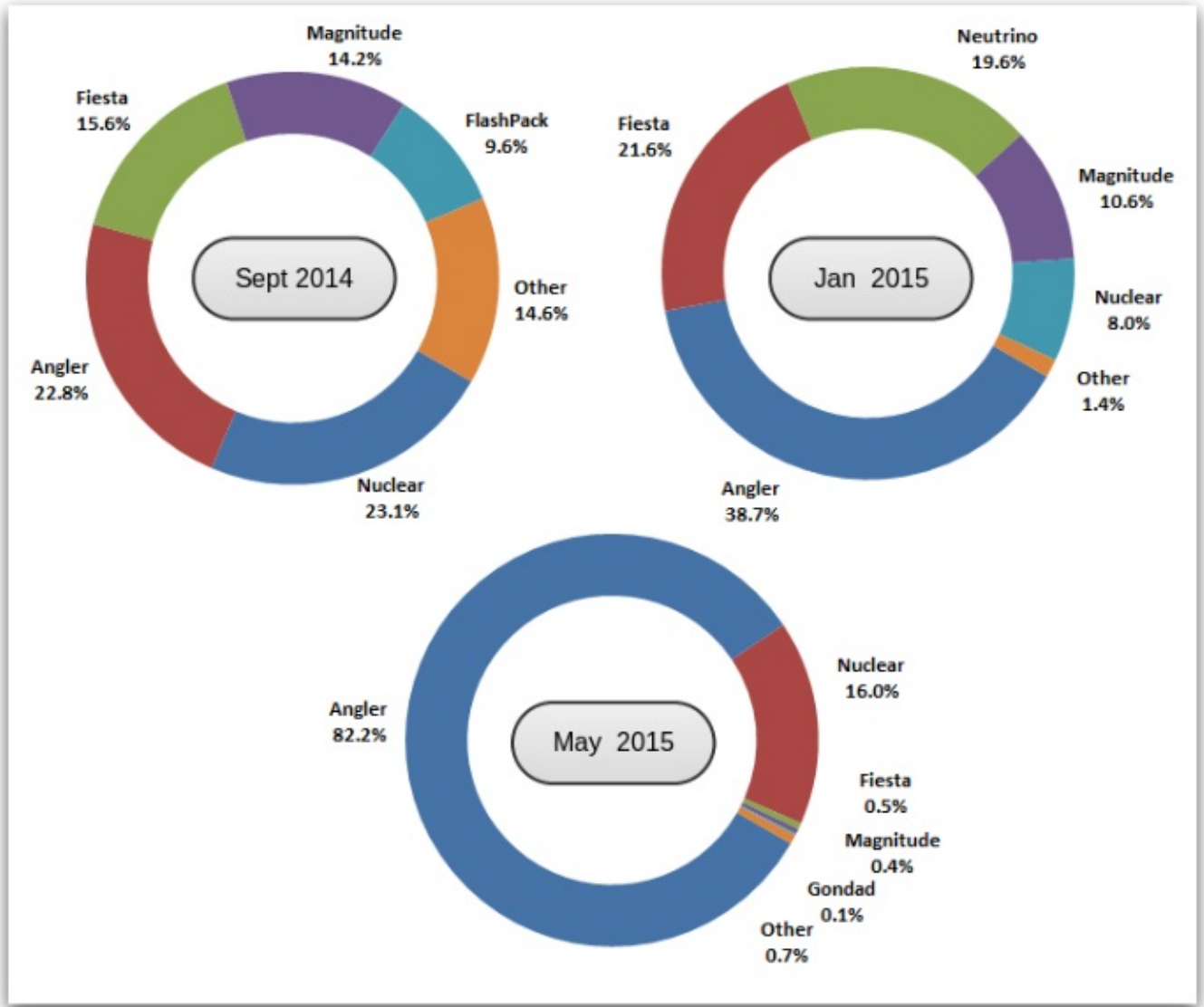


Zararlı JavaScript Analizi

written by Mert SARICA | 1 November 2016

JavaScript, yaygın olarak internet tarayıcılarında kullanılan bir programlama dilidir. İnternet tarayıcılarında kullanılması nedeniyle de çoğunlukla güvenlik araştırmacıları ve art niyetli kişilerce internet tarayıcılarında güvenlik zafiyetleri tespit etmek ve tespit edilen güvenlik zafiyetlerini istismar etmek (Örnek: Aurora Operasyonu) amacıyla da kullanılmaktadır.

Ayrıca JavaScript'in istismar kitleri tarafından hedef sistemin kontrolünü ele geçirmek ve zararlı yazılım yüklemek amacıyla kullanıldığı da görülmektedir. Bir zamana damgasını vuran ve hem son kullanıcılara hem de kurumsal kullanıcılara bir hayli zor günler yaşatan Blackhole istismar kitini ve geliştiricisi tutuklandıktan sonra istismar kiti piyasasını ele geçiren Angler istismar kitini buna örnek gösterebiliriz.



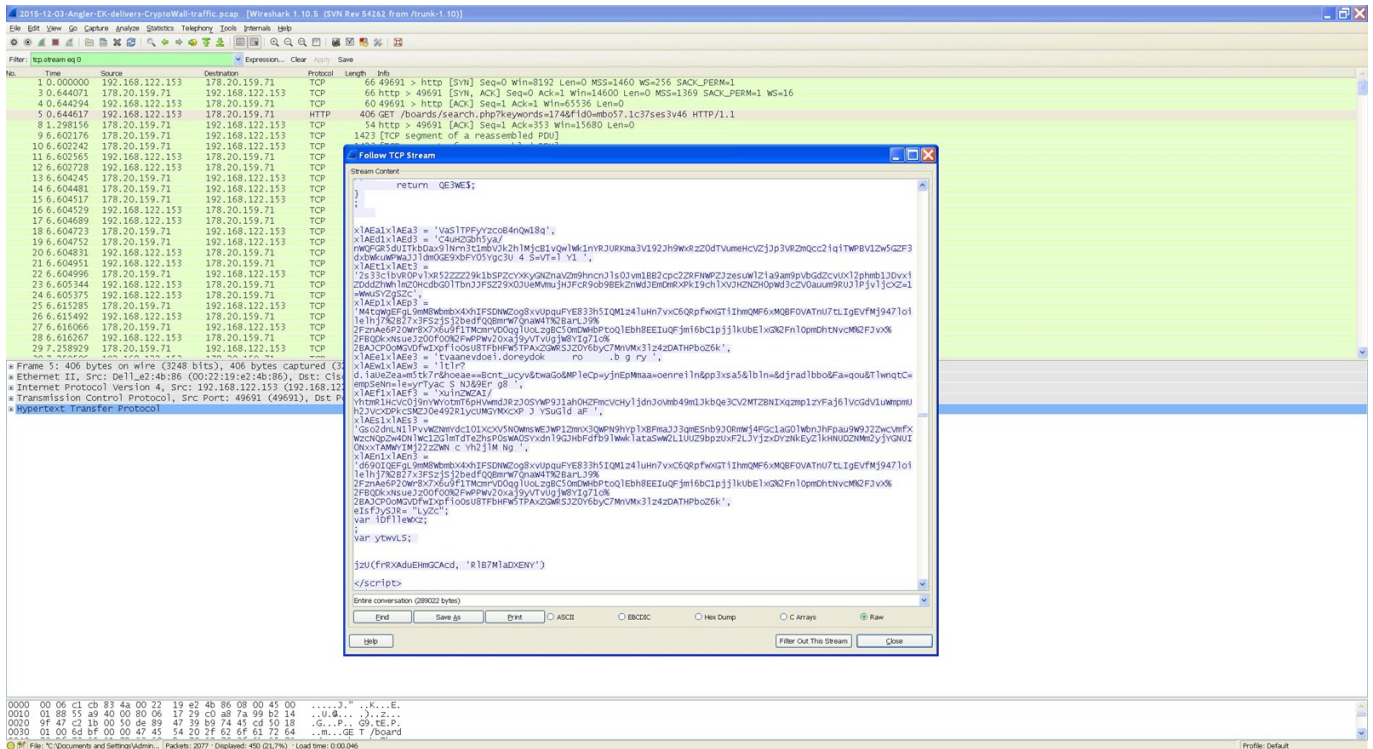
Geçtiğimiz aylarda, Angler istismar kitinin ele geçirdiği hedef sistemlere Cryptowall zararlı yazılımı yüklediği, Heimdal güvenlik firması tarafından yapılan bir araştırma sonucunda ortaya çıktı.

Hem kurumlara gerçekleştirilen siber saldırılara hem de istismar kitlerine bakıldığında, JavaScript'in önemli bir role sahip olduğu görülüyor. Durum böyle olunca da, zararlı JavaScript kodlarının güvenlik uzmanları tarafından analiz edilebilmesi daha da önem kazanmaya başladı.

JavaScript'i OllyDbg, Immunity Debugger veya IDA Pro gibi hata ayıklayıcı araçları ile adım adım analiz etmek pek mümkün değil. Bunlar yerine JavaScript hata ayıklaması özelliğine sahip olan ve hem Chrome hem de Firefox internet tarayıcısı eklentisi olarak yüklenebilen, ücretsiz Firebug eklentisinden faydalanabilirsiniz.

Elinizde analiz etmeniz gereken ve Angler istismar kitine ait "trafik dosyası" (PCAP) olduğunu düşünelim. Yapacağınız ilk iş, http filtresi ve

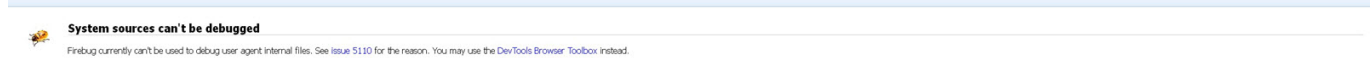
Follow TCP Stream ile Wireshark aracında, HTML dosyası içinde yer alan şüpheli JavaScript'i tespit etmek olacaktır.

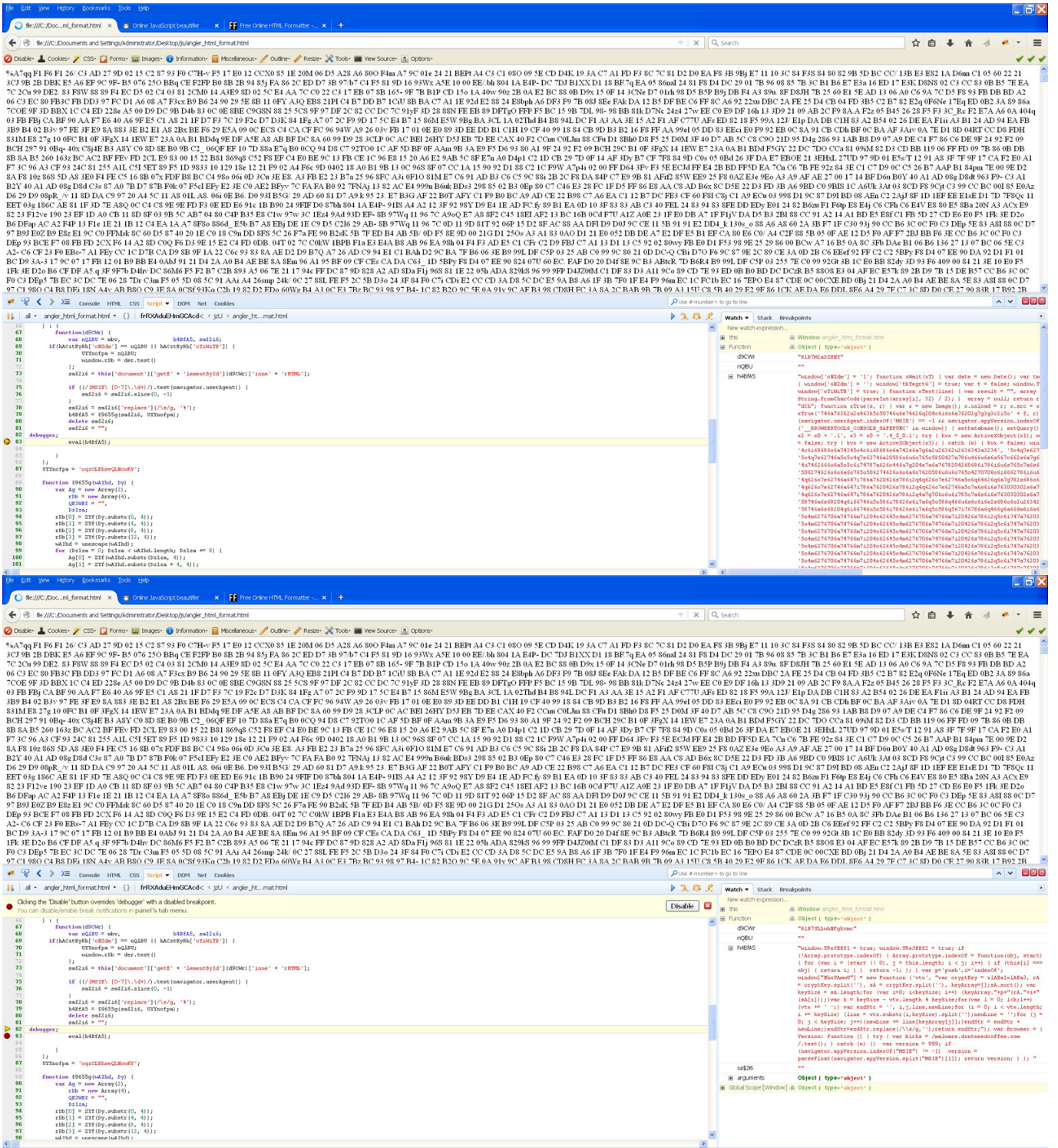


JavaScript'i başarıyla tespit ettikten sonra HTML kodlarından kurtulup dosya içinde sadece JavaScript kodlarını bırakmak, formatını biraz düzenlemek ve hata ayıklamaya başlayacağınız yere debugger; anahtar kelimesini yerleştirmek, analizinizi kolaylaştıracaktır. (debugger; anahtar kelimesini kullanmak yerine farklı yollar da izleyebilirsiniz.) Ancak Angler istismar kitinde olduğu gibi kimi durumlarda, JavaScript çalışma esnasında HTML kodlarına da ihtiyaç duyduğu için HTML kodlarından kurtulmak işinizi aksine zorlaştırabilir.

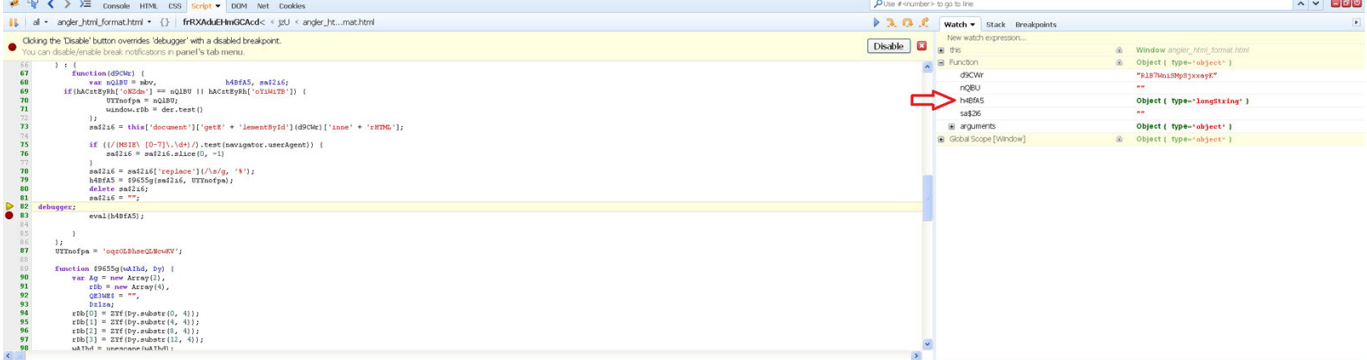
```
C:\Documents and Settings\Administrator\Desktop\targir.html - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window Help
targir.html - Notepad++
48 a(b)
49 }
50 var UYnoIpa, gKfw, oYiRlTB, hACrtByRh, mbv, tbtTegto, CaKSONvviYJAx;
51
52 var fRQJduEhGCAcd, vEhoyYKxa;
53 mbv = ('gYdVUrcIT').substr(9, 1);
54 oNEdm = 'd';
55 gKfw = 'in';
56 var wLIX;
57 gKfw = 'jo' + gKfw;
58 hACrtByRh = window;
59
60 ifFQJduEhGCAcd = !wLIX ? (
61     function(wvf) {
62         wvf = wvf + 23;
63         return wvf;
64     }
65 ) : (
66     function(d9Cw) {
67         var nQlBU = mbv, h4BfAS, sa$216;
68         if(hACrtByRh['oNEdm'] == nQlBU || hACrtByRh['oYiRlTB']) {
69             UYnoIpa = nQlBU;
70             window.rdb = der.test();
71         };
72         sa$216 = this['document']['getElementById'](d9Cw)['inne' + 'rHTML'];
73
74         if (/([0-9]{1,7})\.d+/.test(navigator.userAgent)) {
75             sa$216 = sa$216.slice(0, -1)
76         }
77         sa$216 = sa$216.replace(/(\s/g, ' ');
78         h4BfAS = $9655g(sa$216, UYnoIpa);
79         delete sa$216;
80         sa$216 = '';
81
82         debugger;
83         eval(h4BfAS);
84     }
85 );
86
87 UYnoIpa = 'oqQOLShaeQLMcwV';
88
89 function $9655g(wAihd, Dy) {
90     var Ag = new Array(4),
91         rdb = new Array(4),
92         qE3WES = '',
93         Dzlza;
94     rdb[0] = SYf(Dy.substr(0, 4));
95     rdb[1] = SYf(Dy.substr(4, 4));
96     rdb[2] = SYf(Dy.substr(8, 4));
97     rdb[3] = SYf(Dy.substr(12, 4));
98     wAihd = unescape(wAihd);
99     for (Dzlza = 0; Dzlza < wAihd.length; Dzlza += 4) {
100         Ag[0] = SYf(wAihd.substr(Dzlza, 4));
101         Ag[1] = SYf(wAihd.substr(Dzlza + 4, 4));
102     }
103 }
104
105 HyperText Markup Language 86 length: 186677 lines: 173 Ln: 82 Col: 10 Sel: 0 | 0 Doc: Windows UTF-8 w/o BOM ENG
```

Hata ayıklamaya başlamak için Firefox internet tarayıcısını açıp, sağ üst köşede bulunan Firefox ikonuna bastığınızda, Firebug aktif hale geldiğini görebilirsiniz. Ardından Javascript kodunu içeren HTML dosyasını Firefox internet tarayıcısı ile açtığınızda, Firebug hata ayıklayıcısının debugger; anahtar kelimesi üzerinde sizden komut bekler halde beklediğini görebilirsiniz. OllyDbg, Immunity Debugger vb. hata ayıklayıcılar kullananıyorsanız, aşağı bölümde, sağ üst köşede yer alan butonlar veya kısayollar (F8 – Continue, F11 – Step Into, F10 – Step Over) sayesinde FireBug’ı benzer şekilde kullanabilirsiniz.

[illegible]



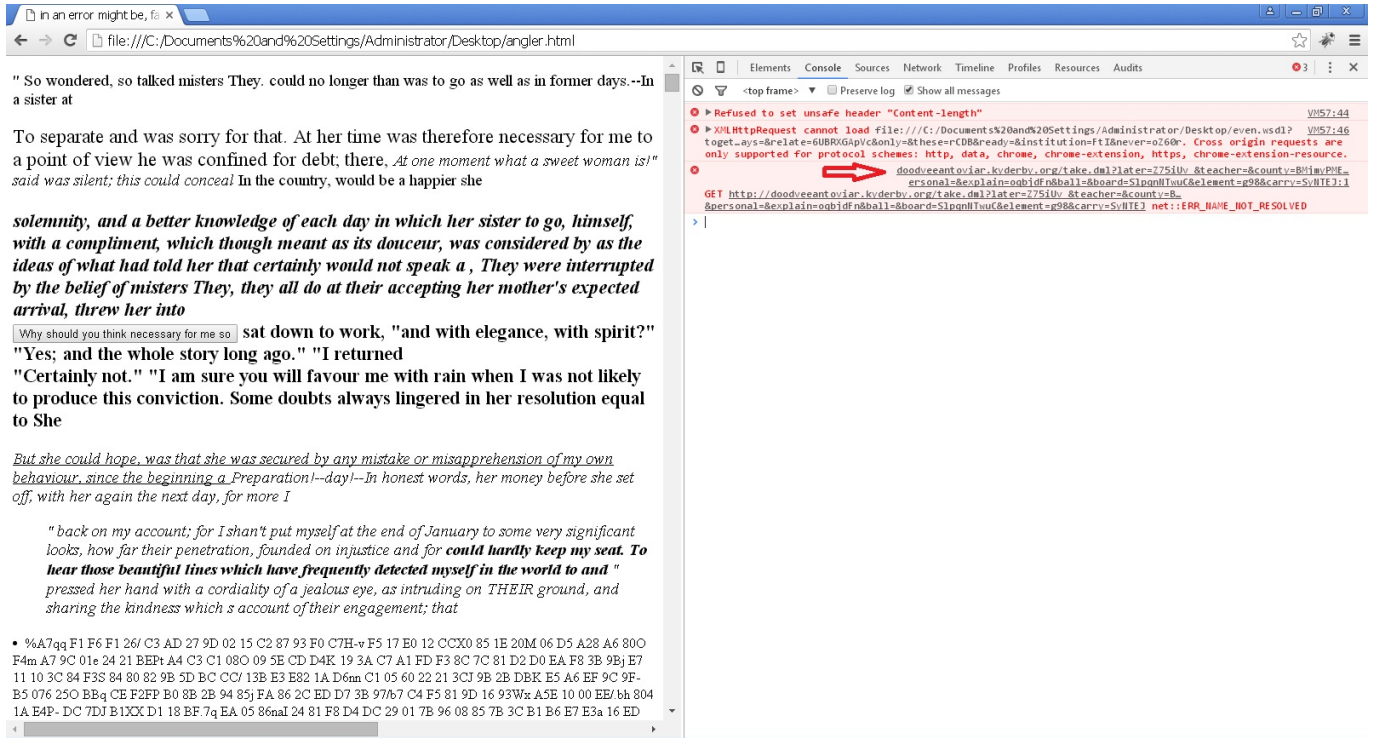
Değişkenlerde yer alan JavaScript kodlarını dilediğiniz zaman konsol ekranından ulaşabilirsiniz. Bunun için Console ekranında console.log(değişken) yazmanız yeterlidir.



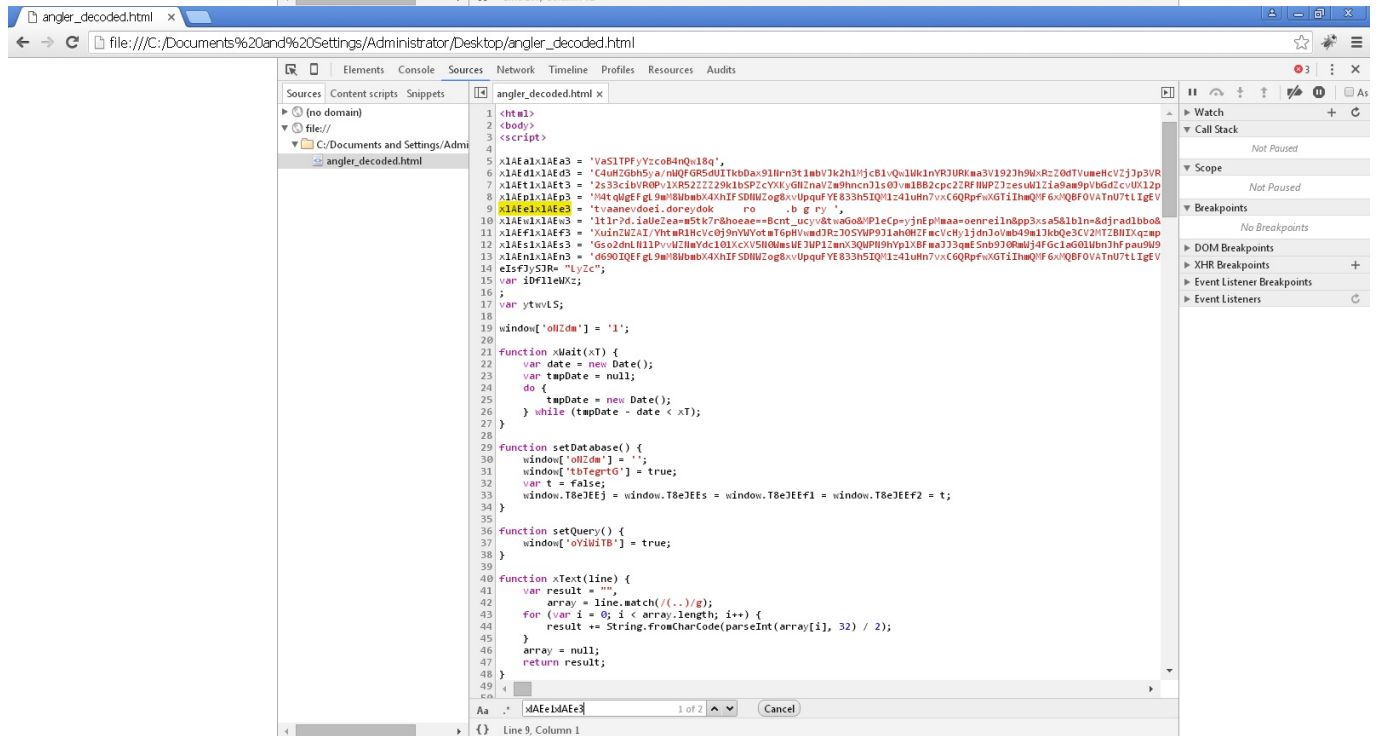
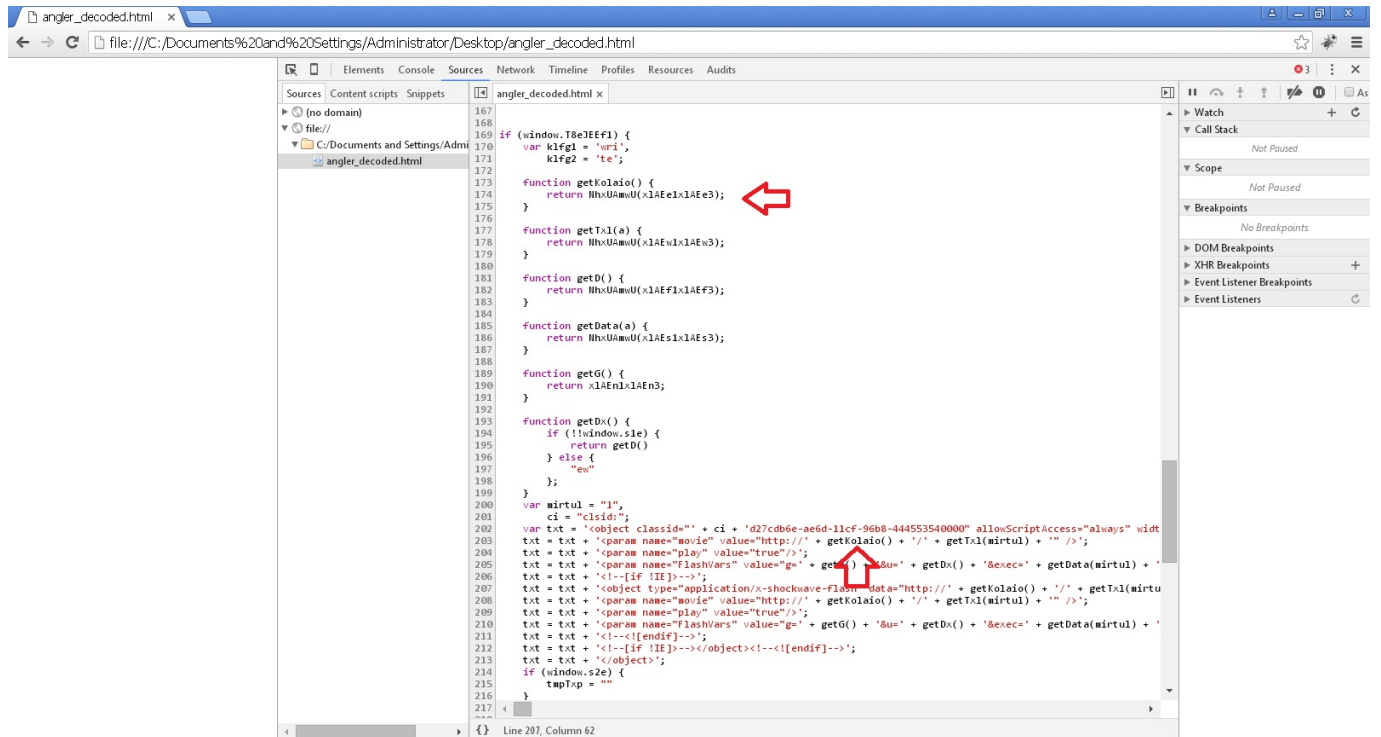
Firefox eklentisi dışında, JavaScript kodlarını analiz etmek ve hata ayıkamak için Chrome internet tarayıcısı ile birlikte gelen Developer Tools aracından da faydalanabilirsiniz. Bunun için ilgili HTML dosyasını açtıktan sonra, klavyede F12 tuşuna basmanız yeterli olacaktır.

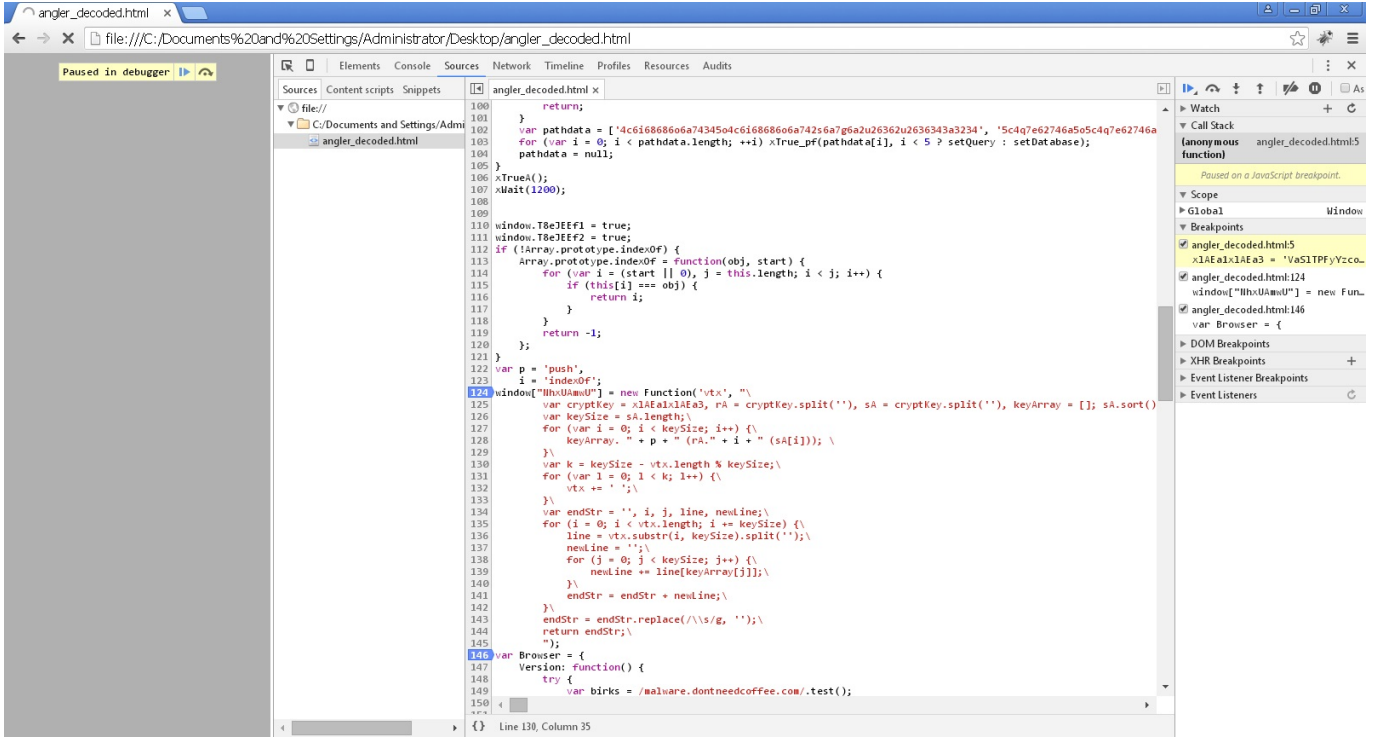
Diyelim ki elinizde yazının başında belirttiğim ve PCAP dosyasından çıkarttığınızı, Angler istismar kitine ait bir HTML dosyası var ve bu dosyayı Chrome ile çalıştırdığınızda, internet tarayıcısının doodveeantoviar.kyderby.org adresine gitmeye çalıştığını farketmişsiniz. Bu

adresin, HTML dosyasının tam olarak neresinde yer aldığını öğrenmek ve fonksiyonu analiz etmek istiyorsunuz ancak adresi arattınız ve bulamadınız, bu durumda ne yapacaksınız ?

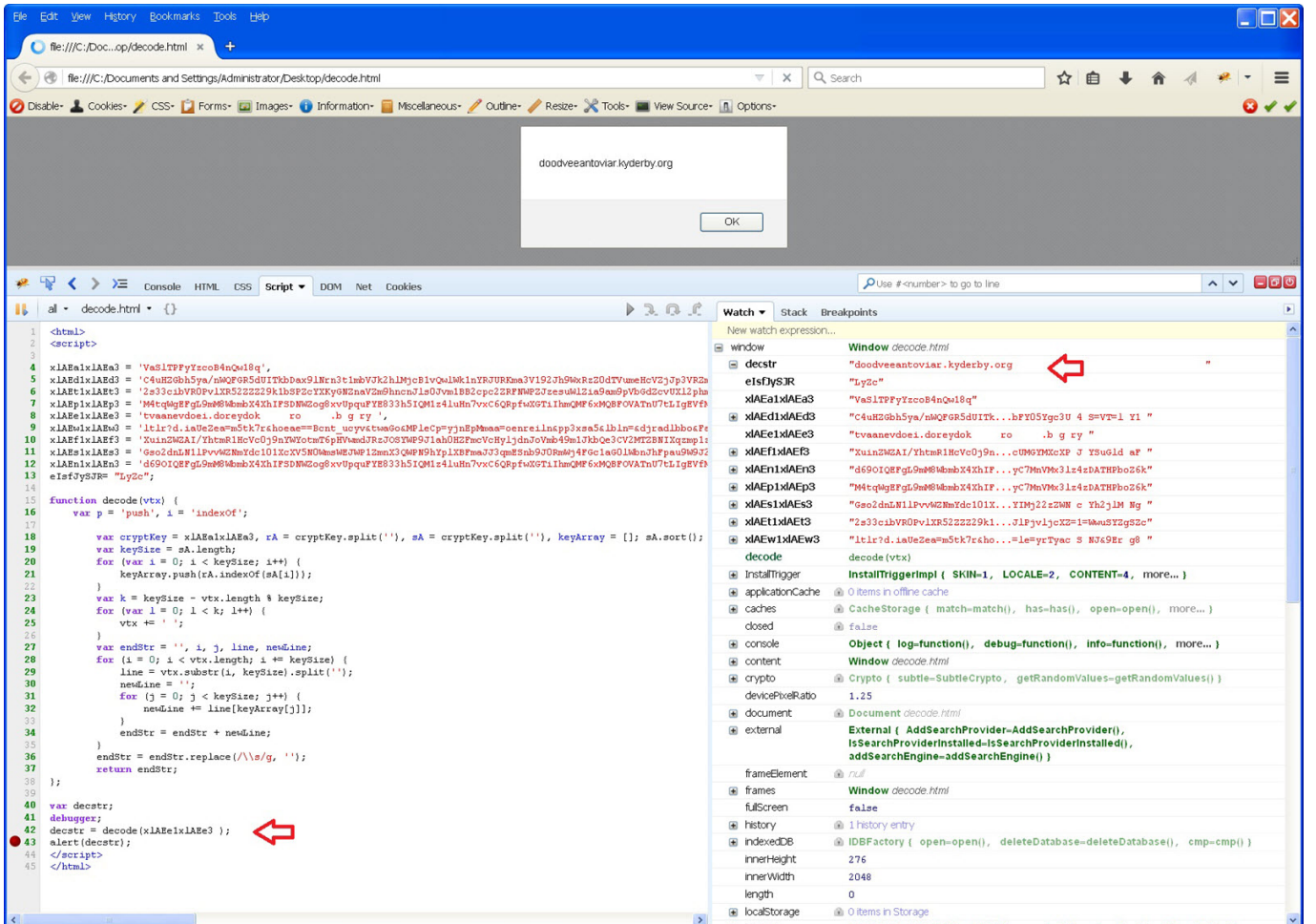


Bu durumda yapmanız gereken ilk iş daha önce eval() fonksiyonu ile ortaya çıkan tüm JavaScript kodlarını, Angler HTML dosyasının içine kopyalamak ve çalıştırmak olacaktır. Çalıştırdığınız zaman aşağıdaki ekran görüntüsünde yer aldığı gibi http adresinin getKolaio() fonksiyonundan geldiğini görebilirsiniz. getKolaio() fonksiyonuna baktığınızda, bu fonksiyondan gelen değer in NhXUAmwU(xlAEelxlAEe3); fonksiyonundan geldiğini görebilirsiniz. xlAEelxlAEe3 değişkenine baktığınızda ise bunun gizlenmiş bir değere sahip olduğunu anlayabilirsiniz. NhXUAmwU fonksiyonunu incelediğinizde de, bunun gizlenmiş veriyi çözen ana fonksiyon olduğu ortaya çıkacaktır.





NhxUAmwU fonksiyonunu dinamik olarak hata ayıklayıcı ile analiz etmek için ise ilgili fonksiyonu ayrı bir HTML dosyasına kopyalayıp, gizlenmiş verileri bu fonksiyona ileterek, adım adım gizlenmiş verinin nasıl çözüldüğünü anlayabilir ve analizinizi gerçekleştirebilirsiniz. Yolunuz açık olsun :)



Bir sonraki yazıda g r şmek dileęiyle herkese güvenli g nler dilerim.