

## Unpacking Process Injection - DarkCloud 5537D725807584D2B92E993A1B5F5430

File type	File size
PE64	977.50 KiB

Scan	Endianness	Mode	Architecture	Type
Automatic	LE	64-bit	AMD64	GUI

▼ PE64

Operation system: Windows(Vista)[AMD64, 64-bit, GUI]	S	?
Linker: Microsoft Linker(14.0)	S	?
Compiler: Microsoft Visual C/C++(16.00)	S	?
Language: C/C++	S	?
(Heur)Packer: Compressed or packed data[Section names repeating + High entropy + Secti...	S	?

Breakpoints set.

```
<25022111cyxo.pdf.exe.OptionalHeader>  
<kernel32.dll.VirtualAlloc>  
<kernel32.dll.VirtualProtect>  
<kernel32.dll.IsDebuggerPresent>  
<kernel32.dll.WriteProcessMemory>  
<ntdll.dll.ZwAllocateVirtualMemory>  
<ntdll.dll.ZwMapViewOfSection>  
<ntdll.dll.NtCreateSection>
```

First VirtualAlloc followed by A call to **VirtualAlloc** was observed, which allocated a region of memory within the target process's address space. This was immediately followed by a call to **WriteProcessMemory**, where the **lpBuffer** parameter pointed to data that was written into the memory region returned by **VirtualAlloc**.

This behavior is characteristic of code injection techniques, where memory is first allocated with **VirtualAlloc** (typically with **PAGE\_EXECUTE\_READWRITE** or **PAGE\_READWRITE** protection), and then malicious or arbitrary code is written into that memory using **WriteProcessMemory**.

The key detail is that the **lpBaseAddress** argument in **WriteProcessMemory** matched the **address returned by VirtualAlloc**, indicating a deliberate setup for injecting or staging code. The **lpBuffer** in this case pointed to the source of the payload, either from the current process or a loaded buffer.

	00007FFDF391BCA8	CC	int3	
	00007FFDF391BCAC	CC	int3	
	00007FFDF391BCAD	CC	int3	
	00007FFDF391BCAE	CC	int3	
	00007FFDF391BCAF	CC	int3	
00007FFDF391BCB0	48:FF25 695B0400	3B	qword ptr ds:[writeProcessMemory]	writeProcessMemory
00007FFDF391BCB1		CC	int3	
00007FFDF391BCB2		CC	int3	
00007FFDF391BCB3		CC	int3	
00007FFDF391BCB4		CC	int3	
00007FFDF391BCB5		CC	int3	
00007FFDF391BCB6		CC	int3	
00007FFDF391BCB7		CC	int3	
00007FFDF391BCB8		CC	int3	
00007FFDF391BCB9		CC	int3	
00007FFDF391BCBA		CC	int3	
00007FFDF391BCBB		CC	int3	
00007FFDF391BCBC		CC	int3	
00007FFDF391BCBD		CC	int3	
00007FFDF391BCBE		CC	int3	
00007FFDF391BCBF		CC	int3	
00007FFDF391BCD0		CC	int3	
00007FFDF391BCD1		CC	int3	
00007FFDF391BCD2		CC	int3	
00007FFDF391BCD3		CC	int3	
00007FFDF391BCD4		CC	int3	
00007FFDF391BCD5		CC	int3	
00007FFDF391BCD6		CC	int3	
00007FFDF391BCD7		CC	int3	
00007FFDF391BCD8		CC	int3	
00007FFDF391BCD9		CC	int3	
00007FFDF391BCDA		CC	int3	
00007FFDF391BCDB		CC	int3	
00007FFDF391BCDC		CC	int3	
00007FFDF391BCDD		CC	int3	
00007FFDF391BCDE		CC	int3	
00007FFDF391BCDF		CC	int3	
00007FFDF391BCE0		CC	int3	
00007FFDF391BCE1		CC	int3	
00007FFDF391BCE2		CC	int3	
qword ptr ds:[00007FFDF3961820 <kernel32.writeProcessMemory>]=<kernelbase.writeProcessMemory>				
.text:00007FFDF391BCB0 kernel32.dll:33BCB0 #380B0 <writeProcessMemory>				
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5
Address	Hex	ASCII		
0000020A68AF0000	00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF40	00007FF6D6D7886F return to 25022111
0000020A68AF0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF48	00007FF6D6D7D732 return to 25022111
0000020A68AF0020	00 00 00 00 FF FF 00 00 00 00 00 00 00 00 00 00	....YY.....	0000008C2618EF50	0000020A68AF0000
0000020A68AF0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF58	0000000000001000
0000020A68AF0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF60	0000000000000000
0000020A68AF0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF68	0000008C2618F580
0000020A68AF0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF70	00007FF6D6E23C09 25022111cyxo.pdf.0
0000020A68AF0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF78	00007FF6D6D7722E return to 25022111
0000020A68AF0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF80	00007FFDF38F7034 return to kernel32
0000020A68AF0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF88	00007FFDF3804A24 mtd11.N15MBCodePag
0000020A68AF00A0	00 00 00 00 80 8F 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EF90	0000008C2618F050
0000020A68AF00B0	3A 9A 61 00 00 00 00 00 00 00 00 00 00 00 00 00	..A.....PP.wf...	0000008C2618EF98	0000008C2618F550
0000020A68AF00C0	02 02 00 00 3E FD 7A 00 00 00 00 00 00 00 00 00 00	...9Z?+.....	0000008C2618EFA0	0000008C2618F580
0000020A68AF00D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFA8	0000008C2618F5A8
0000020A68AF00E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFB0	0000000000000000
0000020A68AF00F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFC0	0000008C2618F5C0
0000020A68AF0100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFC8	0000008C2618F568
0000020A68AF0110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFD0	0000008C2618F560
0000020A68AF0120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0000008C2618EFD8	0000000000000000

### Parameters of WriteProcessMemory from ChatGPT

```

c
Copy Edit
BOOL WriteProcessMemory(
    HANDLE hProcess,
    LPVOID lpBaseAddress,
    LPCVOID lpBuffer,
    SIZE_T nSize,
    SIZE_T *lpNumberOfBytesWritten
);

```

### 🧠 Parameter Breakdown:

Parameter	Description
<code>hProcess</code>	Handle to the <b>target process</b> (must have <code>PROCESS_VM_WRITE</code> and <code>PROCESS_VM_OPERATION</code> rights)
<code>lpBaseAddress</code>	Pointer to the <b>target address in the remote process</b> where you want to write
<code>lpBuffer</code>	Pointer to the <b>local buffer</b> containing the data you want to write
<code>nSize</code>	Number of bytes to write
<code>lpNumberOfBytesWritten</code>	Optional pointer to receive how many bytes were written (can be <code>NULL</code> )

lpBuffer parameter points to the address on the allocated memory from VirtualAlloc

[illegible]

Current register Values. RSI is pointing to MSBuild.exe

RAX	00007FFDF3918CB0	<kernel32.WriteProcessMemory>
RBX	0000000055862785	'..'
RCX	0000000000000080	
RDY	0000000000400000	
RBP	0000008C2618F7A0	
RSP	0000008C2618EF38	
RSI	0000008C2618F4E0	"KC:\\windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe"
RDI	00000000A608C8E3	
R8	0000020A68A33E80	
R9	0000000000001000	
R10	00007FFDF572D304	ntdll.00007FFDF572D304
R11	0000000000000246	L'z'
R12	0000000094374349	
R13	00000000B6F832C6	
R14	0000000000000001	
R15	000000002B430F34	
RIP	00007FFDF32A05F0	<kernelbase.writeProcessMemory>
RFLAGS	0000000000010202	
ZF	0	PF 0 AF 0
OF	0	SF 0 DF 0
CF	0	TF 0 IF 1
LastError	00000000 (ERROR_SUCCESS)	
LastStatus	C0000139 (STATUS_ENTRYPOINT_NOT_FOUND)	
GS	002B FS 0053	
ES	002B DS 002B	
CS	0033 SS 002B	
< Default (x64 fastcall) 5		

Apparently MSBuild is already running. I forgot to put the breakpoints for Process Creation (CreateProcessA, CreateProcessW, CreateProcessInternalW etc)

svchost.exe	1560	2.59 MB	NT AUTHORITY\SYSTEM	Host Proce
sihost.exe	4036	6.8 MB	DESKTOP-26...	Shell Infrast
x64dbg.exe	1276 0.07	43.89 MB	DESKTOP-26...	x64dbg
25022111CY...	4044	2.38 MB	DESKTOP-26...	Microsoft V
MSBuild....	1136	804 kB	DESKTOP-26...	MSBuild.ex

Process Writing to MSBuild.exe

00007FFDF32A05F0	48:8BC4	mov rax, rsp	writeProcessMemory
00007FFDF32A05F3	48:8958 08	mov qword ptr ds:[rax+8], rbx	
00007FFDF32A05F7	4C:8948 20	mov qword ptr ds:[rax+20], r9	
00007FFDF32A05F8	4C:8940 18	mov qword ptr ds:[rax+18], r8	
00007FFDF32A05FF	48:8950 10	mov qword ptr ds:[rax+10], rdx	
00007FFDF32A0603	55	push rbp	
00007FFDF32A0604	56	push rsi	rsi: "KC:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe"
00007FFDF32A0605	57	push rdi	

WriteProcessMemory is called again and the lpBuffer has the MZ signature header of 4D 5A indicating that an executable is being loaded into memory into MSBuild.exe

The screenshot shows a debugger window with the 'Dump 1' pane displaying a memory dump. The first two bytes of the dump are 4D 5A, which is the MZ signature header for an executable. The 'Registers' pane shows the RAX register pointing to the kernel32.dll module. The 'Call Stack' pane shows the current function call as WriteProcessMemory.

Looking into the memory of MSBuild.exe using SystemInformer shows that the executable is being loaded into it at the base address of 0x400000. Note in the WriteProcessMemory call above, lpBaseAddress is pointing to 0x401000 for the next run.

The screenshot shows the SystemInformer application with the 'Memory' tab selected. It displays a list of memory regions for MSBuild.exe. The first region is at 0x400000, which is the base address of the executable. The 'Properties' window for this region is open, showing details about the memory allocation.

It's possible to save and dump the file from SystemInformer by clicking on save and creating a new file from that region of memory.

Dumped file seems to be written in VB and is now 32bit instead of 64bit like the previous one.

The screenshot displays the PE32 file analysis tool interface. At the top, key file properties are shown: File type (PE32), File size (352.00 KiB), Base address (00400000), and Entry point (004031a4). Below these are various analysis tools like File info, Memory map, Disasm, Hex, Strings, Signatures, VirusTotal, MIME, Visualization, Search, Hash, Entropy, Extractor, and YARA. The main section displays file metadata: Sections (0003), Time date stamp (2025-04-23 16:17:50), Size of image (00058000), and Resources (Manifest, Version). The Scan section shows Endianness (LE), Mode (32-bit), Architecture (I386), and Type (GUI). The PE32 details section lists: Operation system: Windows(95)[I386, 32-bit, GUI], Linker: Microsoft Linker(6.0), Compiler: Visual Basic(6.00.8041)[Native], and Language: VB. The bottom section includes Signatures, Flags, Database, Directory, Log, and a Scan button. The right sidebar contains checkboxes for Advanced, Demangle, Shortcuts, Options, About, and Exit.

File type	File size	Base address	Entry point
PE32	352.00 KiB	00400000	004031a4

Sections	Time date stamp	Size of image	Resources
0003	2025-04-23 16:17:50	00058000	Manifest, Version

Scan	Endianness	Mode	Architecture	Type
Automatic	LE	32-bit	I386	GUI

PE32	S	?
Operation system: Windows(95)[I386, 32-bit, GUI]	S	?
Linker: Microsoft Linker(6.0)	S	?
Compiler: Visual Basic(6.00.8041)[Native]	S	?
Language: VB	S	?

Signatures	Flags	Database	Directory	Log	Scan
					138 msec

VT was able to analyze this file with a lot of detection and some vendors labelling it as Dark Cloud.

41

/ 72

Community Score

41/72 security vendors flagged this file as malicious

1fa57ee601cbb64e7996216f4e78986cf94f167a61cc7ab6cc194e02ef50dd11

Size352.00 KB

Last Analysis Datea moment ago

EXE

DETECTION

DETAILS

BEHAVIOR

COMMUNITY

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

Basic properties

MD5

dd34bd3c5098580f9eff618ad26e704f

SHA-1

bacbc52029c24f5b172f4fd09613bfe92feef448

SHA-256

1fa57ee601cbb64e7996216f4e78986cf94f167a61cc7ab6cc194e02ef50dd11

Vhash

0350367d05fzdz33z

Authentihash

f515a938619377ca0ccb70d9ee37d64d38d57c02e6cb43b8b7bf3b0e517d4f4

Imphash

d3bc8d371a35ac07e9d67e895d2c4962

Rich PE header hash

4b4293436909365b5e47e83b7c138034

SSDEEP

6144:P8d1/w5KA811J8GpF6nuTmOOUIUPheKdFkJsMjy/3v/10IT3:EjVKKJj6GmZUZPheZJfjy/IN

TLSH

T160748B27E720601EF2A3858089B9619765263D791A88EC1BF7418F6C34714D3A8F631F

File type

Win32 EXE executable windows win32 pe peexe

Magic

PE32 executable (GUI) Intel 80386, for MS Windows

TrID

Win32 Executable Microsoft Visual Basic 6 (48.1%) | UPX compressed Win32 Executable (15.8%) | Win32 EXE Yoda's Crypter (15.5%) | Win64 Executable (generic) (6.1%) ...

DetectItEasy

PE32 | Compiler: Visual Basic (6.00.8041) [Native] | Linker: Microsoft Linker (6.0)

Magika

PEBIN

File size

352.00 KB (360448 bytes)

PEID packer

Microsoft Visual Basic v5.0/v6.0

History

Creation Time

2025-04-23 23:17:50 UTC

First Submission

2025-04-24 03:53:12 UTC

Last Submission

2025-04-24 03:53:12 UTC

Last Analysis

2025-04-24 03:53:12 UTC

PE-Bear on the dumped file. To fix the import address table; the Raw address is initially changed to match the virtual address and subtracting the values of the raw addresses (Next raw address subtracted by the previous one) and applying the value to raw size. Afterwards, match the virtual size with the raw size and try to fill in the memory for the last slot in the raw size. The image base in the Optional Hdr tab may also be needed to match the address from where we dumped.

Looks like there is no need to repair the IAT.

Disasm: .text	General	Strings	DOS Hdr	Rich Hdr	File Hdr	Optional Hdr	Section Hdrs	Imports	Resources	BoundImports
Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.		
> .text	1000	55000	1000	54DD8	60000020	0	0	0		
> .data	56000	1000	56000	F3C	C0000040	0	0	0		
> .rsrc	57000	1000	57000	8E8	40000040	0	0	0		

Some clear text configurations can be found running Strings on the dumped file



```
00031BA0 rollandcraig32@gmail.com
00031BD8 order@oleonidas.gr
00031C04 +}bVmK6i0mnd|
00031C24 mail.oleonidas.gr
```

```

00031C64 @StrBotToken
00031C84 @ChatID
00031C98 \Microsoft\Windows\Templates\firefly.exe
00031CF0 HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\
00031D80 boots
00031D8C RegWrite
00031F2C http://showip.net
00031F54 http://www.mediacollege.com/internet/utilities/show-ip.shtml
00031FF0 ?id=
0003200C Uploader
00032024 POST

```

Seems like in the original dump1.bin there were two executables inside. I split them and did a quick look and it looks like the payloads are packed by upx.

 DumpDarkCloudMaybe.bin	4/23/2025 10:23 PM	BIN File	340 KB
 DumpDLLMaybe.bin	4/23/2025 10:22 PM	BIN File	13 KB

▼ PE32			
Operation system: Windows(2000)[I386, 32-bit, DLL]	S	?	
Linker: Microsoft Linker(9.00.21022)	S	?	
Compiler: Microsoft Visual C/C++(15.00.21022)[C]	S	?	
Language: C	S	?	
Tool: Visual Studio(2008)	S	?	
Packer: UPX(3.03)[NRV,brute]	S	?	
(Heur)Packer: Compressed or packed data[Imports like UPX (v2.90-3.XX) + Sections like UP...	S	?	
▼ Overlay: Binary[Offset=0x00028600,Size=0x0002c6f0]			
Unknown: Unknown	S	?	

Strings content is still the same but Ida can interpret the dumped files properly now.