

Threat Hunting with Yara

written by Mert SARICA | 1 September 2017

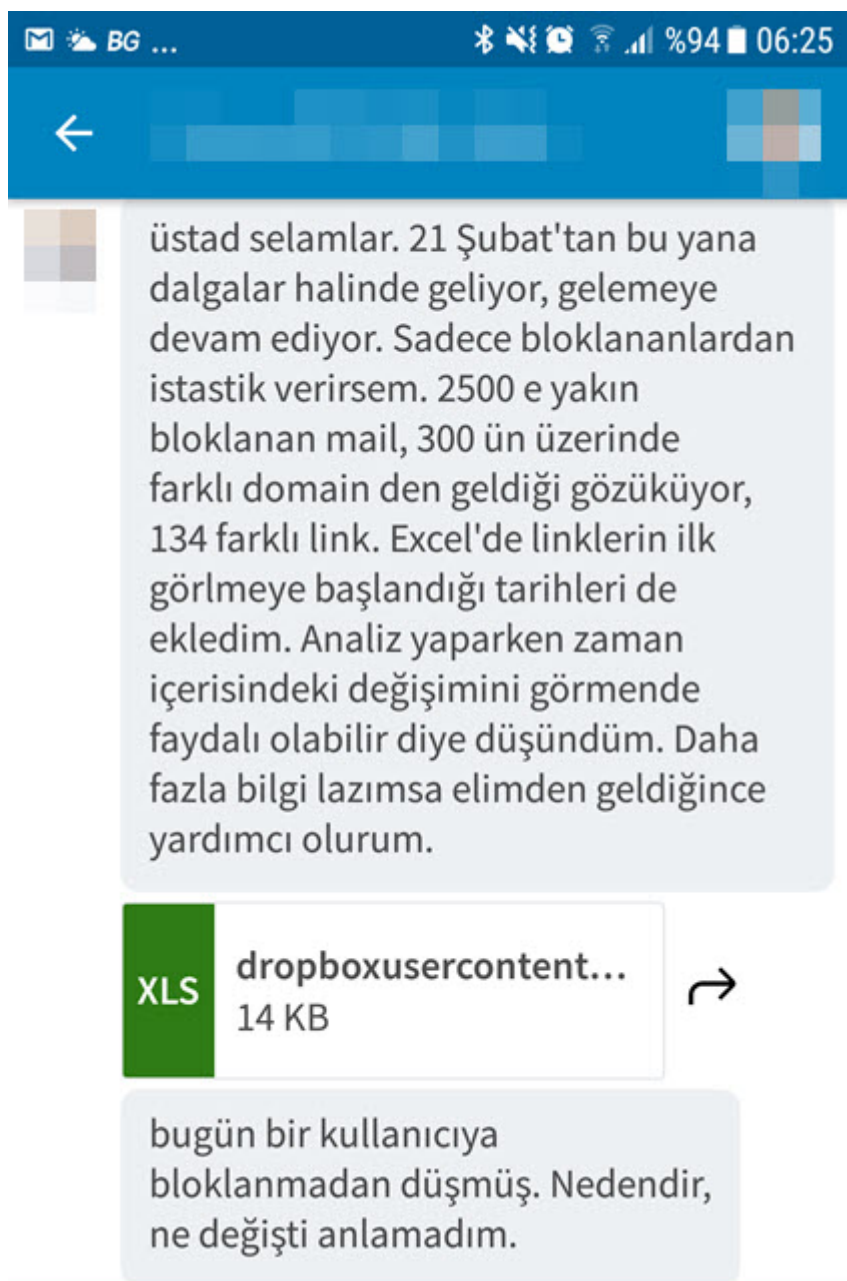
Throughout the world, the trend of encrypting data on end-user systems with malicious software (such as Cryptolocker) and then attempting to profit by selling the decryption key to users continues unabated. Occasionally, security researchers are able to decrypt the data that has been encrypted by malware due to flaws in the encryption algorithms being used. However, in most cases, users are often forced to pay the high ransom demanded by malicious individuals in order to regain access to their files. With each incident, the value of data backups becomes more evident. However, as long as there are users who act on impulse rather than heeding advice, it seems unlikely that malicious actors will easily give up on this lucrative avenue in the near future.

Indeed, with the rapid increase in cyber attacks, the ability to detect threats and respond to them promptly has become of great importance for organizations. In fact, visionary companies have started embracing the practice of cyber threat hunting, aiming to identify threats that can bypass existing security technologies within their networks and systems. When looking at the technologies that enable threat hunting, you will often find that many of them support Yara, a tool that allows you to write your own signatures. Yara provides the capability to create custom rules and signatures that can be used to search for specific patterns or indicators of compromise, enhancing the detection capabilities of organizations in the ever-evolving threat landscape.

When you look at the blog post published by Halil ÖZTÜRKÇİ in 2014 regarding Yara, you can see that Yara was predominantly used in digital forensics and memory analysis, particularly with the Volatility tool. However, today you can see that Yara is widely used in various fields, ranging from threat hunting to malware analysis, from commercial products like FireEye NX to open-source and free tools like x64dbg, and even in technologies such as full packet capture. This allows security professionals to define their own rules and signatures that can be used in security systems and devices with Yara support, independent of security vendors. While the idea of writing signatures may not be pleasant for security experts who have had challenging experiences with different security technologies in the past, the situation changes when it comes to Yara because writing rules with Yara is quite

simple, yet it provides significant added value, as confirmed by experience.

During the recent resurgence of the Cryptolocker outbreak, I noticed on social media platforms and the NetSec email list that some security systems and technologies were inadequate in detecting and preventing such outbreaks. In light of this situation, I wanted to draw attention to how defensive security experts can use a simple signature written with Yara to detect similar threats with modified content when faced with such a situation.



When we look at the Cryptolocker outbreak, we observe that numerous variants of Cryptolocker were being sent from different email addresses under the name

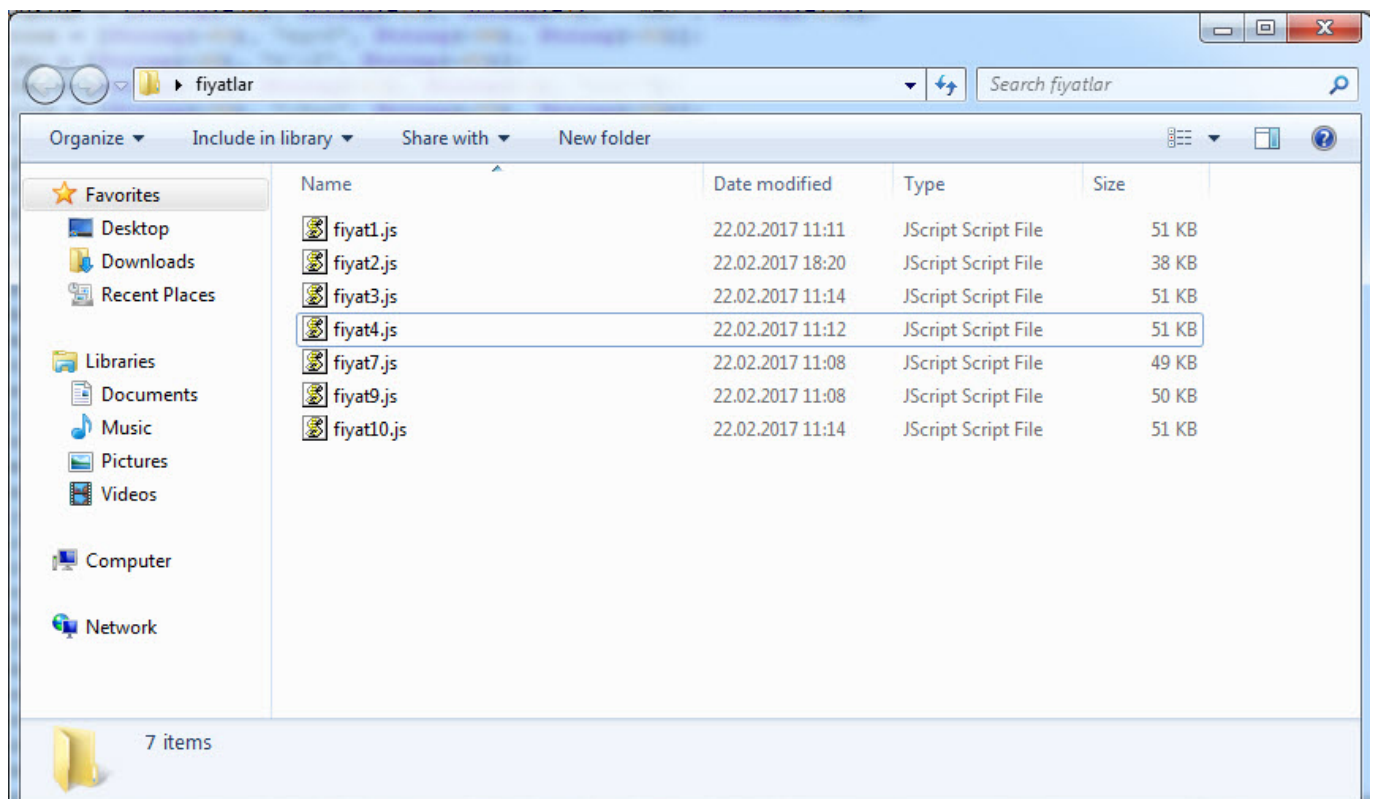
“priceX.zip” within a span of 24 hours. Each zip file contained a downloader with obfuscated JavaScript code, which, when executed, would download and run the encryption malware on the system.

Doruk Tekin rammer@tele2.at

Ekte gönderilen mallar için birim fiyat ve teslim süresi rica ederim.

<https://dl.dropboxusercontent.com/s/be9kvoym3hwjk69/fiyat3.zip>

İyi günler.



When it comes to detecting variants that differ in size, dimensions, and content, you can easily achieve this using Yara. First, when we list the sizes, we see that all variants except one are smaller than 55 KB. When examining the content of the files, although the content is completely

different, we can proceed by analyzing the complex code using the String function to extract the domain name and the downloaded file.

The image displays a Windows desktop environment with several open applications:

- Notepad++ (Left):** Contains JavaScript code defining variables like `var aniv = ["String", "String(-5)"]` and `var aniz = ["String(-100), "P", "String(-29)"]`.
- Notepad++ (Middle):** Contains JavaScript code defining variables like `var aniz = ["String(-5), "String(-6), "String(-8)"]` and `var aniz = ["String(-5), "String(-6), "String(-8)"]`.
- Notepad++ (Right):** Contains JavaScript code defining variables like `function erini() { return new String("esu"), new String("ljev"), new String("ljev") }` and `function erini() { return new String("esu"), new String("ljev"), new String("ljev") }`.
- Terminal (Center):** Shows the execution of `grep` commands to find the number of occurrences of the string `"String"` in various files:

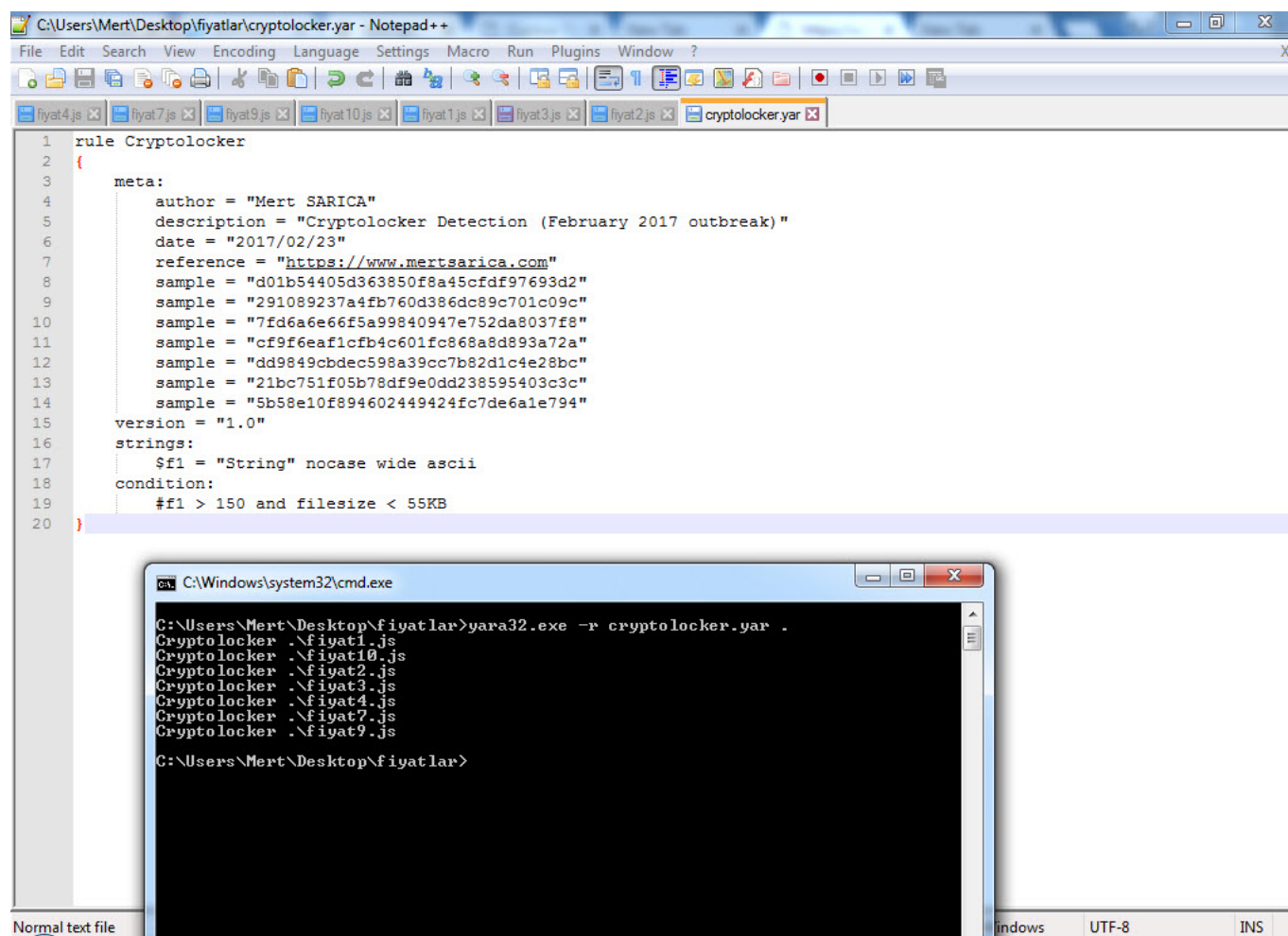
```
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat1.js | wc -l
442
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat2.js | wc -l
199
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat3.js | wc -l
444
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat4.js | wc -l
443
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat7.js | wc -l
430
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat9.js | wc -l
436
C:\Users\Mert\Desktop\fiyatlar>grep "String" fiyat10.js | wc -l
448
C:\Users\Mert\Desktop\fiyatlar>
```
- File Explorer (Bottom):** Shows a directory listing of files in `C:\Users\Mert\Desktop\fiyatlar\`. The files are listed with their names, dates, times, file types, and sizes.

Name	Date modified	Date created	File type	Size
fiyat1.js	22.02.2017 11:11	22.02.2017 07:45	JScript Script File	51 KB
fiyat2.js	22.02.2017 18:20		JScript Script File	38 KB
fiyat3.js	22.02.2017 11:14		JScript Script File	51 KB
fiyat4.js	22.02.2017 11:12		JScript Script File	51 KB
fiyat7.js	22.02.2017 11:08		JScript Script File	49 KB
fiyat9.js	22.02.2017 11:08		JScript Script File	50 KB
fiyat10.js	22.02.2017 11:14		JScript Script File	51 KB
grep.exe	14.04.2003 00:00		Application	79 KB
wc.exe	10.11.1999 23:00		Application	29 KB

At the bottom of the File Explorer window, a status bar shows: `JavaScript file`, `length: 51974`, `lines: 450`, `Ln: 6`, `Col: 33`, `Sel: 0 | 0`, `UNIX`, `UTF-8`, `INS`.

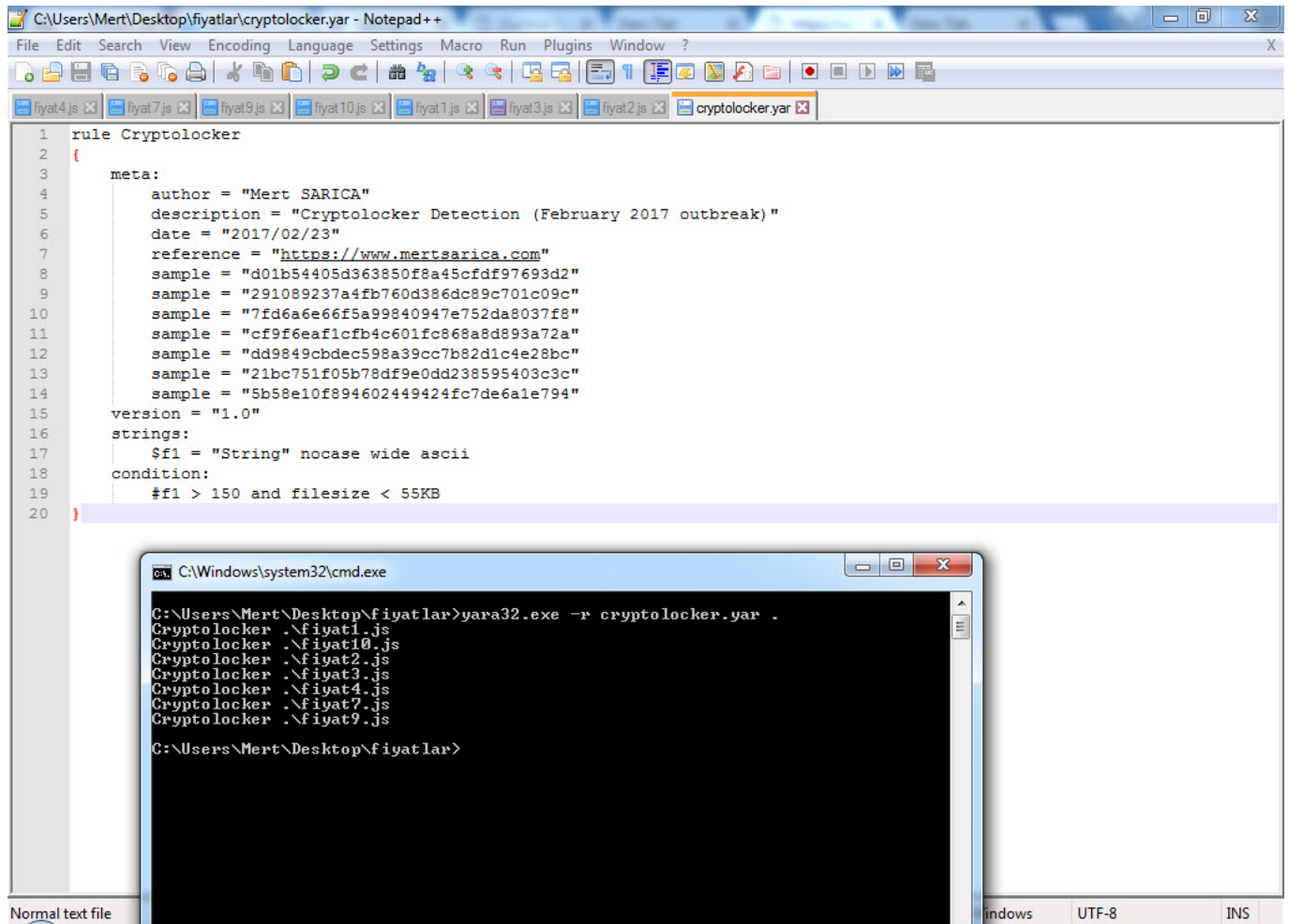
Under normal circumstances, assuming that the number of occurrences of the

String function in a file smaller than 55 KB would be less than 150 unless the file is suspicious, we can create a Yara signature using Yara keywords as follows. After confirming the correct functioning of our signature “cryptolocker.yar” and verifying that it can detect all the variants using the Yara tool, we can upload our signature to all security systems and technologies that support Yara. By doing so, we make significant progress in detecting new outbreaks and threats.



```
1 rule Cryptolocker
2 {
3   meta:
4     author = "Mert SARICA"
5     description = "Cryptolocker Detection (February 2017 outbreak)"
6     date = "2017/02/23"
7     reference = "https://www.mertsarica.com"
8     sample = "d01b54405d363850f8a45cfd97693d2"
9     sample = "291089237a4fb760d386dc89c701c09c"
10    sample = "7fd6a6e66f5a99840947e752da8037f8"
11    sample = "cf9f6eaf1cfb4c601fc868a8d893a72a"
12    sample = "dd9849cbdec598a39cc7b82d1c4e28bc"
13    sample = "21bc751f05b78df9e0dd238595403c3c"
14    sample = "5b58e10f894602449424fc7de6a1e794"
15  version = "1.0"
16  strings:
17    $f1 = "String" nocase wide ascii
18  condition:
19    #f1 > 150 and filesize < 55KB
20 }
```

```
C:\Windows\system32\cmd.exe
C:\Users\Mert\Desktop\fiyatlar>yara32.exe -r cryptolocker.yar .
Cryptolocker .\fiyat1.js
Cryptolocker .\fiyat10.js
Cryptolocker .\fiyat2.js
Cryptolocker .\fiyat3.js
Cryptolocker .\fiyat4.js
Cryptolocker .\fiyat7.js
Cryptolocker .\fiyat9.js
C:\Users\Mert\Desktop\fiyatlar>
```



The image shows a Notepad++ window with a YARA rule named 'Cryptolocker'. The rule includes a meta section with author 'Mert SARICA', description 'Cryptolocker Detection (February 2017 outbreak)', date '2017/02/23', and a reference to 'https://www.mertsarica.com'. It lists several sample hashes and a version '1.0'. The condition section specifies that the file must be a string, nocase, wide ascii, and larger than 150 bytes with a filesize less than 55KB.

```
1 rule Cryptolocker
2 {
3   meta:
4     author = "Mert SARICA"
5     description = "Cryptolocker Detection (February 2017 outbreak)"
6     date = "2017/02/23"
7     reference = "https://www.mertsarica.com"
8     sample = "d01b54405d363850f8a45cfd97693d2"
9     sample = "291089237a4fb760d386dc89c701c09c"
10    sample = "7fd6a6e66f5a99840947e752da8037f8"
11    sample = "cf9f6eaf1cfb4c601fc868a8d893a72a"
12    sample = "dd9849cbdec598a39cc7b82d1c4e28bc"
13    sample = "21bc751f05b78df9e0dd238595403c3c"
14    sample = "5b58e10f894602449424fc7de6a1e794"
15    version = "1.0"
16    strings:
17      $f1 = "String" nocase wide ascii
18    condition:
19      #f1 > 150 and filesize < 55KB
20 }
```

Below the Notepad++ window, a Windows command prompt window is open, showing the execution of 'yara32.exe' against several JavaScript files. The output shows that the files are identified as 'Cryptolocker'.

```
C:\Windows\system32\cmd.exe
C:\Users\Mert\Desktop\fiyatlar>yara32.exe -r cryptolocker.yar .
Cryptolocker .\fiyat1.js
Cryptolocker .\fiyat10.js
Cryptolocker .\fiyat2.js
Cryptolocker .\fiyat3.js
Cryptolocker .\fiyat4.js
Cryptolocker .\fiyat7.js
Cryptolocker .\fiyat9.js
C:\Users\Mert\Desktop\fiyatlar>
```

Hope to see you in the following articles.