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

cloudsek.com/technical-analysis-of-code-signed-blister-malware-campaign-part-2/

Anandeshwar Unnikrishnan February 17, 2022

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.

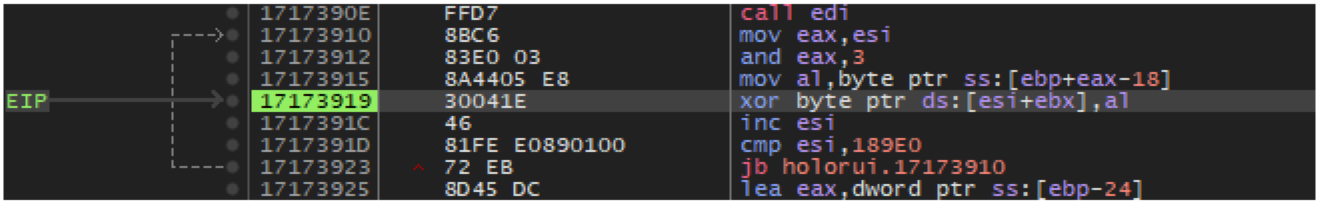
> The Blister dropper calls the function `LaunchColorCpl,` which is one of the functions exported by the .dll, via `rundll32.exe.`

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

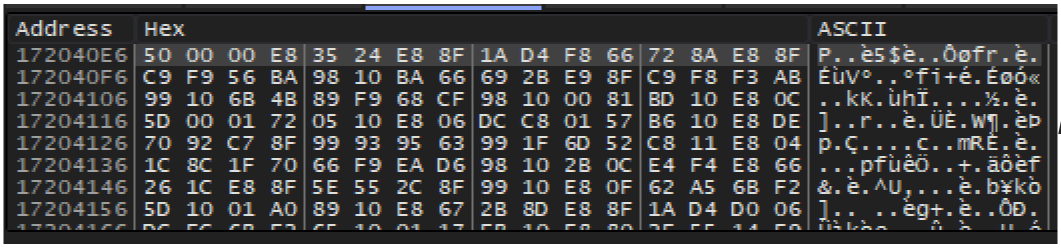
Functions exported by the malicious DLL

Staging

> The exported function `LaunchColorCpl` retrieves the staging code from the resource section of the PE file. This staging code is protected by a simple XOR encoding scheme.

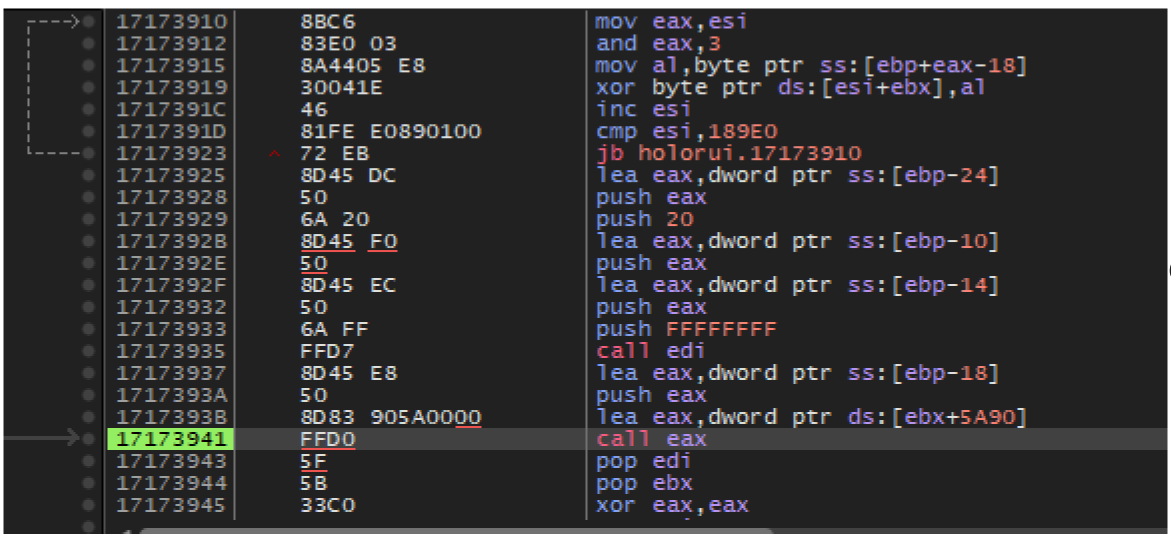


Code responsible for decoding the staging code



Encoded staging code in the resource section of the PE file

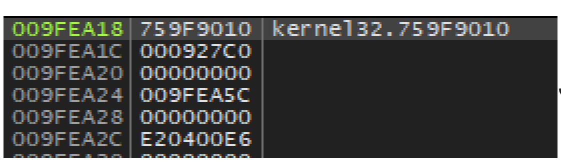
- After the iterative decoding of the staging code, the control is transferred to decoded code in the memory.
- The control flow is transferred to the staging code by calling the address in the EAX register.



Calling the address in the EAX register

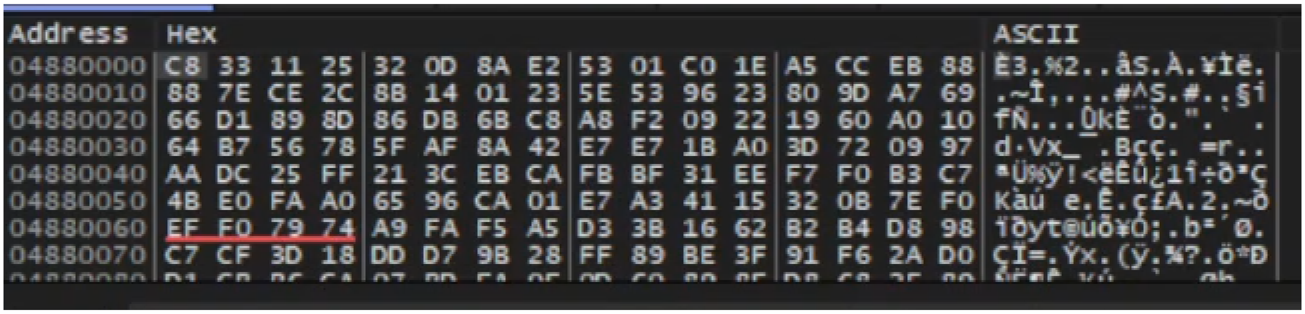
Anti-Analysis

- 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.



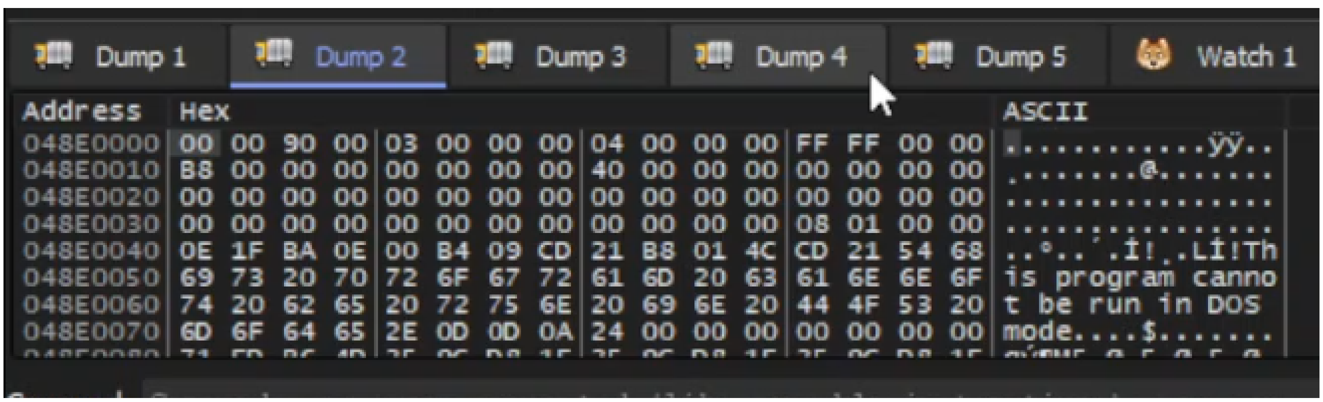
Stackframe before the malware calls the Sleep Windows API

- 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.



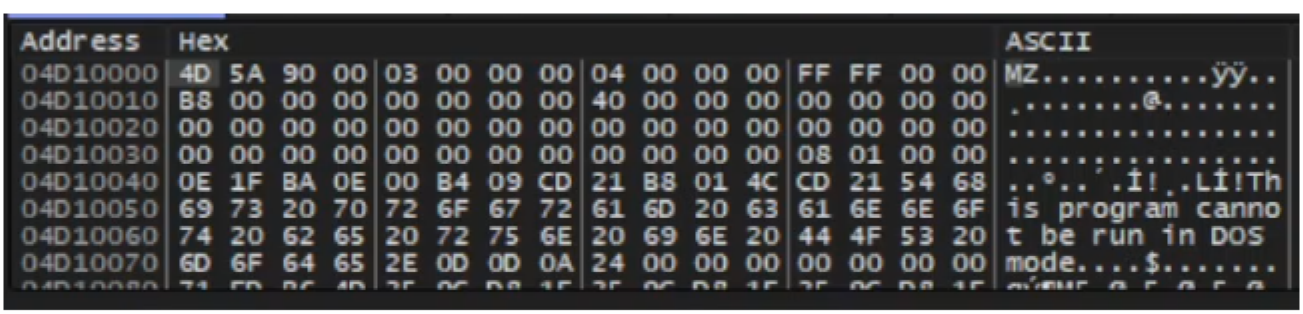
Snippet of the protected payload stored in the memory

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.

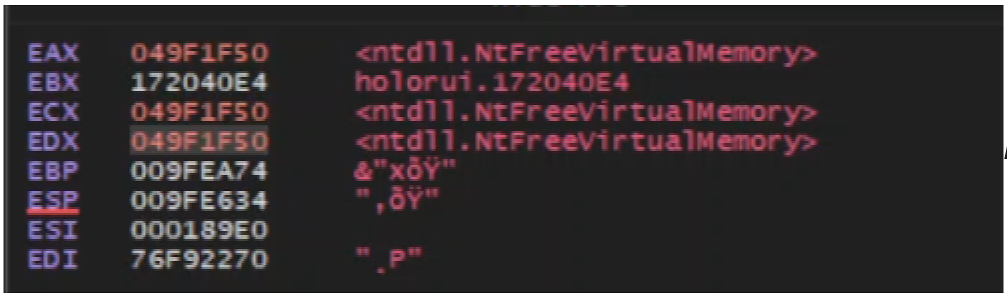


Addition of the MZ byte after the decryption process

Process Hollowing

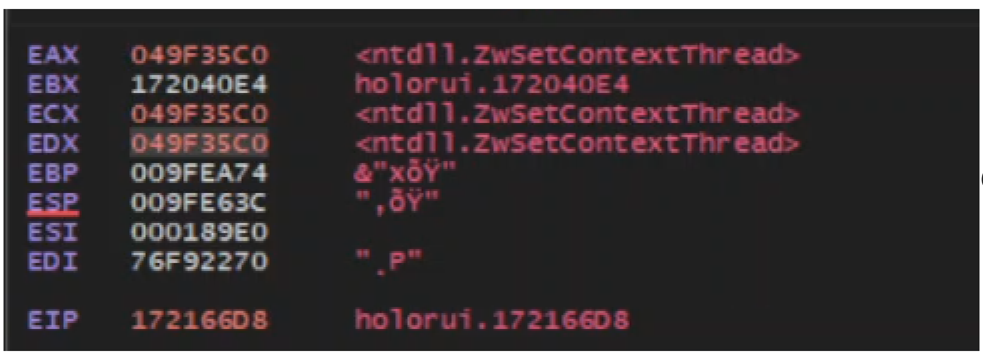
In general, process hollowing allows an attacker to change the content of a legitimate process from genuine code to malicious code before it is executed by carving out the code logic within the target process.

- 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.

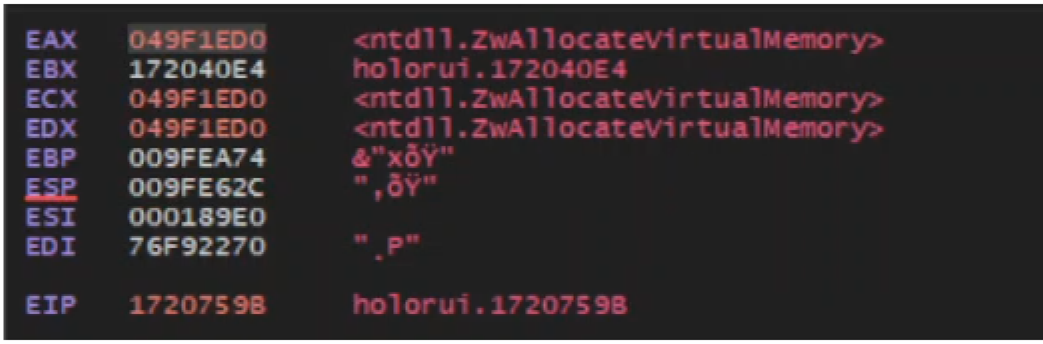


Creation of the new rendll32.exe

- 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.

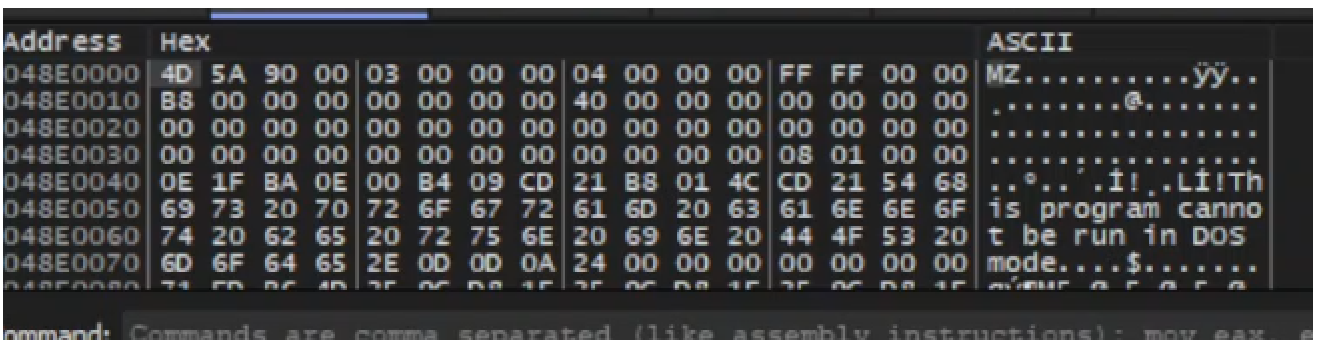
Step by Step Working of the DLL

The staging code allocates a new memory via *ZwAllocateVirtualMemory* to transfer the previously decrypted final payload.



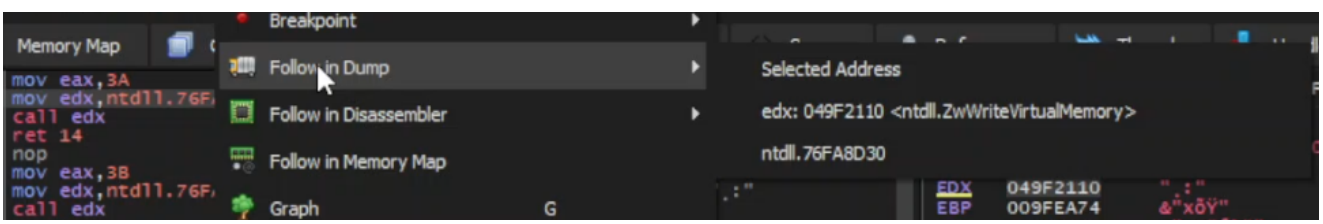
Allocation of new memory via ZwAllocateVirtualMemory

The payload is then copied to a newly created buffer.. Based on CloudSEK's testing on the extracted payload, one of the analyzed samples contained the *Raccoon stealer* as the final stage payload. However, other samples used *Cobalt Strike beacon* and *BitRAT* to compromise the target and gain unauthorized access.



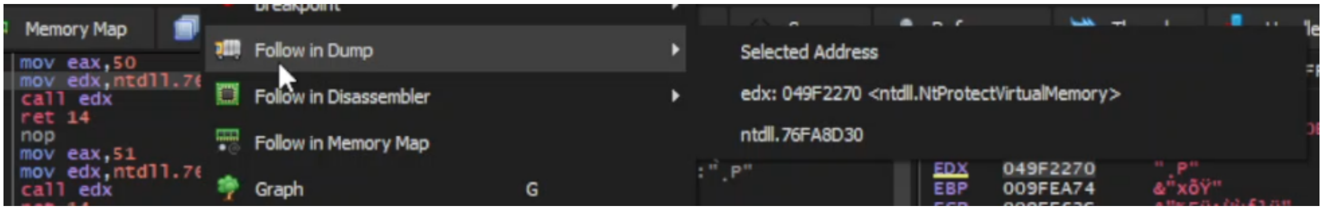
Moving the payload to a newly created buffer

The staging code then injects the code into the newly created remote process i.e *Rundll32.exe*.



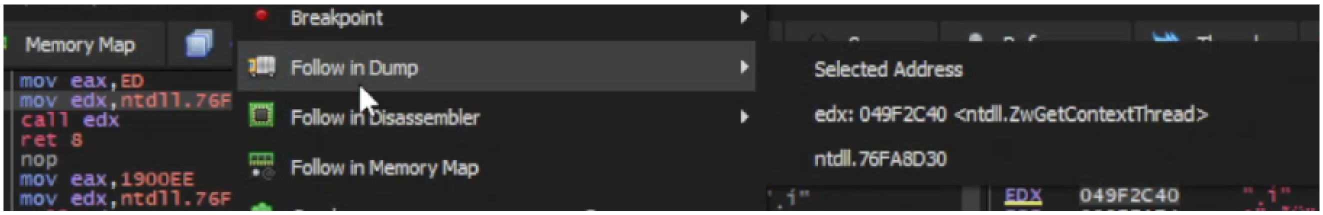
Code *injections into the newly created rendll32.exe*

Later, the memory protections are changed to appropriate ones for the execution of the residing code via *NTProtectVirtualMemory*.



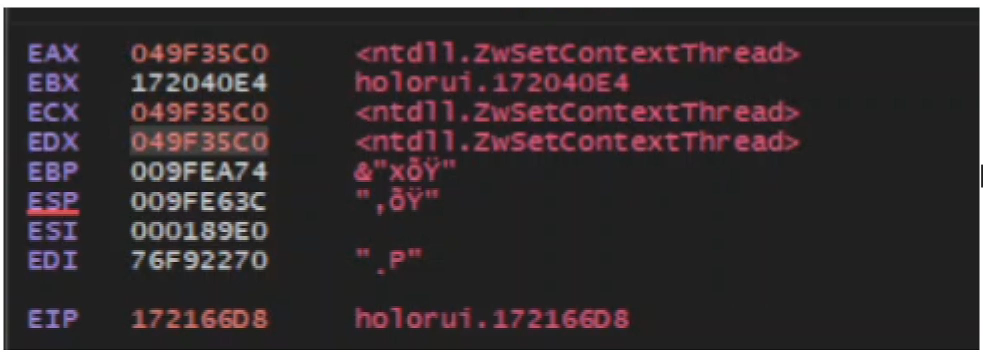
Alteration of the memory protections

The thread context is retrieved via *ZwGetContextThread API* to change the entry point of the thread to execute the payload injected into the remote process.



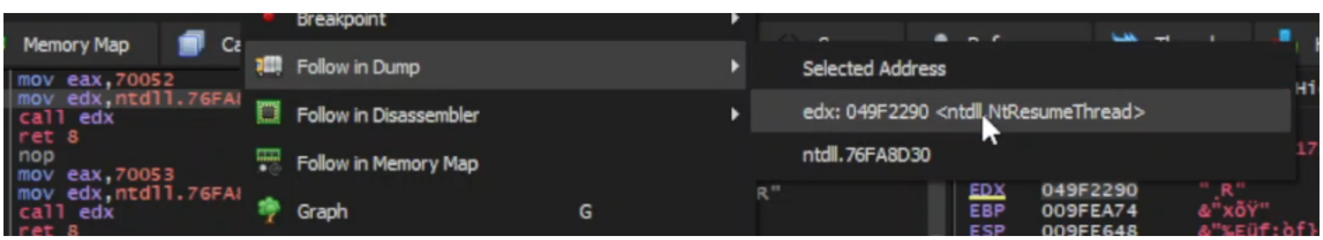
Addition of the MZ byte after the decryption process

The*ZwSetContextThread* is used to modify the thread entry point to that of the newly copied PE file.



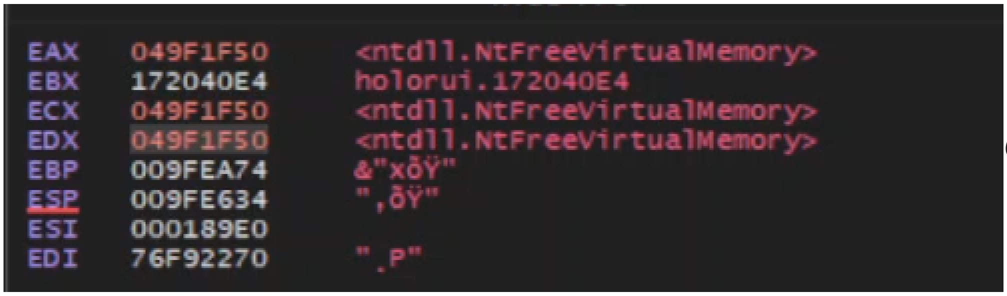
Modification of the thread entry point to the copied PE file

At the final stage of process hollowing, the suspended thread of the `Rundll32.exe` is resumed via `NtResumeThread`. Then the `Rundll32.exe` process starts executing the malicious code hollowed into it by the malware.



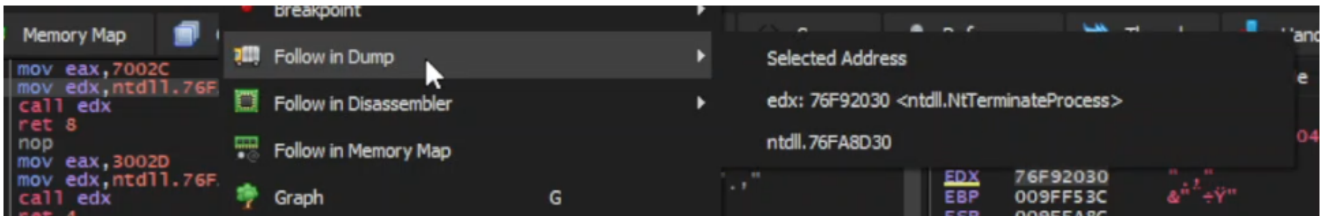
Resuming the suspended thread

In the clean-up process, the staging code uses *NtFreeVirtualMemory* to release the allocated memory, which holds the payload assembly, one by one.



Clean-up process releasing the allocated memory

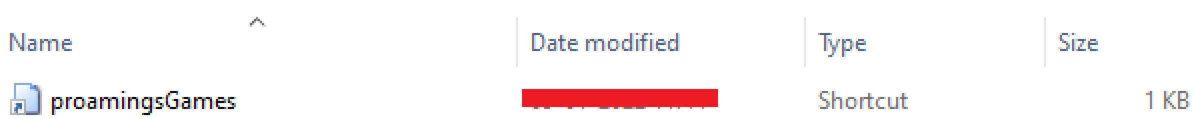
The current process used for staging is terminated via the `NtTerminateProcess` .



Termination of the current process

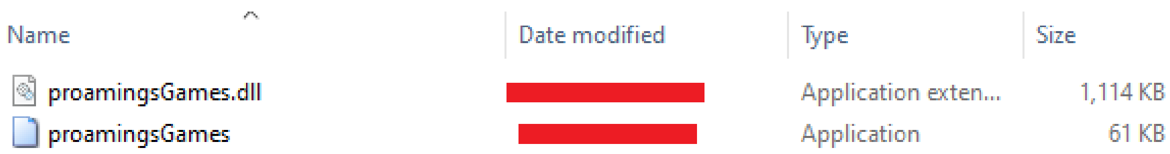
Blister Malware – Maintaining Persistence

- The Blister malware achieves persistence on the target system by creating an "Ink" file named *proamingsGames* in the `C:\Users\<username>\AppData\Roaming\Microsft\Windows\Start Menu\Startup` directory.
- Whenever the user logs in, `explorer.exe` executes any file in the `Startup` folder. As a result, when the user signs into the account, following the boot process, the malware runs as a child process of `explorer.exe` .



Ink file produced in the Startup directory

The target for the Ink file is set as `C:\ProgramData\proamingsGames\proamingsGames.dll,LaunchColorCpl` . Here, the malware copies the `Rundll32.exe` as `proamingsGames.exe` and the malicious .dll (initially into `C:\ProgramData\proamingsGames directory`) is dropped in the `Temp` folder.



Contents of the proamingsGames.dll file

Every time that the system powers up and the user logs in, the lnk file runs a malicious `.dll` through a renamed instance of `Rundll32.exe`.

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.

Author Details

Anandeshwar Unnikrishnan

Threat Intelligence Researcher , CloudSEK

Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that "a strong mind starts with a strong body." When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

Hansika Saxena

Total Posts: 2

Hansika joined CloudSEK's Editorial team as a Technical Writer and is a B.Sc (Hons) student at the University of Delhi. She was previously associated with Youth India Foundation for a year.

-
-

×

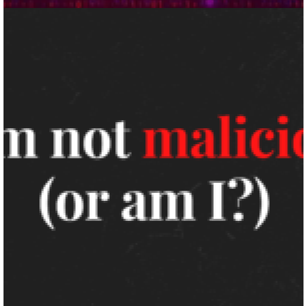
Anandeshwar Unnikrishnan

Threat Intelligence Researcher , CloudSEK

Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that "a strong mind starts with a strong body." When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

-
-

Latest Posts

- 
pandora
- 
(or am I?)
- 
- 
malware