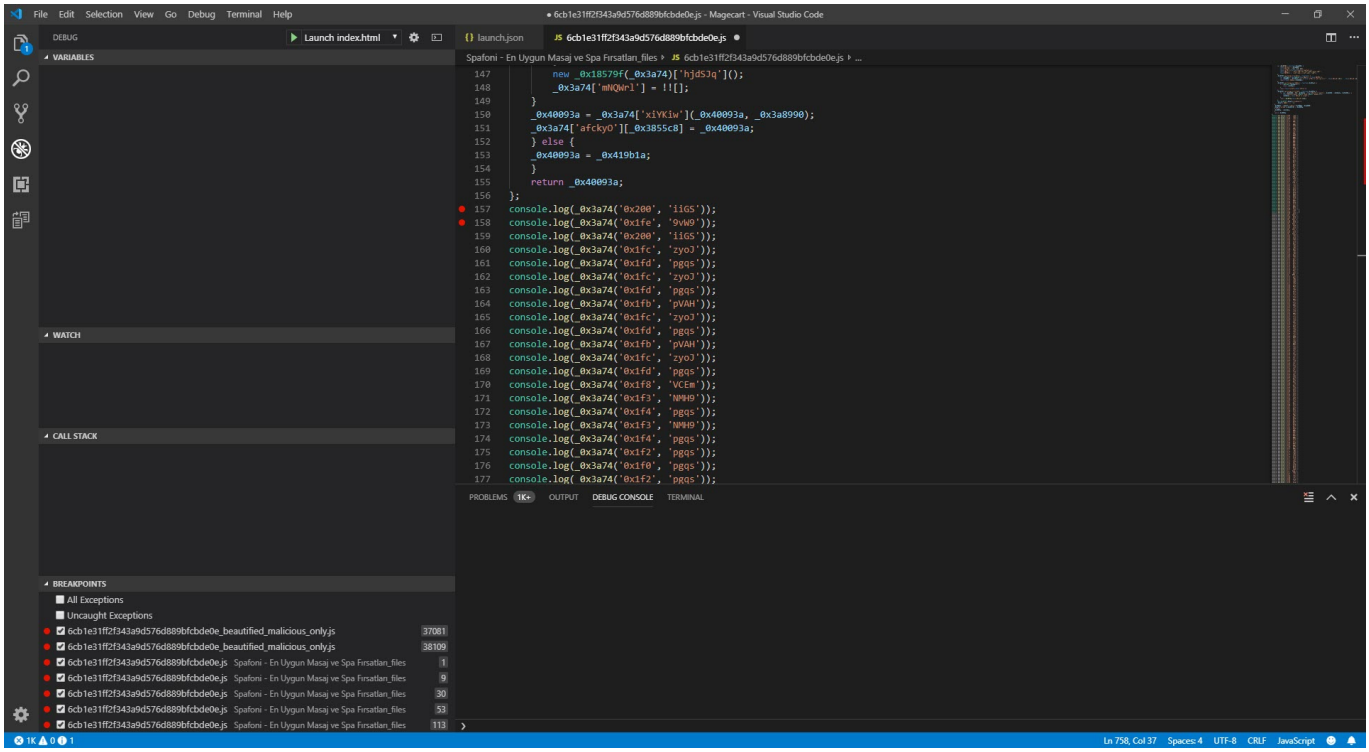
başladım ve Visual Studio Code isimli kaynak kodu editörü ile karşılaştım. Chrome hata ayıklama eklentisi sayesinde arka planda HTML, JavaScript kodu analiz etmeye imkan tanıyan ve çok sayıda eklentiye sahip olan bu editör ile hata ayıklaması yapmaya başladığımda, SetCookie ile ilişkili fonksiyonun çok sayıda dizi (Array) oluşturarak bellekteki kullanılabilecek alanları tüketerek hata ayıklamasını etkisiz hale getirdiğini (self defending) gördüm.



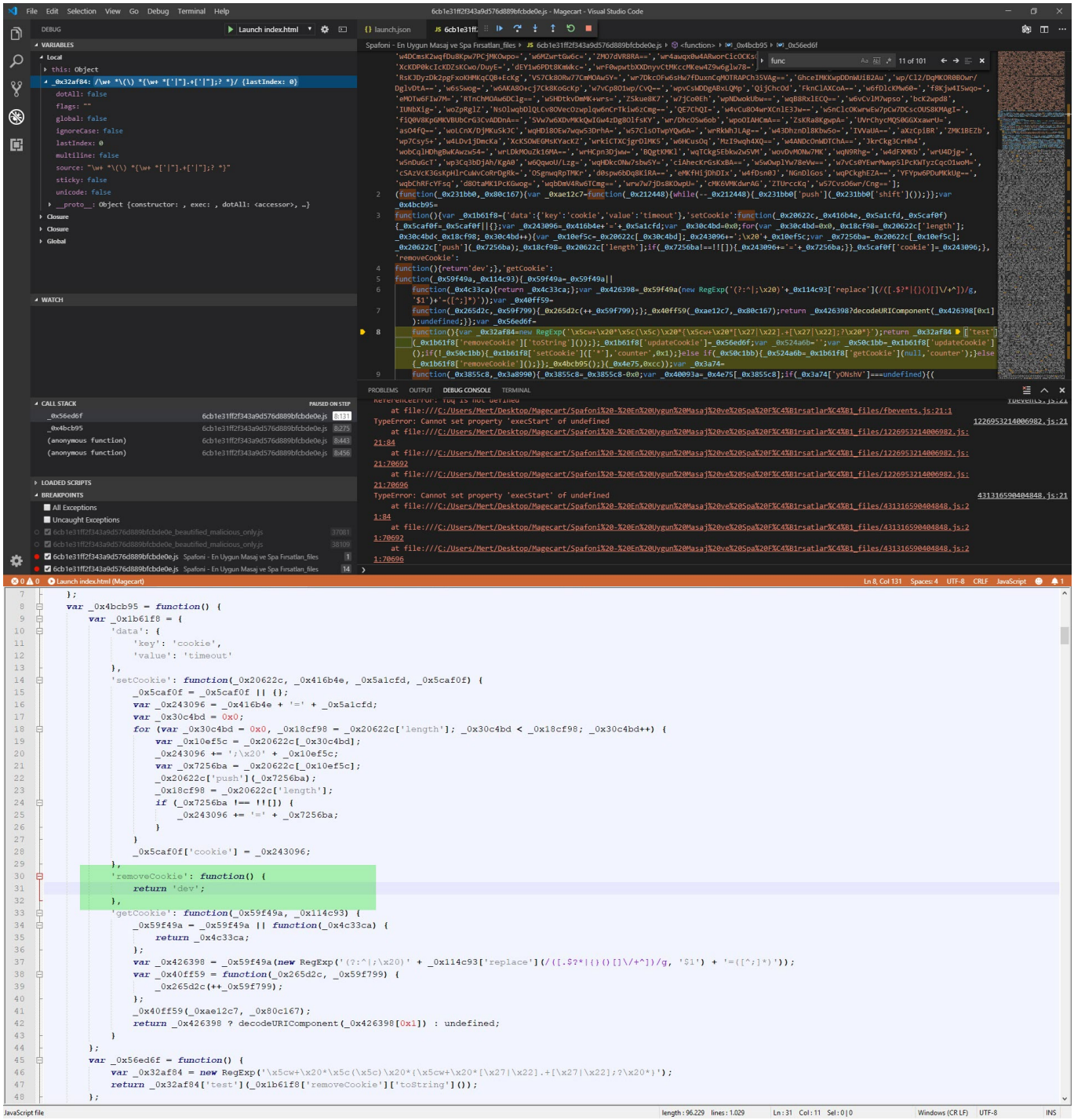


Art niyetli kişilerin amaçları doğrultusunda bu kodun internet tarayıcısında sorunsuz bir şekilde çalışması gerektiği için kodda hata ayıklamasına yönelik kontroller olduğunu düşünerek her bir fonksiyonu adım adım analiz etmeye başladım. Nihai hedefim kodu baştan sona dinamik olarak analiz etmek olmayıp çalınan bilgilerin hangi web sitesine iletildiğini ve gizlenmiş karakter dizilerini (strings) çözmek olduğu için gizlenmiş dizilerin çözülmesi amacıyla kullanılan _0x3a74 fonksiyonundan yola çıkarak ilerledim.

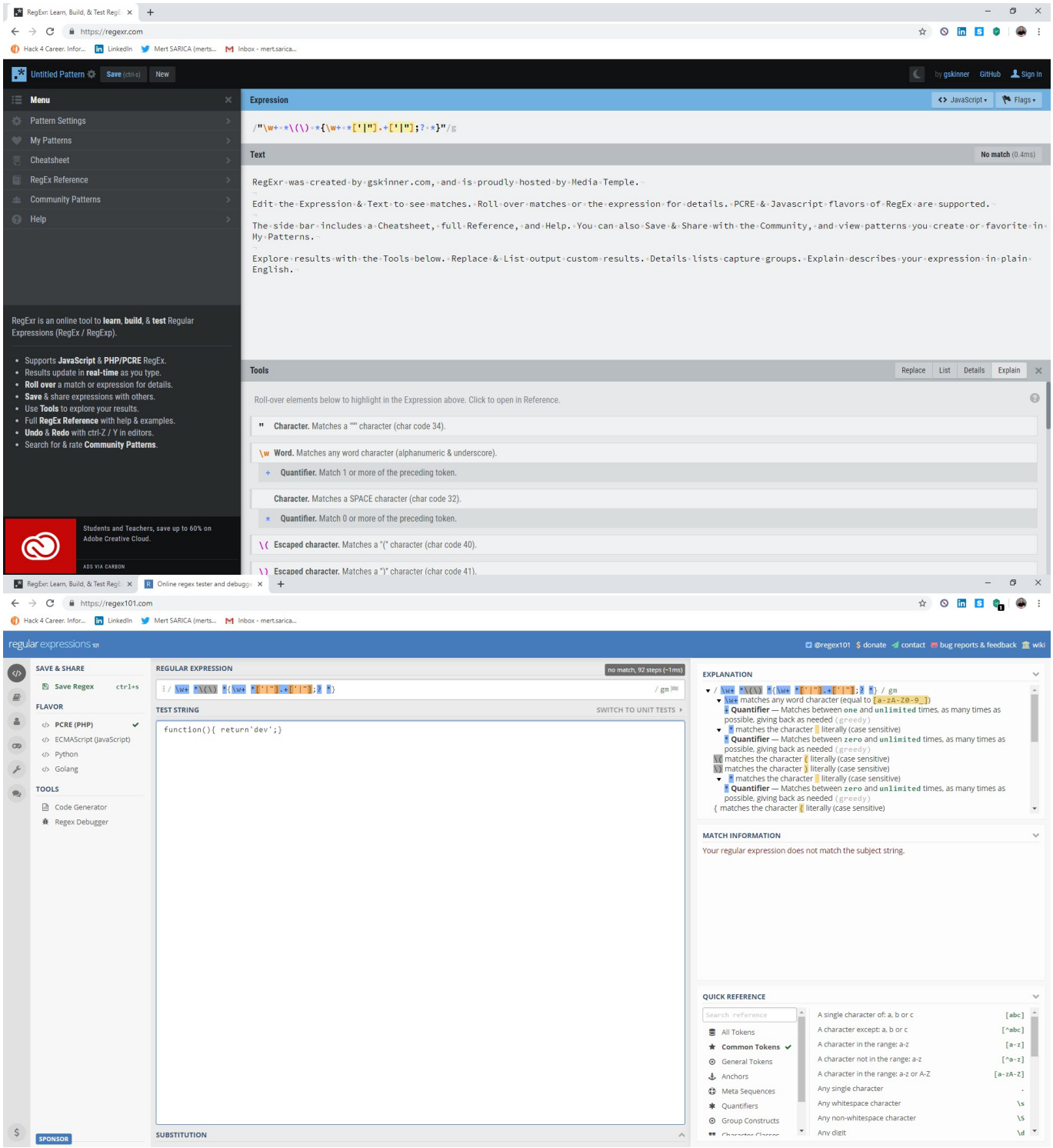




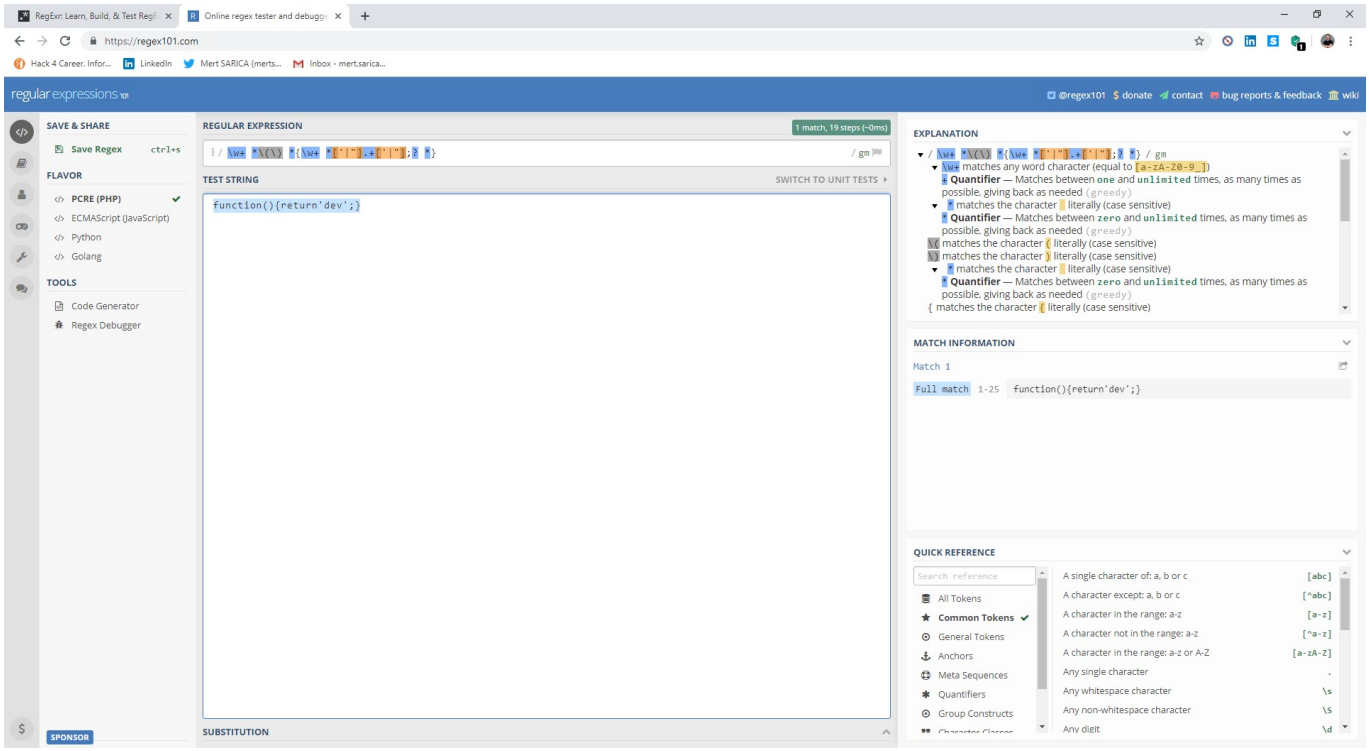
Analiz ederken bir yerde Regex ile removeCookie değerinde { işareti ile return kelimesi arasında boşluk kontrolü yapıldığını farkettim. Boşluk karakterinin tespit edilmesi durumunda kodun akışı yukarıda bahsettiğim çok sayıda dizi oluşturup probleme yol açan fonksiyona ilerliyordu. Peki art niyetli geliştirici neden böyle bir kontrol koymuştu ? Bu gibi karmaşıklaştırılmış, okunaklı olmayan kodlarla karşılaşan analistlerin ilk yaptığı iş, kodu araçlar (JavaScript Beautifier gibi) yardımıyla okunaklı, formata uygun bir hale getirmek olduğu için bu araçlar otomatik olarak araya boşluk koyuyorladı ve bu da kodun analiz edildiğine dair art niyetli kişilere güzel bir tespit mekanizması oluşturmaya imkan tanıyordu.



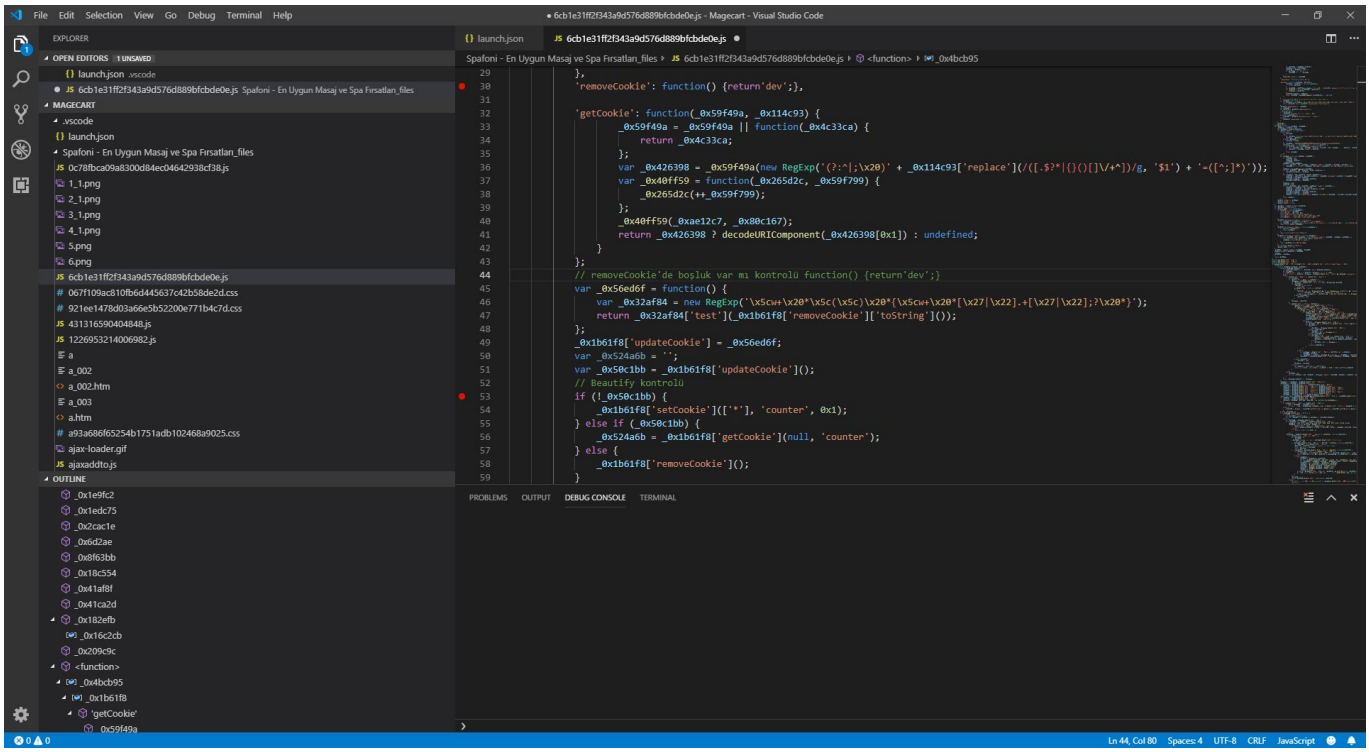
```
};
var _0x4bcb95 = function() {
    var _0x1b61f8 = {
        'data': {
            'key': 'cookie',
            'value': 'timeout'
        },
        'setCookie': function(_0x20622c, _0x416b4e, _0x5a1cfd, _0x5caf0f) {
            _0x5caf0f = _0x5caf0f || {};
            var _0x243096 = _0x416b4e + '=' + _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']());
    };
```

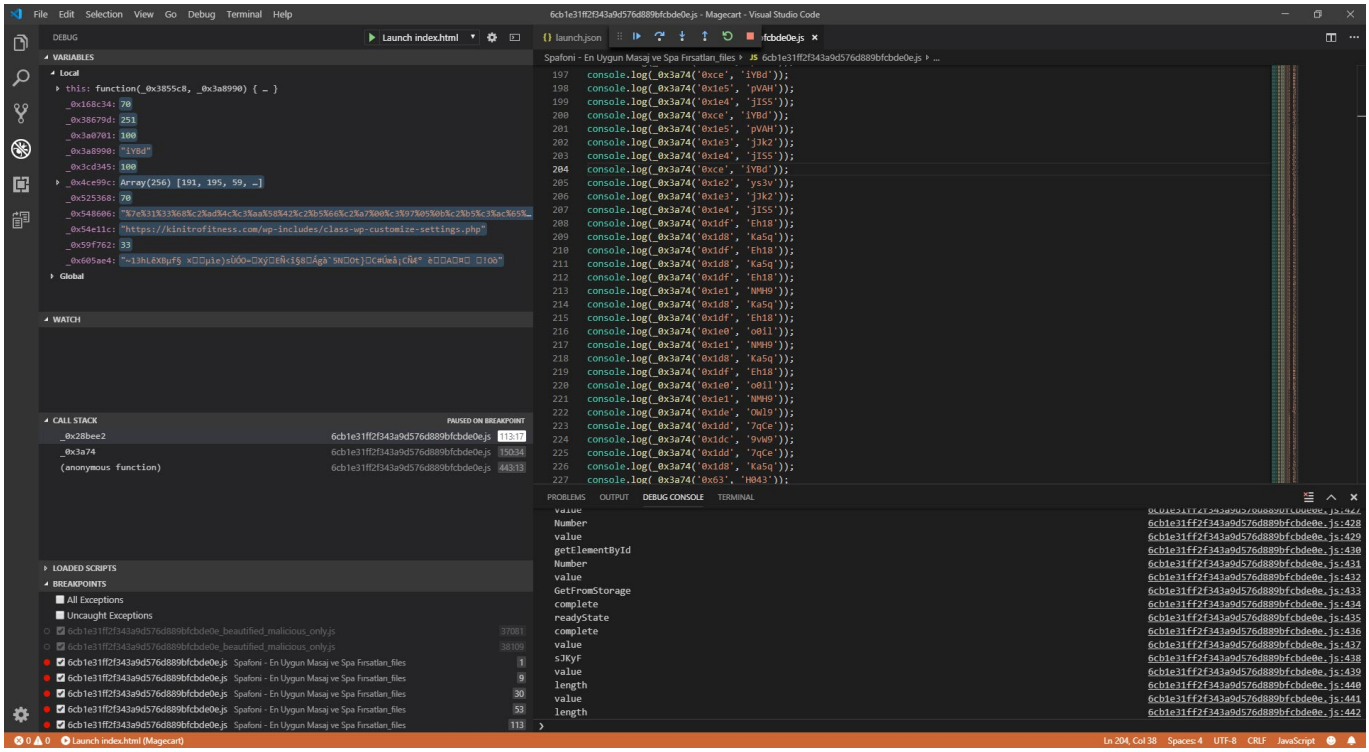

RegExr: Learn, Build, & Test Regex
https://regexr.com
Untitled Pattern
Save (ctrl-s)
New
by gskinner
GitHub
Sign In
Menu
Pattern Settings
My Patterns
Cheatsheet
RegEx Reference
Community Patterns
Help
Expression
JavaScript
Flags
/"\w+=\(\)*{\w+*['|"].+['|"];?*}"/g
Text
No match (0.4ms)
RegExr was created 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



Kodu Regex kontrolünden başarıyla geçecek şekilde boşluksuz olarak düzenleyip gizlenmiş karakter dizilerini çözdükten sonra statik ve dinamik kod analizi sayesinde kredi kartı bilgilerinin (CVV, Holder, ccexpiry, ccnumber, cvc, fullname) çalınarak https://kinitrofitness[.]com/wp-includes/class-wp-customize-settings.php adresine iletildiğini tespit etmiş oldum.





Ülke olarak COVID-19 salgınını geride bırakacağımız sağlıklı günlerde yeni bir yazı ile tekrar görüşmek dileğiyle herkese sağlıklı ve güvenli günler dilerim.

Not:

1. Bu yazı ayrıca Pi Hediyem Var #17 oyununun çözüm yolunu da içermektedir.