# Technical Analysis of Code-Signed “Blister” Malware Campaign (Part 2)

By No items found.

Published: 2025-08-21 · Archived: 2026-04-05 17:08:41 UTC

The blister is a code-signed malware that drops a malicious DLL file on the victim’s system, which is then executed by the loader via rundll32.exe, resulting in the deployment of a RAT/ C2 beacon, thus allowing unauthorized access to the target system over the internet. Blister Malware campaigns have been active since 15 September 2021.

Part I of CloudSEK’s analysis provides a detailed understanding of how the loader functions. Part 2 will delve into the details of this campaign’s second stage, which is the .dll payload, and its internal working.

## Dissecting the Malicious DLL – Blister Malware

As discussed in Part 1, the Blister dropper drops the malicious `.dll` file in the `Temp directory` of the user, inside a newly created folder. This malicious `.dll` then carries out the second stage of the campaign, in which a RAT/ agent is deployed on the system to gain unauthorized access and steal data.

- The Blister dropper calls the function `LaunchColorCpl,` which is one of the functions exported by the .dll, via `rundll32.exe.`

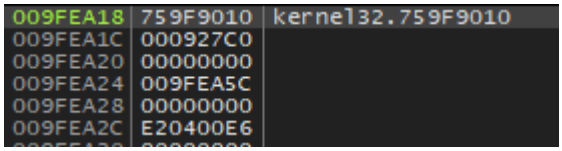
| Ordinal | Function RVA | Name Ordinal | Name RVA | Name |
|---|---|---|---|---|
| | | | | |
| (nFunctions) | Dword | Word | Dword | szAnsi |
| 00000001 | 00003A6C | 0000 | 00011E20 | LaunchColorCpl |
| 00000002 | 000037AA | 0001 | 00011E2F | DllCanUnloadNow |
| 00000003 | 00005D62 | 0002 | 00011E3F | DllGetClassObject |
| 00000004 | 000039F5 | 0003 | 00011E51 | DllMain |
| 00000005 | 000037D1 | 0004 | 00011E59 | DllRegisterServer |
| 00000006 | 000037DB | 0005 | 00011E6B | DllUnregisterServer |

*Functions exported by the malicious DLL*

### Anti-Analysis

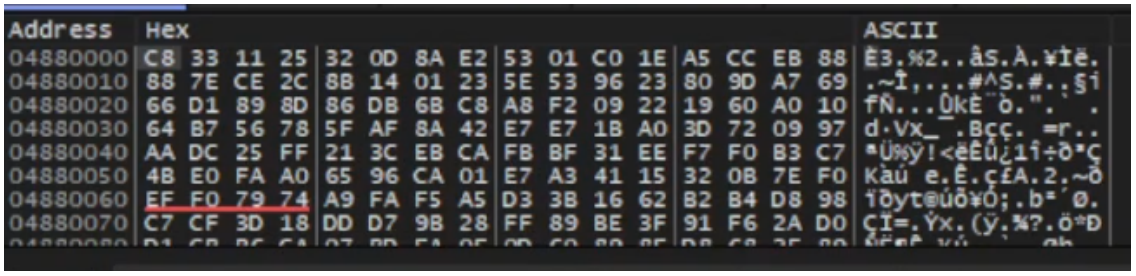
- The staging code is heavily obfuscated, and has a logic similar to a spaghetti code, to hinder analysis. All the calls to Windows APIs are obscured and dynamically resolved.
- The first thing that the staging code does is to make the malware go to sleep by calling the Sleep Windows API. This is a typical strategy used by most malicious codes to bypass security sandboxes and dynamic

testing of security products.



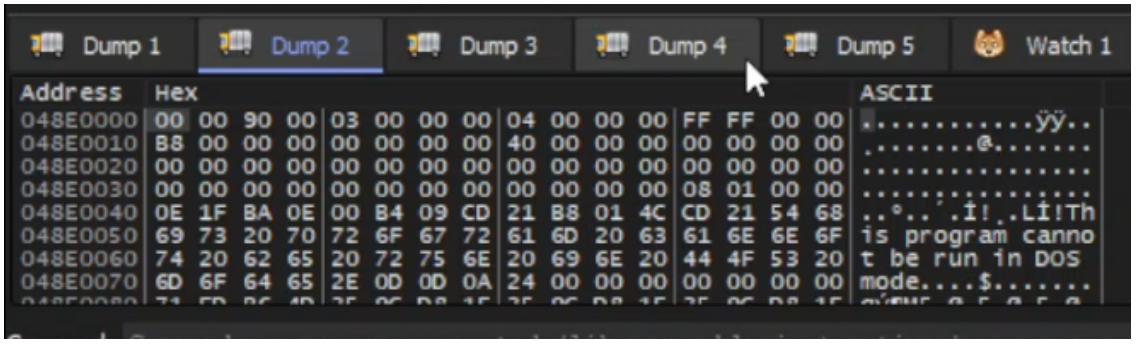

*Stackframe before the malware calls the Sleep Windows API*

- The hex value “927C0” is passed to `kernel32.759F9010` i.e the *Sleep function.* This value (927C0) translates to “600000” in decimal. Since the Sleep API takes arguments in milliseconds (ms), the 600000 ms get converted to 10 minutes.
- When the malware resumes from sleep, it fetches the final payload from the resource section of the PE file.



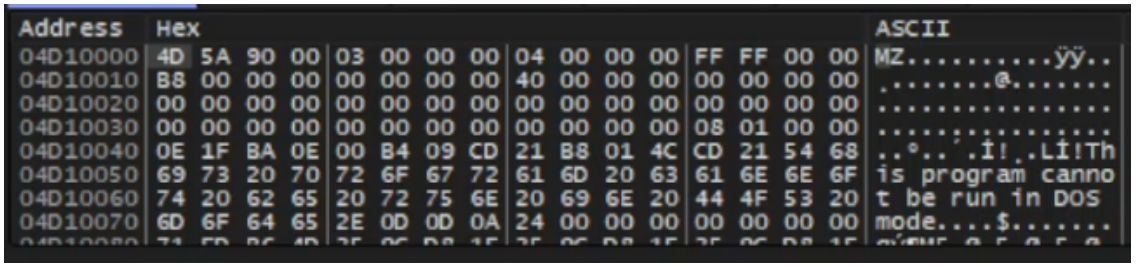

*Snippet of the protected payload stored in the memory*

- In the memory, the protected payload is decoded. The presence of a DOS header, in the payload bytes, confirms that the payload is in PE format and not a shellcode.




*Decrypted payload stored in the memory*

- An interesting observation from this analysis, is the addition of MZ byte after the decryption process. In the above image, the initial byte is not MZ, rather the MZ byte is later added at the beginning of the payload separately. This behavior is primarily for operational security.

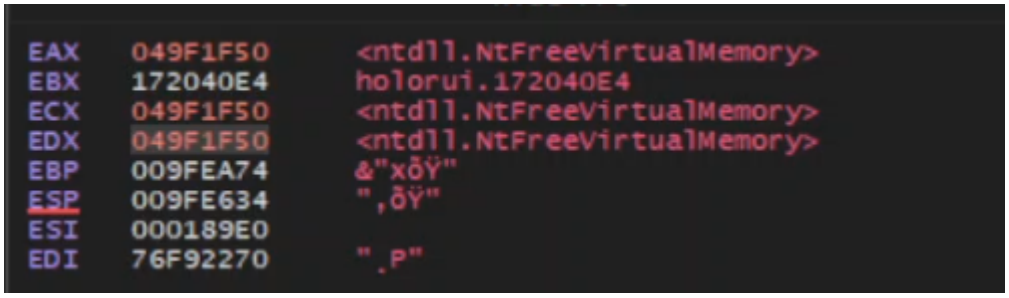


*Addition of the MZ byte after the decryption process*

## Process Hollowing

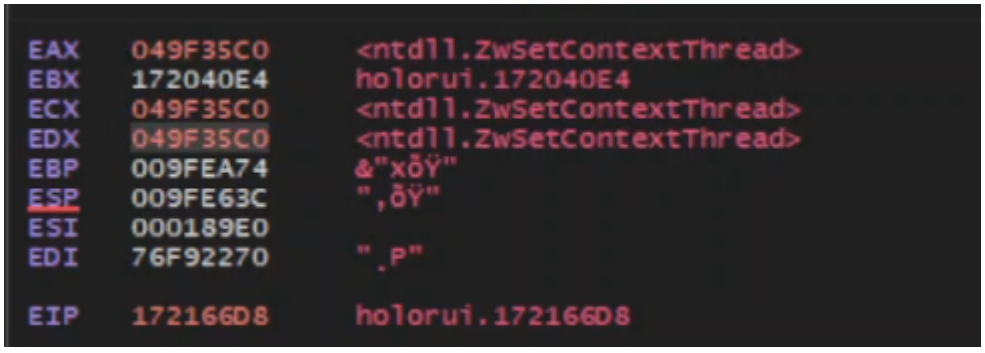
In general, process hollowing allows an attacker to change the content of a legitimate process from genuine code to malicious code before it is executed by carving out the code logic within the target process.

- After decrypting the final payload, the malware prepares for execution.
- This is done by creating a new process to deploy the extracted code and then performing process hollowing to execute the payload in the remote process. The staging code retrieves the *Rundll32.exe* location from the compromised system.




*Retrieval of the location of rundll32.exe*

- A new process of *Rundll32.exe* is created via the *CreateProcessInternalW* API in the suspended state.




*Creation of the new rendll32.exe*

- The malware uses the following Win32 APIs for process hollowing:
  - ZwUnmapViewOfSection
  - ZwReadVirtualMemory
  - ZwWriteVirtualMemory
  - ZwGetContextThread
  - ZwSetContextThread
  - NtResumeThread
- *ZwWriteVirtualMemory* is used to write malicious code into the target process.
- To make the thread of the new process point to newly written code, the attacker alters the entry point of the current thread via *ZwGetContextThread* and *ZwSetContextThread*.
- These functions are used to perform processor housekeeping activities on the data structure that stores the current context of the running thread. Process hollowing takes advantage of these features to make the

process thread run the attacker code.

## Conclusion

Given that threat actors are actively using valid code-signing certificates in Windows systems, to avoid detection by antivirus software, it is essential for network and endpoint security products to be updated with the malwares' latest Indicators of Compromise (IoCs). The latest IoCs for the Blister Malware are enumerated in Part 1 of the technical analysis.

---