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

X 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.

<u>Part I of CloudSEK's analysis</u> 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 *.dl1* file in the *Temp directory* of the user, inside a newly created folder. This malicious *.dl1* 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 *LaunchColorCp1*, 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
0000001	00003A6C	0000	00011E20	LaunchColorCpl
0000002	000037AA	0001	00011E2F	DIICanUnloadNow
0000003	00005D62	0002	00011E3F	DIIGetClassObject
0000004	000039F5	0003	00011E51	DIIMain
0000005	000037D1	0004	00011E59	DIIRegisterServer
0000006	000037DB	0005	00011E6B	DIIUnregisterServer

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.

•	1717390E	FFD7	call edi
>0 :	17173910	8BC 6	mov eax,esi
•	17173912	83E0 03	and eax,3
	17173915	8A4405 E8	mov al,byte ptr ss:[ebp+eax-18]
	17173919	30041E	<pre>xor byte ptr ds:[esi+ebx],al</pre>
	1717391C	46	inc esi
•	1717391D	81FE E0890100	cmp esi,189E0
i	17173923	72 EB	jb holorui.17173910
• :	17173925	8D45 DC	lea eax,dword ptr ss:[ebp-24]

*Code responsible for decoding the staging code* 

		-				_											
Address	Hex	¢															ASCII
																	Pè5\$èÔøfr.è.
172040F6	C9	F9	56	BA	98	10	BA	66	69	2B	E9	8F	C9	F8	F3	AB	ÉùVººfi+é.Éøó«
17204106	99	10	6B	4B	89	F9	68	CF	98	10	00	81	BD	10	E8	0C	kK.ùhϽ.è.
																	]rè.ÜÈ.W¶.èÞ
																	p.ÇcmRÉ.è.
																	pfùêÖ+.äôèf
17204146	26	1C	E8	8F	5E	55	2C	8F	99	10	E8	0F	62	A5	6B	F2	&.è.^U,è.b¥kò
17204156	5D	10	01	A0	89	10	E8	67	2B	8D	E8	8F	1A	D4	DO	06	]èg+.èÔĐ.
17204166	DC.	50	C D	<b>F D</b>	C.C.	10	01	47		10	<b>F</b> 0	80	ar.		4.4	50	1034255 TA 5 11 6

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.

	17173910	8BC 6	mov eax,esi
	17173912	83E0 03	and eax,3
	17173915	8A4405 E8	mov al, byte ptr ss:[ebp+eax-18]
	17173919	30041E	xor byte ptr ds:[esi+ebx],al
	1717391C	46	inc esi
	1717391D	81FE E0890100	cmp esi,189E0
L	17173923	∧ 72 EB	jb holorui.17173910
	17173925	8D45 DC	lea eax,dword ptr ss:[ebp-24]
	17173928	50	push eax
	17173929	6A 20	push 20
	1717392B		lea eax,dword ptr ss:[ebp-10]
		8D45 F0	
	1717392E	50	push eax
	1717392F	8D45 EC	lea_eax,dword ptr ss:[ebp-14]
	17173932	50	push eax
	17173933	6A FF	push FFFFFFFF
	17173935	FFD7	call edi
	17173937	8D45 E8	lea eax,dword ptr ss:[ebp-18]
	1717393A	50	push eax
	1717393B	8D83 905A0000	lea eax,dword ptr ds:[ebx+5A90]
	17173941	FFD0	call eax
	17173943	SF	pop edi
	17173944	5B	pop ebx
	17173945	33C0	xor eax.eax
	11113343	3300	Noi can, can

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.
- The hex value "927Co" is passed to
   *kerne132.759F9010* i.e the *Sleep function*.

   This value (927Co) translates to "600000" in decimal. Since the Sleep API takes
   arguments in milliseconds (ms), the 600000
   ms get converted to 10 minutes.

009FEA18	759F9010	kernel32.759F9010
009FEA1C	000927C0	
009FEA20	00000000	
009FEA24	009FEA5C	
009FEA28	00000000	
009FEA2C	E20400E6	
00055130	0000000	

Stackframe before the malware calls the Sleep Windows API

• When the malware resumes from sleep, it fetches the final payload from the resource section of the PE file.

Address	Hex	¢															ASCII
04880000	C8	33	11	25	32	0D	8A	E2	53	01	CO	1E	A5	CC	EB	88	È3.%2âs.À.¥Ìë.
04880010	88	7E	CE	2C	8B	14	01	23	5E	53	96	23	80	9D	A7	69	.~1,#^S.#§1
																	fÑÛkÈ 0.".`.
																	d·VxBcc. =r
																	■Ü%ÿ!<ëÊû¿1î÷ð¤Ç
																	Kaú e.Ê.ç£A.2.~ð
04880060	EF	F0	79	74	A9	FA	F5	A5	D3	3B	16	62	<b>B</b> 2	B4	D8	98	ïðyt⊜úð¥Ó;.b⁼′Ø.
04880070	C7	CF	3D	18	DD	D7	9B	28	FF	89	BE	3F	91	F6	2A	DO	CI=. Yx. (ÿ. %?. 0*D
04990090	<b>D</b> 1	<b>Cn</b>	nc	<b>C</b> *	07	nn		0	an	50	-			-	25	-	OFER VA Ch

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.

I Dump 1	Imp 2	💷 Dump 3	🕮 Dump 4 🛛 🕮	Dump 5 🛛 🧐 Watch 1
Address Hex 048E0000 00	00 90 00 03	00 00 00 04 00	00 00 FF FF 00 0	ASCII
048E0020 00	00 00 00 00	00 00 00 00 00	00 00 00 00 00 0	00@
048E0040 0E 048E0050 69	1F BA 0E 00 73 20 70 72	B4 09 CD 21 B8 6F 67 72 61 6D	01 4C CD 21 54 6 20 63 61 6E 6E 6	8°'.1! .L1!Th F is program canno O t be run in DOS
048E0070 GD	6F 64 65 2E	0D 0D 0A 24 00	00 00 00 00 00 0	0 mode\$

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.

Address																	ASCII
04D10000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZÿÿ
04D10010	88	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	
04D10020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
																	º'.1!L1!Th
																	is program canno
																	t be run in DOS
04D10070	GD	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode\$
04010080	74	-	20	40	20	~	-	15	25	~	-	15	20	20		15	avent ar ar a

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.

EAX	049F1F50	<ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory>
EBX	172040E4	holorui.172040E4
ECX	049F1F50	<ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory>
EDX	049F1F50	<ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory>
EBP	009FEA74	&"xõŸ"
ESP	009FE634	",õŸ"
ESI	000189E0	
EDI	76F92270	"_P"

Retrieval of the location of rundll32.exe

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

EAX	049F35C0	<ntdll.zwsetcontextthread></ntdll.zwsetcontextthread>
EBX	172040E4	holorui.172040E4
ECX	049F35C0	<ntdll.zwsetcontextthread></ntdll.zwsetcontextthread>
EDX	049F35C0	<ntdll.zwsetcontextthread></ntdll.zwsetcontextthread>
EBP	009FEA74	&"xôŸ"
ESP	009FE63C	".õŸ"
ESI	000189E0	,01
EDI	76F92270	" p"
EDI	10-92270	
EIP	172166D8	holorui.172166D8

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.

EAX	049F1ED0	<ntdll.zwallocatevirtualmemory></ntdll.zwallocatevirtualmemory>
EBX	172040E4	holorui.172040E4
ECX	049F1ED0	<ntdll.zwallocatevirtualmemory></ntdll.zwallocatevirtualmemory>
EDX	049F1ED0	<ntdll.zwallocatevirtualmemory></ntdll.zwallocatevirtualmemory>
EBP	009FEA74	&"xõŸ"
ESP	009FE62C	",ðŸ"
ESI	000189E0	
EDI	76F92270	
EIP	17207598	holorui.17207598

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.

			_	_	-											Tanaha
Address																ASCII
																MZÿÿ
048E0010	B8 (	00 00	00	00	00	00	00	40	00	00	00	00	00	00	00	@
048E0020	00 0	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
																º'.1!L1!Th
																is program canno
																t be run in DOS
48E0070	GD 6	5F 64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode\$
14850080	74 5	-	40	lar.	~	-	-	- 75	~	-	15	- 26	~		-	a fara
mmand: C	ommai	nds ;	TP	CO37	ma	sen	ara	ted	(1)	1 ke	35	sem	blv	17	str	uctions): mov eax

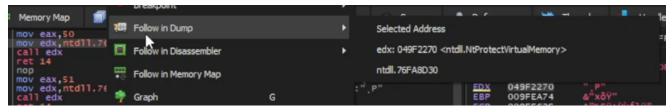
Moving the payload to a newly created buffer

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

	-		Breakpoint		1 II	4		~ ^	-		
Memory Map		1	Follow in Dump		۱.	Selected Addr	ess				
mov eax,3A mov edx,ntdl call edx	1.76F	Ø	Follow in Disassembler		•	edx: 049F211	.0 <ntdll< td=""><td>l.ZwWriteVi</td><td>irtualMemory&gt;</td><td></td><td>f</td></ntdll<>	l.ZwWriteVi	irtualMemory>		f
ret 14 nop			Follow in Memory Map			ntdll.76FA8D3	10				
mov eax,38 mov edx,ntdl		•@	Pollow in Memory Map					EDX 0	49F2110		
call edx		- 🥐	Graph	G	• •			EBP 0	09FEA74	&"xôŸ"	

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.

EAX EBX ECX EDX EBP ESP EST	049F35C0 172040E4 049F35C0 049F35C0 009FEA74 009FE63C 000189E0	<ntdll.zwsetcontextthread> holorui.172040E4 <ntdll.zwsetcontextthread> <ntdll.zwsetcontextthread> &amp;"xôŸ" ",ôŸ"</ntdll.zwsetcontextthread></ntdll.zwsetcontextthread></ntdll.zwsetcontextthread>
ESI EDI	000189E0 76F92270 172166D8	".P" holorui.172166D8

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.

	-	Breakpoint		1		
Memory Map 📄 Ca mov eax,70052		Follow in Dump		Þ	Selected Address	4
mov edx,ntdll.76FA	Ø	Follow in Disassembler		×	edx: 049F2290 <ntdl.ntresumethread></ntdl.ntresumethread>	
ret 8 nop mov eax,70053		Follow in Memory Map			ntdll.76FA8D30	
mov edx,ntdll.76FA call edx	•	Graph	G		R" <u>EDX</u> 049F2290 ".R" EBP 009FEA74 &"xöŸ" ESP 009FE648 &"⊾EŪf:òf	3

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.

EAX EBX ECX EDX	049F1F50 172040E4 049F1F50 049F1F50	<pre><ntdll.ntfreevirtualmemory> holorui.172040E4 <ntdll.ntfreevirtualmemory> <ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory></ntdll.ntfreevirtualmemory></pre>
EBP ESP ESI EDI	009FEA74 009FE634 000189E0 76F92270	&"xðŸ" ",ðŸ" ".₽"

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 "lnk" 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*.

Name	Date modified	Туре	Size
📄 proamingsGames		Shortcut	1 KB

#### Ink file produced in the Startup directory

The target for the lnk 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.

Name	Date modified	Туре	Size
proamingsGames.dll proamingsGames	_	Application exten Application	1,114 KB 61 KB

#### $Contents \ of \ the \ proamings Games. dll \ file$

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 <u>Part 1 of the technical analysis</u>.

#### 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.

#### <u>Hansika Saxena</u>

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.



