Web Exploits Kung-Fu with Fiddler

Jerome Segura
Senior Malware Research
jsegura@malwarebytes.org

Drive-by attacks are the most common infection vector and have been so for several years. The Exploit Kit market is also thriving and the kits getting more sophisticated and pricier. Whether you suspect your own site has been infected or you are a security researcher tracking down malicious URLs, Fiddler is a very capable and useful tool to help you identify traffic patterns, malicious code and exploit URLs.

Fiddler acts as a proxy between client applications (such as a web browser) and the websites they are connecting too.

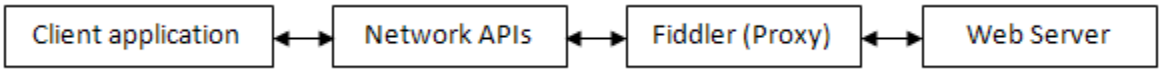


Figure 1: Fiddler's proxy between client application and web server

All HTTP(S) requests and responses transit through the Proxy, giving you the ability to see exactly what is going on between your browser and the servers it is connecting to.

Analyzing web traffic:

Every time you navigate to a website, your browser sends out a Request for a particular URL. The web server will reply with a Response containing the page you asked for (or a not found 404 error if that document did not exist). This Request-Response workflow is known as a **Web Session** in Fiddler. Each Session is represented by a row in the Web Sessions List:

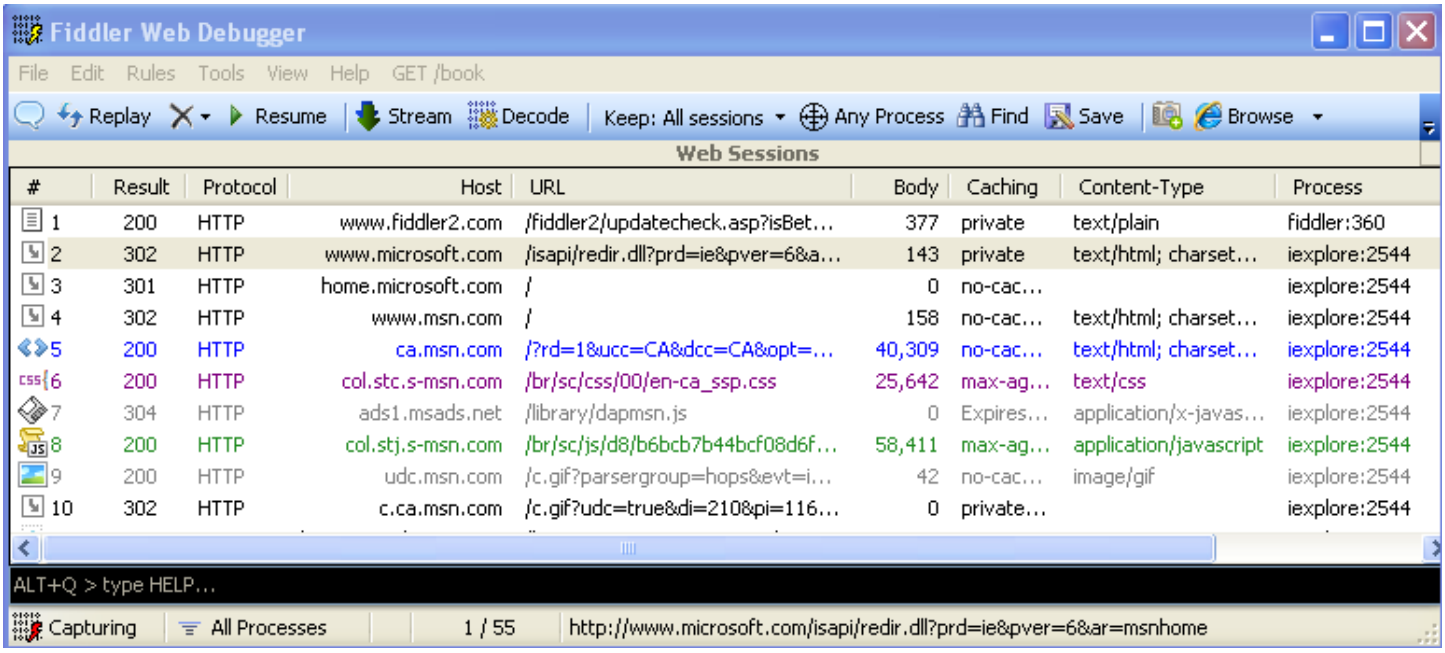


Figure 2:Fiddler's main view showing the Web Sessions list

Fiddler uses standard columns (you can add more or customize your own) that display certain properties for each Web Session:

#: A number that sorts each Session by chronological order

Result: The HTTP response code indicating whether the server was able to fulfill the request or not.

Protocol: Fiddler only works for HTTP(S) and FTP protocols.

Host: The website's domain name.

URL: The full path of the URL requested.

Body: The size of the response

Caching: Caching, as supported by client applications.

Content-Type: As described, the type of content returned (html, JavaScript, image…)

Process: The client application making the request (i.e. Internet Explorer, Firefox, Adobe Reader, etc)

Most people only use Fiddler to view web traffic or find which URLs are being requested and its simple interface does the job quite well. But there's a whole new world beyond that if you are interested in learning more about the code that goes through your browser.

By default, Fiddler's Tab section is on the right hand side and gives you more information on each Web Session. We will focus on the Inspectors tab as it is the most relevant to our needs. When you highlight a particular Web Session, the Inspectors tab is divided into the Request at the top and the Response at the bottom.

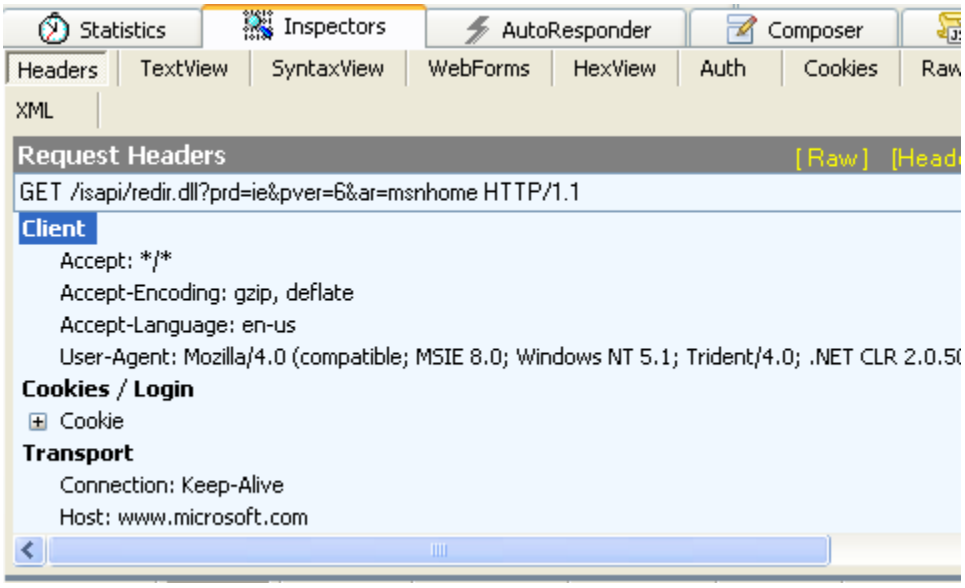


Figure 3: The Request headers window

The Request view (Figure 3) gives you information about the client (through its User-Agent), its Request type (GET, POST, etc…) as well as other parameters such as compression (Encoding), cookies, etc.

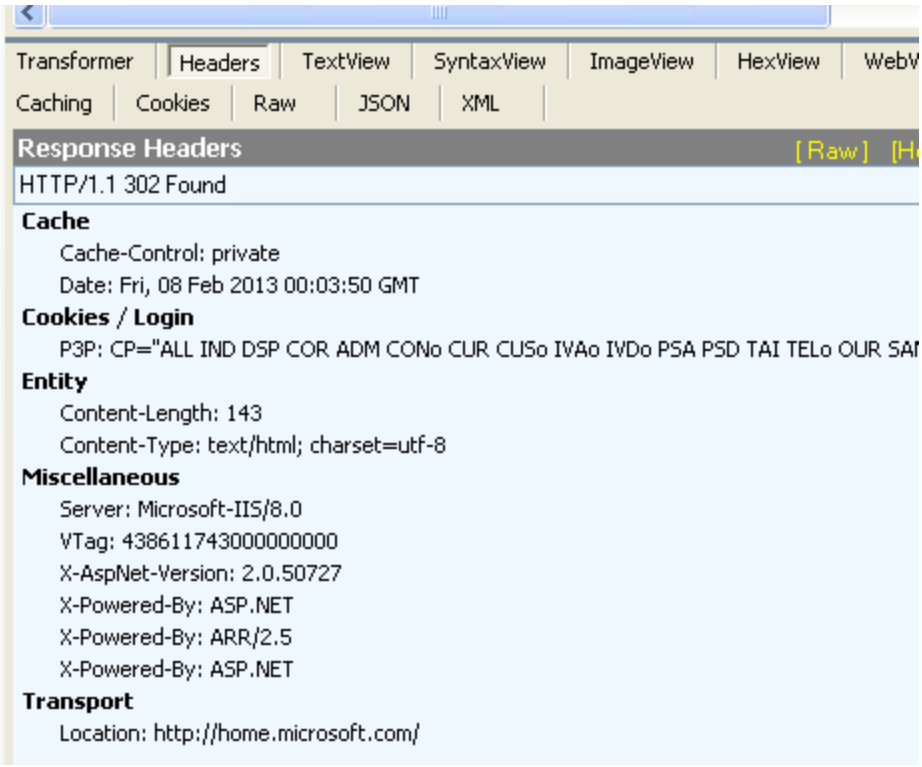


Figure 4: The Response Headers window

The web server's Response (Figure 5) which includes the type, length of the response's body as well as some information revealed by the server's configuration files (Operating System, server software, etc.)

You will note there are other "sub-tabs" for both the Request and Response. The Raw tab is your go-to shortcut to rapidly view the content's body. The information is rendered in plain text format:

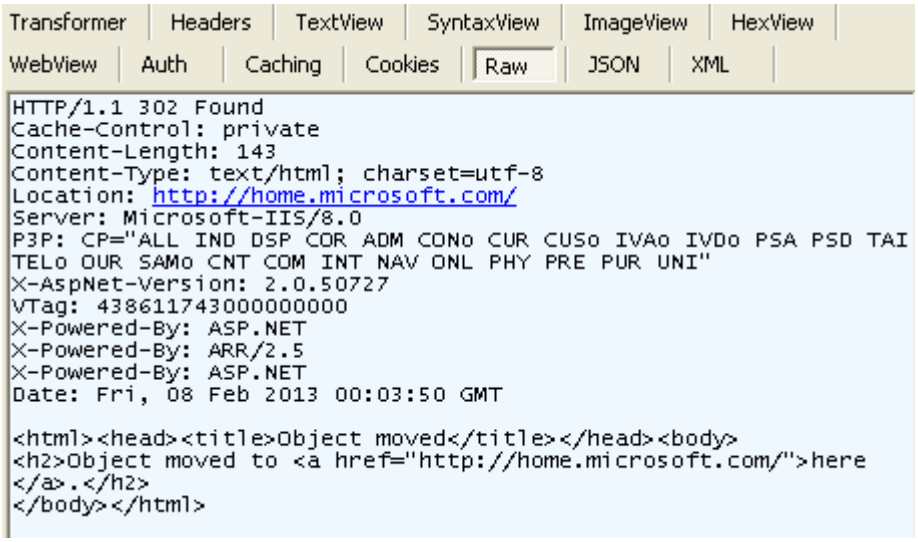


Figure 5: The Response's Raw window

Based on the content type, the raw view may not be best suited. If you are dealing with an image you may want to check the ImageView tab, if you have an executable you may want to open the HexView tab. Since often times we want to review the source code of a webpage, I recommend the SyntaxView which nicely formats html, JavaScript or CSS code.

Figure 6: SyntaxView and it's friendly formatting.

To enable some of these views and more, you can check out Fiddler's extensions page.

Spotting malicious code:

Now that you know the basic principles, let's dig in deeper and look at how Fiddler can help you to identify malicious code. But first let's find out what you should expect to be looking for. A typical drive-by download attack has some aspects that may vary slightly from one attack to another but generally speaking, they still use a similar pattern.

Infected legitimate website URL: In the majority of cases, it all stems from a typical website that has been infected with malicious code. The purpose of that code is to redirect the visitor to an external and malevolent website. The Response's body viewed using the SyntaxView shows such an example (I manually circled in red the malicious link):

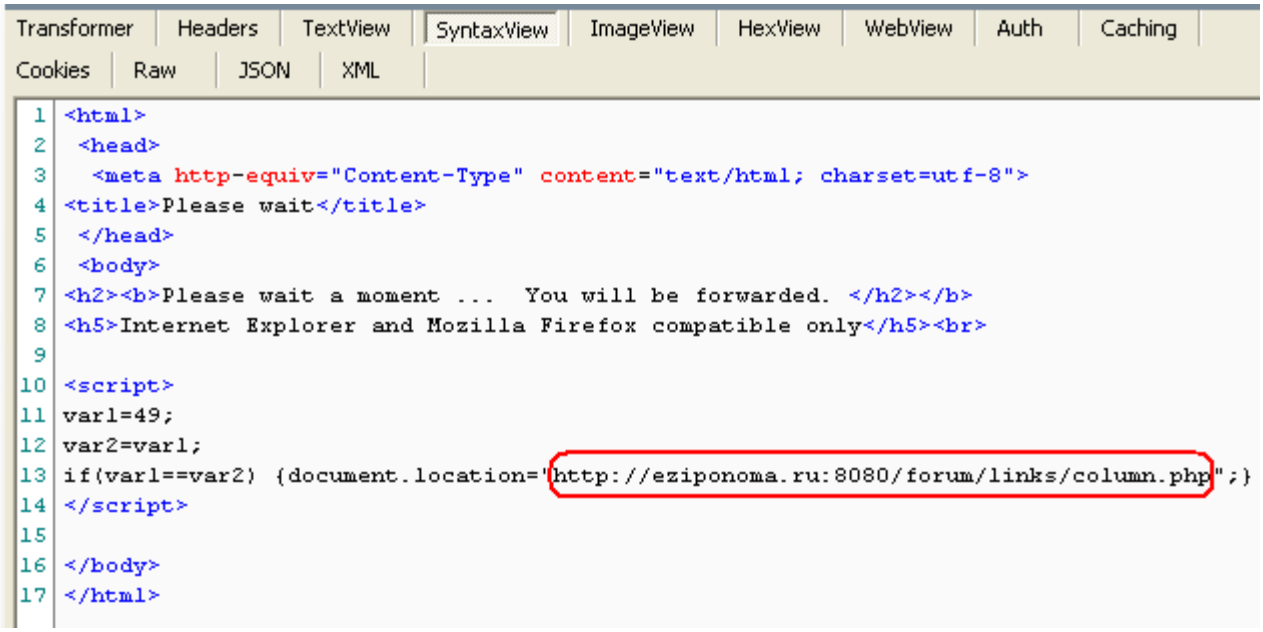


Figure 7: Source code view showing a malicious external link

Redirects: As a way to evade simple detection and be more dynamic, the bad guys will often redirect traffic several times before taking the victim to an exploit site.

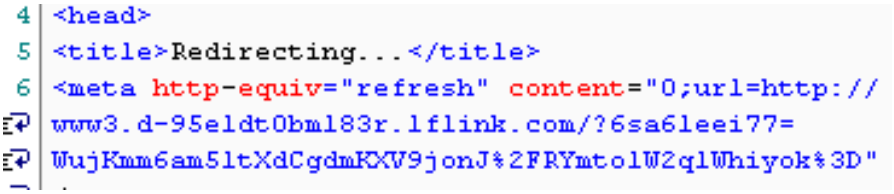


Figure 8: A typical http-equiv="refresh" that redirects the user

HTTP	.org	/main/p37	infected site
HTTP	oduc17tspr.rr.nu	/nl.php?p=d	redirect #1
HTTP	iting09arrang.rr.nu	/n.php?h=1&s=nl	redirect #2
HTTP	www3.d-95e1dt0b...	/?6sa61eei77=Wuj	redirect #3
HTTP	www1.uu8c275jtk2...	/ce0dmpdo?r71v=lql	exploit landing page

Figure 9: Multiple domains being used before reaching an exploit landing page

Exploit Kit landing page URL: This is where the exploits begin. In many cases, the victim's system (Operating System, browser type and version, plugins etc.) will be identified prior to delivering the malicious code. To make detection harder, such code is usually well obfuscated. Not all obfuscated code is malicious (some websites want to protect their Intellectual Property and actually encrypt their code from prying eyes for good reasons).

```
2 var zc9rve090RcrUL286D3Tlcll4bRdINSs="410412";
3 var v9Iiupl5holr93l3k2gQ23Lett4NQflXX26ecGL="
  ravz wbaiou{=wq``qr``hjaaf:sl,eafx````yhf:
  la``,esgnnrq``olaf:,esc``fitjdnf:``laesdd``,
  kvy:``af,esldk``envj:``kj,l-u``mh:``fmd<``vic
  ssal``~=ectnllardnani_gniam>``~t~r~t~n~id<
  vlc=ssac``~nertallaidngnl_~tfei ``=
  d``~tneclaraldn_gnifel~t>``n~r~~t~<t-!id<-c
  valssc``~=tnearllidna_gnpsca35re>``~~<d/->vi~>
  -~r~n<t~tvidc al~=ssec``tnarnallnid_gell_tf>
  ``~r~n~            id< c valssl``~=tfei_oc~l_n<
```

Figure 10: A landing page with obfuscated code

Exploit Kit malicious files (JARs, PDFs, SWF, etc…): Besides exploiting the browser itself, most exploits target third party browser plugins in software like Java or Adobe. As such, exploit kits will download infected documents which as soon as they are open will exploit one of many vulnerabilities and allow for undesired code execution.

The image below shows a Java, PDF and Flash exploit.

16.dropawayarrowrest.com	/read/at-demand_parties_contents.php?...	19,575	application/java-archive
16.dropawayarrowrest.com	/read/at-demand_parties_contents.php?...	11,543	application/pdf
16.dropawayarrowrest.com	/read/at-demand_parties_contents.php?...	2,690	text/html

Figure 11: An Exploit Kit's URLs pointing at infected documents.

The Flash exploit isn't obvious until you take a look at the Request headers which tells you what version of Flash is installed and going to be exploited.

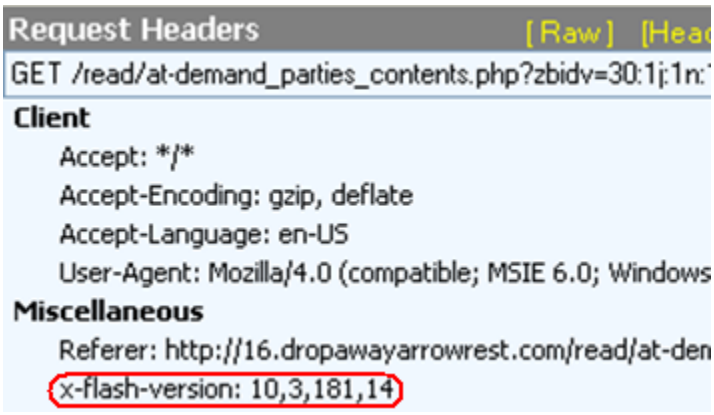


Figure 12: The browser telling the server which version of Flash is installed.

Exploit Kit malware payload: the main reason all these efforts have been put into place is for a malicious binary to be downloaded and infect the victim's PC. The HexView tab will show you the file in hexadecimal code:

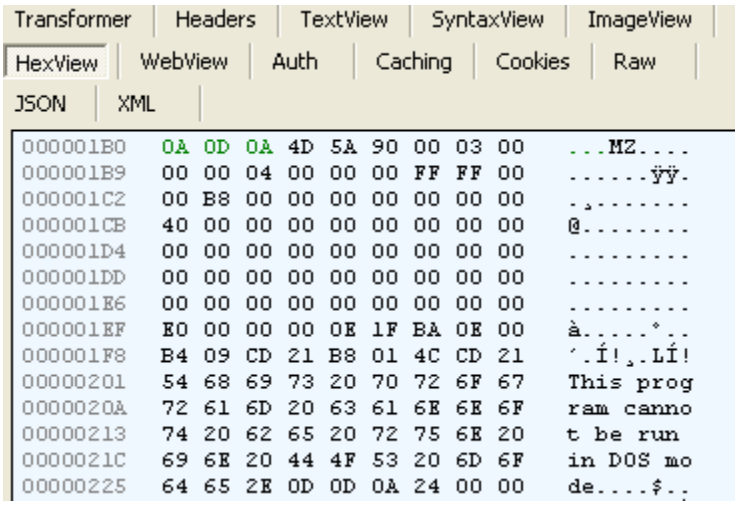


Figure 13: The "MZ" Magic Number representing an executable

Archiving files for later or to exchange with others:

Another feature that few people know is that Fiddler stores all of the files in its Web Session's list. In other words, you can save them to disk directly from Fiddler: Select a session, right-click and choose *Save>Response>Response Body*

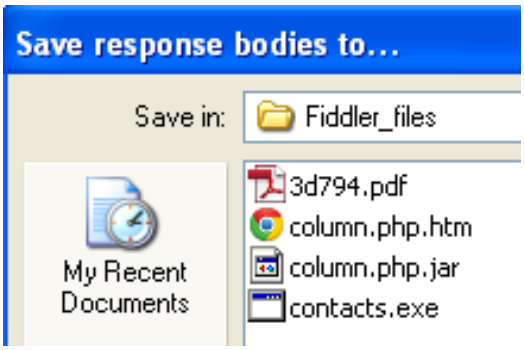


Figure 14: Saving files captured by Fiddler to the local disk

As with other tools, your traffic capture can be exported to various formats:

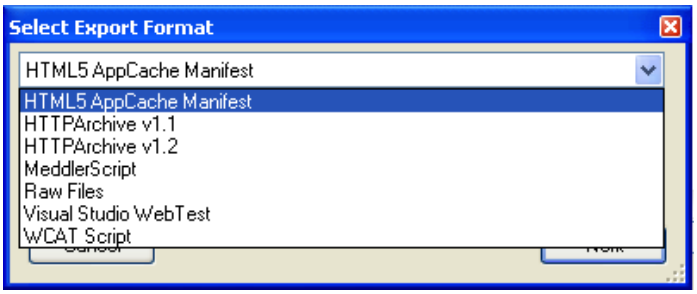


Figure 15: Export feature in Fiddler

It is also possible to save your entire capture as a SAZ (Session Archive Zip) file, Fiddlers native file type, so you can re-open it later, in Fiddler. Password-protect SAZ files with AES encryption to store sensitive traffic captures that might have embedded malicious code

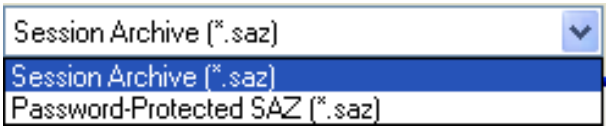


Figure 16: Saving Sessions as a compressed archive with or without password protection

Automating malicious code detection on the fly:

Fiddler comes with a scripting engine that enables you to write powerful rules coded in JScript.net. To get started, open the *CustomRules.js* file from Rules>Customize Rules… I recommend downloading the FiddlerScript Editor and checking out the cookbook page to get yourself more familiar with it.

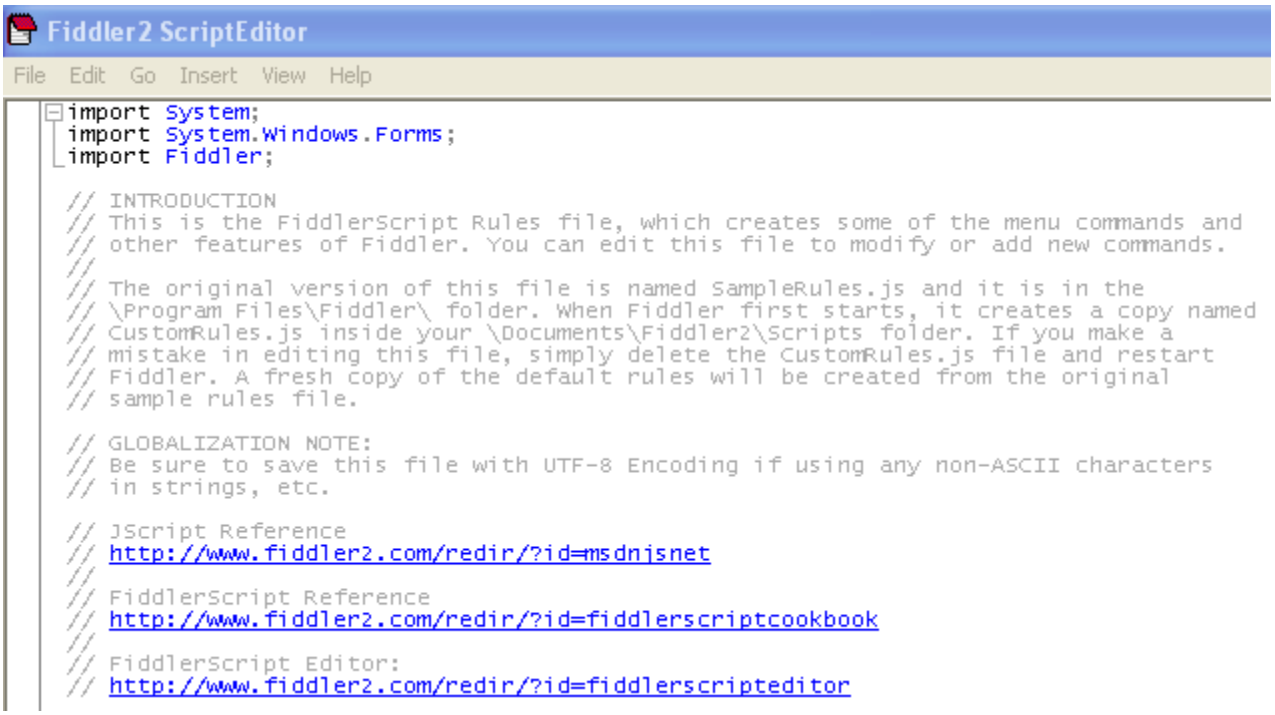


Figure 17: Fiddler2's Script Editor

The advantage of writing rules is that you can inspect both Requests and Responses and trigger a particular behavior. In essence Fiddler gives you the power to manipulate them and even fake them.

I will show you some bits of code that have helped me to identify malicious URLs. Once you are comfortable with the syntax and Fiddler's object model, as they say: "the sky's the limit".

Detect a malicious URL based on a string pattern:

```
static function OnBeforeRequest(oSession: Session) {

  var urlpattern = ":8080/forum/";

  if (oSession.fullUrl.match(urlpattern)
  {
    MessageBox.Show("Bad site identified");
  }
}
```

What does this do?

The ***OnBeforeRequest*** function is called when your browser is going to request a URL from the server. Before that happens, you can create an event. First, we define a string we want to look for in that URL. Then, as Fiddler makes each request, it will check whether the URL contains that string. *oSession* is the current session on which it performs a match on the full url.

Detect if a webpage contains malicious code:

```
static function OnBeforeResponse(oSession: Session) {

oSession.utilDecodeResponse();
```

```
 for(var i = 0; i < malware_signatures.length; i++)
 {
       if (oSession.GetResponseBodyAsString().match(malware_signatures[i]))
       {
             MessageBox.Show("Malicious code found in " + oSession.fullUrl);
             break;
       }
 }

}
```

The ***OnBeforeResponse*** function is triggered when a server sends its response back to the browser. Of course Fiddler being a proxy, it first has to go through it before it can reach our client.

As the server 's response can be encoded (compressed), we use the *UtilDecodeResponse* method to decode it so it can be read properly. For each Session we loop through our list of malware signatures (they were loaded from a file into an array) looking for a match. We get the Response's content by using the *GetResponseBodyAsString* method which essentially places the whole content into a variable. All we have to do next is find a match with one of our signatures.

Detect a particular file type:

As shown earlier, the Content-Type column in Fiddler will tell you what the file type is. But how do we do this programmatically? Here is an example for detecting a Java applet.

By looking at the Response headers:

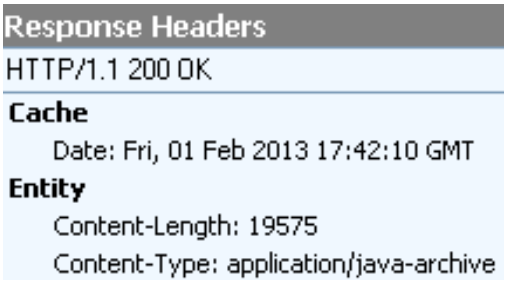


Figure 18: Response Headers showing the type of file sent by the server.

```
if (oSession.oResponse.headers.ExistsAndContains("Content-Type","application/java-archive"))
```

Or, by using MagicByte detection. (Magic numbers are the first bits of a file which uniquely identify its type.)

```
if (Utilities.HasMagicBytes(oSession.responseBodyBytes, "PK"))
{
       MessageBox.Show("Java applet identified");
}
```

Dump a response's body to disk:

Fiddler allows for dumping of any web server's Response onto your local disk. You can automate that with a bit of code:

```
oSession.SaveResponseBody("C:\\Fiddler\\payload\\" + fileName);
```

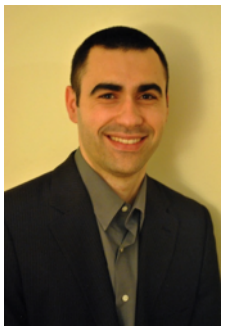
Exploit Kit detection:

Each Exploit Kit has its own characteristics that you can learn to recognize over time. Just like malware, Exploit Kits have certain patterns in their landing pages (specific strings in the URL or in the source code) as well as the type of exploits

they are serving. This is probably the most difficult part of all as getting such an understanding requires long hours of exposure to exploit code, CVEs, etc…

Whatever knowledge gathered can be translated into signatures that in turn can be applied by some of the rules mentioned above, making it possible to collect malicious URLs, gather your own statistics and see changes in the Exploit Kits' release cycle.

About the author:



Jerome Segura is a Senior Security Researcher at Malwarebytes with experience in both client and server side malware with a focus on web exploits research. He has built high interaction honey-clients to capture drive-by download attacks and has performed hundreds of web server remediations for infected WordPress and Joomla! sites.