

Teaching an old RAT new tricks

Posted on April 21, 2016 by Joseph Landry



Attackers have been successfully deploying RATs for years to remotely control users systems - giving them full access to the victim's files or resources such as cameras, recording key strokes, or downloading further malware. Traditionally RATs have been deployed when a user opens an email attachment, or downloads a file from a website or peer-to-peer network. In both cases, these vectors involve use of files to deliver the payload - which are easier to detect.

Recently we detected a more sophisticated technique that a handful of countries across Asia are actively using to infect systems with RATs. This new technique ensures that the payload/file remains in memory through its execution, never touching the disk in a de-encrypted state. In doing so, the attacker can remain out of view from antivirus technologies, and even 'next-generation' technologies that only focus on file-based threat vectors. Also, the samples analyzed have the ability detect the presence of a virtual machine to ensure it's not being analyzed in a network sandbox.

And finally it's important to highlight that the RAT itself is not new. In fact this technique can be used to deliver any "known" RAT to a victim's system. We analyzed this sample against our SentinelOne EPP to confirm it does not evade our behavior-based detection mechanisms. This is due to the fact that we're monitoring all processes at the user-space/kernel-space interface - and because all communication between the application and the kernel must be unencrypted, we detect the sample at both process-injection points.

Samples Analyzed

Main Sample

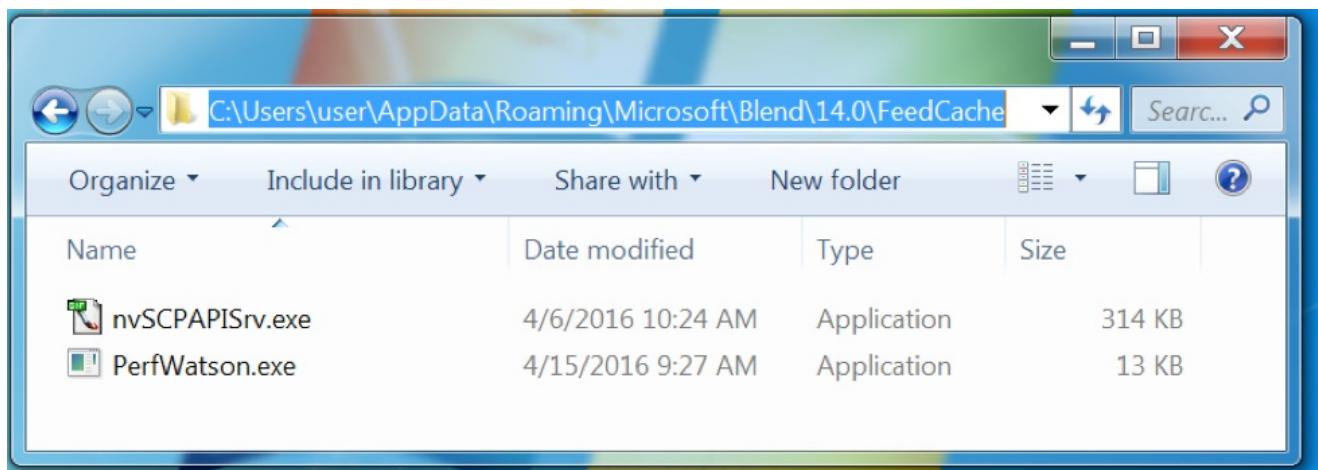
- **Format:** Win32 PE .NET 2.0
- **SHA-256 sum:**
b7cfc7e9551b15319c068aae966f8a9ff563b522ed9b1b42d19c122778e018c8
- **HSA-1 sum:** **3b1ac573509281cdc0b6141f8ea6ed3af393b554**
- **MD5 sum:** **65752e742d643d121ee7e826ab65dc9b**
- **File size:** 321024 bytes (324 kb)

Unpacked Samples

- **Main Sample**
 - e5c71180f117270538487cd9b9b1b6d8 - Packed "Benchmark" DLL
 - 9e05fb115bd4e85cfc0e32c72aa721be - Monitor (PerfWatson.exe)
 - d740ed3f33ca4cef3a6aa717f94bf52a - NanoCore RAT dumped from memory

Behavioral Analysis

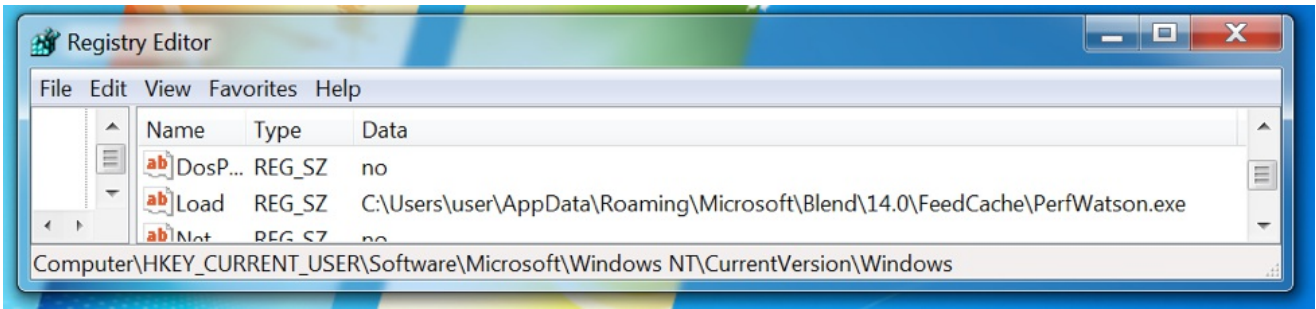
When run, the binary will copy itself to %APPDATA%\Microsoft\Blend\14.0\FeedCache\nvSCPAPISrv.exe and extracts a second binary named PerfWatson.exe



It then executes both binaries.

File Name	MD5	SHA-256	Size	Parent MD5	Parent SHA-256	Parent Size	Parent Name	Parent SHA-256
b7cfc7e9551b15319c068aae9...	1.00	21,508 K	22,508 K	3256	iheardof		Evan Jett	
b7cfc7e9551b15319c068aa...	0.14	19,372 K	17,328 K	3176	iheardof		Evan Jett	
PerfWatson.exe	0.55	16,372 K	16,480 K	3836	PerfWatson2.exe			
nvSCPAPISrv.exe	0.85	20,920 K	19,896 K	2844	iheardof		Evan Jett	
nvSCPAPISrv.exe	0.03	11,360 K	11,608 K	3396	iheardof		Evan Jett	

For persistence, a registry key is created at HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\Load pointing to the PerfWatson.exe binary.



Finally, the RAT tries connecting back to its control server:

- azona2015.chickenkiller.com:1617 (TCP)
- azona.chickenkiller.com:1617 (TCP)

chickenkiller.com is owned by a free dynamic DNS service.

At the time of this writing, the DNS records still exist, but the address they resolve to appears to be down.

60 105.753484	172.16.108.128	8.8.8.8	DNS	87 Standard query 0x0be8 A azona2015.chickenkiller.com
61 105.822747	8.8.8.8	172.16.108.128	DNS	103 Standard query response 0x0be8 A azona2015.chickenkiller.com A 154.66.17.101
62 105.823277	172.16.108.128	154.66.17.101	TCP	66 49177 → 1617 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
63 108.841210	172.16.108.128	154.66.17.101	TCP	66 [TCP Retransmission] 49177 → 1617 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
64 109.566886	154.66.17.101	172.16.108.128	TCP	60 1617 → 49173 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0

Unpacking

"Benchmark" .NET DLL

The main executable contains an XOR encrypted .NET DLL in its .NET managed resources and the logic to unpack it. This DLL contains the logic to unpack and inject the RAT as well as monitor the application, PerfWatson.exe. This DLL is referred to as "Benchmark" because that is the .NET namespace it uses.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000020	73	6F	75	72	63	65	52	65	61	64	65	72	2C	20	6D	73	sourceReader.ms
00000030	63	6F	72	6C	69	62	2C	20	56	65	72	73	69	6F	6E	3D	corlib.Version=
00000040	34	2E	30	2E	30	2E	30	2C	20	43	75	6C	74	75	72	65	4.0.0.0.Culture
00000050	3D	6E	65	75	74	72	61	6C	2C	20	50	75	62	6C	69	63	=neutral.Public
00000060	4B	65	79	54	6F	6B	65	6E	3D	62	37	37	61	35	63	35	KeyToken=b77a5c5
00000070	36	31	39	33	34	65	30	38	39	23	53	79	73	74	65	6D	61934e089#System
00000080	2E	52	65	73	6F	75	72	63	65	73	2E	52	75	6E	74	69	.Resources.Runti
00000090	6D	65	52	65	73	6F	75	72	63	65	53	65	74	02	00	00	meResourceSet.
000000A0	00	01	00	00	00	00	00	00	00	50	41	44	50	41	44	50PADFADP
000000B0	19	A5	02	00	00	00	00	00	C3	00	00	00	02	BC	10	00Å.....k+
000000C0	00	00	00	20	00	A4	00	00	68	6D	F3	BB	27	DA	0B	AFhmo>'0s
000000D0	B8	64	78	8D	06	54	19	D8	61	B8	9B	FA	12	2D	0D	49	.dx-T+0a_luI-I
000000E0	3B	0D	51	EF	C3	9F	82	A0	CA	D5	C2	0D	16	B7	85	B7	..QiA!!!E0A+-I-
000000F0	62	F3	71	AA	F7	6C	9A	83	35	64	8B	CB	5A	80	2B	2B	b0q?+l! 5d EZ ++
00000100	C7	12	A1	15	24	EF	F1	27	5C	4B	FA	C4	44	C1	90	0B	C!l-siñ`KúADA ♂

After decrypting the resource, it is linked into the process using `System.Reflection.Assembly.Load(byte[])`. This method is documented on [MSDN here](#). Using this method, the DLL will never be written to the filesystem. This technique could have been chosen by the developers to evade antivirus detection.

```
internal static object LoadAssembly(object object_0)
{
    return Assembly.Load(object_0);
}
```

Under the hood, Assembly.Load(), uses a call to the win32 api call CreateFileMappingW() with the hFile parameter set to INVALID_HANDLE_VALUE. According to MSDN, this will create a mapped file that is backed by the paging filesystem, not the standard filesystem. A layer below CreateFileMapping, the system call NtCreateSection is invoked.

Stack	Data	Procedure	Called from
0025BD4C	75D8A276	ntdll.NtCreateSection	KERNELBASE.CreateFileMappingW+0A5
0025BD50	0025BD0C	Arg1 = 25BD0C	
0025BD54	000F0007	Arg2 = 0F0007	
0025BD58	00000000	Arg3 = 0	
0025BD5C	0025BD98	Arg4 = 25BD98	
0025BD60	00000004	Arg5 = 4	
0025BD64	08000000	Arg6 = 8000000	
0025BD68	00000000	Arg7 = 0	
0025BDA8	6DCA1878	KERNEL32.CreateFileMappingW	mcorwks.6DCA1872
0025BDAC	FFFFFFFF	hFile = INVALID_HANDLE_VALUE	
0025BDB0	00000000	pSecurity = NULL	
0025BDB4	00000000	Protect = 0	
0025BDB8	00000000	MaxSizeHigh = 0	
0025BDBC	0000A400	MaxSizeLow = 0A400	
0025BD00	00000000	Name = NULL	
0025BD04	6DEF6258	mcorwks.6DEF6258	mcorwks.6DEF6253
0025BD08	FFFFFFFF	Arg1 = -1	
0025BD0C	00000000	Arg2 = 0	
0025BD10	00000004	Arg3 = 4	
0025BD14	00000000	Arg4 = 0	
0025BD18	0000A400	Arg5 = 0A400	
0025BD1C	00000000	Arg6 = 0	
0025BE18	6DEF6481	mcorwks.6DEF6481	mcorwks.6DEF647C
0025BE1C	01959F00	Arg1 = 1959F00	

After the empty file is created, it is mapped into memory using the Win32 API call MapViewOfFileEx. The layer below this invokes the system call NtMapViewOfSection.

The screenshot shows a debugger window titled "CPU - main thread, module mcorwks". The assembly window displays the following instructions:

```

6DCA1884 40 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 CALL 6DC317D0
6DCA1888 53 00 00 00 00 00 00 00 48 00 00 00 00 00 00 00 PUSH DWORD PTR DS:[EBP+1C]
6DCA188C 8B 95 68 13 C3 66 MOV ESI, DWORD PTR DS:[<<KERNEL32.MapViewOfFileEx>]
6DCA1892 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+18]
6DCA1896 5D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+14]
6DCA189A 5C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+10]
6DCA189E 5B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+0C]
6DCA18A2 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+8]
6DCA18A6 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CALL ESI
6DCA18AA 89 45 F0 MOV DWORD PTR SS:[EBP-10], EAX
6DCA18AE 85 C0 TEST EAX, EAX
6DCA18B2 8B 1D 88 43 17 66 MOV EBX, DWORD PTR DS:[6E174388]
6DCA18B6 89 50 EC MOV DWORD PTR SS:[EBP-14], EBX
6DCA18BA 74 04 C1 FF 21 0A JZ 6DEC1979
6DCA18BE 5D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 TEST EBX, EBX
6DCA18C2 74 04 85 00 00 00 00 00 00 00 00 00 00 00 00 JNZ 6DEC1920
6DCA18C6 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CMP DWORD PTR SS:[EBP-10], 0

```

The registers window shows the following values:

- EAX: 0025BD04
- ECX: 7FFDE000
- EDX: 00000000
- EBX: 00459894
- ESP: 0025BD4C
- EBP: 0025BD80
- ESI: 76D91796 (KERNEL32)
- EDI: 00000000
- EIP: 75D8E017 (KERNELBASE)

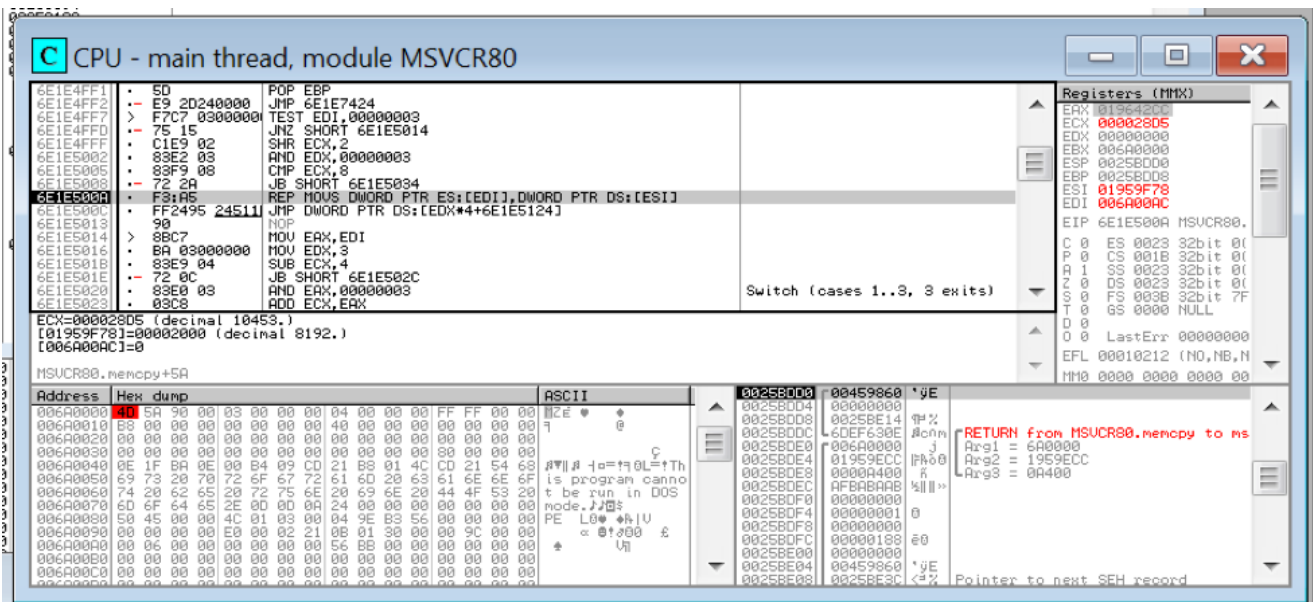
The disassembly window shows the following instructions:

```

0025BD4C 80 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 CALL 6DCA1884
0025BD50 53 00 00 00 00 00 00 00 48 00 00 00 00 00 00 00 PUSH DWORD PTR DS:[EBP+1C]
0025BD54 8B 95 68 13 C3 66 MOV ESI, DWORD PTR DS:[<<KERNEL32.MapViewOfFileEx>]
0025BD58 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+18]
0025BD5C 5D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+14]
0025BD60 5C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+10]
0025BD64 5B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+0C]
0025BD68 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PUSH DWORD PTR SS:[EBP+8]
0025BD6C 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CALL ESI
0025BD70 89 45 F0 MOV DWORD PTR SS:[EBP-10], EAX
0025BD74 85 C0 TEST EAX, EAX
0025BD78 8B 1D 88 43 17 66 MOV EBX, DWORD PTR DS:[6E174388]
0025BD7C 89 50 EC MOV DWORD PTR SS:[EBP-14], EBX
0025BD80 74 04 C1 FF 21 0A JZ 6DEF6258
0025BD84 5D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 TEST EBX, EBX

```

Now, a call to memcpy() is used to copy the decrypted DLL into the newly allocated address range.

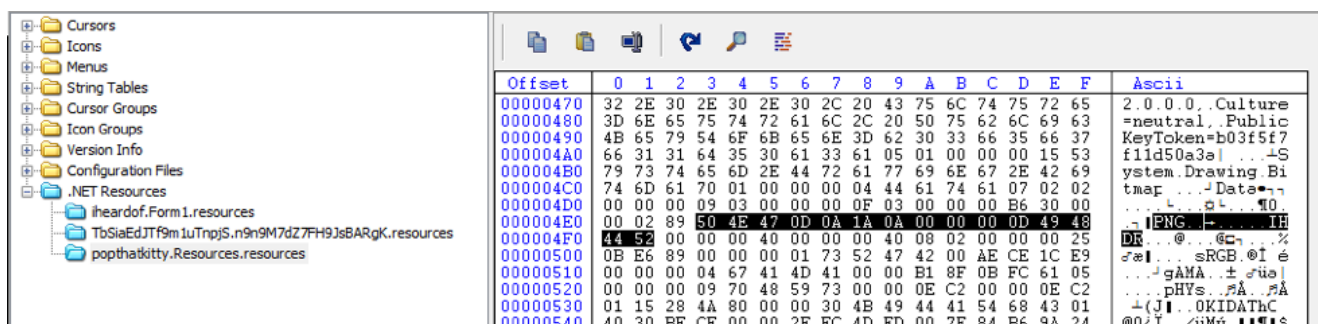


Unpacking Settings and NanoCore

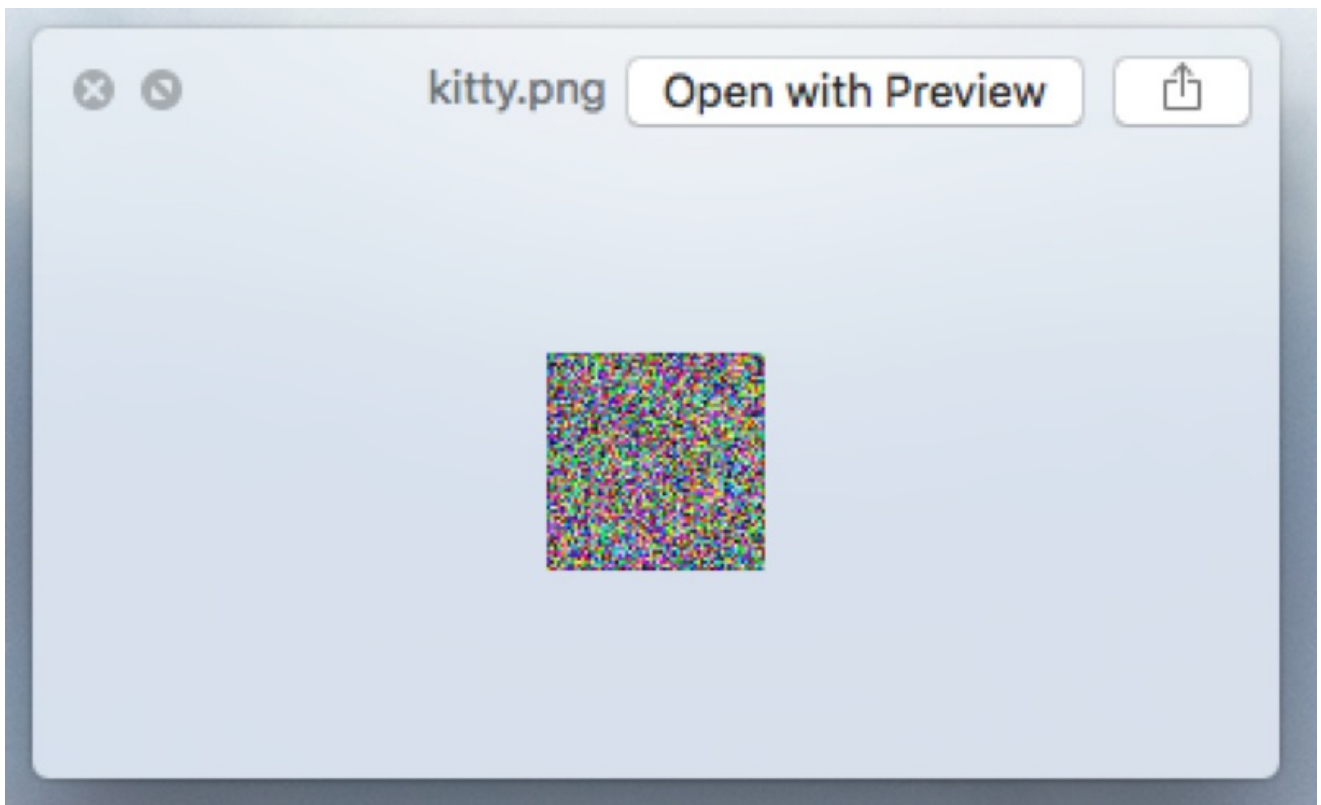
The settings for "Benchmark" and the NanoCore executable are serialized, DES encrypted, spliced, and stored across multiple PNG files as pixel data. The PNG files are concatenated and stored in the .NET managed resources of the main executable.

Some of the settings that can be configured are:

- Exit if a virtual machine is detected
- Paths and filenames to store PerfWatson.exe and NanoCore
- Display a message box to the user
- Delete ":Zone.Identifier" information for files from NTFS ADS.
- Download an encrypted file from the Internet, decrypt it, and run it.
- Monitor the Injected process



After viewing one of these images, it is obvious they are not used to conveying visual information to a human eye.



After writing a short python script, I was able to extract all 19 PNG files. If you have robot eyes, you can see a cat.



Here is a C# decompilation of the method used to extract the information out of the pixel data.

```

// Benchmark.EntryPoint
internal static byte[] ImagesToData(Image[] images)
{
    MemoryStream memoryStream = new MemoryStream();
    for (int i = 0; i < images.Length; i++)
    {
        Image image = images[i];
        Rectangle rect = new Rectangle(Point.Empty, image.Size);
        MemoryStream memoryStream2 = new MemoryStream();
        image.Save(memoryStream2, ImageFormat.Bmp);
        Bitmap expr_40 = new Bitmap(memoryStream2);
        BitmapData bitmapData = expr_40.LockBits(rect, ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
        byte[] array = new byte[Marshal.ReadInt32(bitmapData.Scan0)];
        Marshal.Copy(new IntPtr(bitmapData.Scan0.ToInt32() + 4), array, 0, array.Length);
        expr_40.UnlockBits(bitmapData);
        memoryStream2.Close();
        memoryStream.Write(array, 0, array.Length);
    }
    memoryStream.Close();
    return memoryStream.ToArray();
}

```

Once everything is decrypted, the set options are executed, and the NanoCore RAT payload is injected into a new child process. The method of injection is discussed later.

Unpacking PerfWatson.exe

Now that "Benchmark" is loaded into memory, it is tasked with copying the main executable and extracting PerfWatson.exe to %APPDATA%\Microsoft\Blend\14.0\FeedCache\.

PerfWatson.exe is stored inside Benchmark as a base64 encoded string. There is no encryption or obfuscation outside of the base64 encoding.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00005700	45	72	72	6F	72	01	08	01	00	03	00	00	00	00	00	C0	Error
00005710	00	42	B6	01	00	C0	00	42	AE	54	56	71	51	41	41	4D	.B! .A.B@TVqQAAM
00005720	41	41	41	41	45	41	41	41	41	2F	2F	38	41	41	4C	67	AAAAEAAAAA//8AALg
00005730	41	41	41	41	41	41	41	41	41	51	41	41	41	41	41	41	AAAAAAAAAAAAQAAAAAA
00005740	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAAAAAA
00005750	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAAAAAA
00005760	41	41	41	41	41	41	41	41	41	67	41	41	41	41	41	34	AAAAAAAAAAAgAAAAA4
00005770	66	75	67	34	41	74	41	6E	4E	49	62	67	42	54	4D	30	fug4AtAnNIbgBTM0
00005780	68	56	47	68	70	63	79	42	77	63	6D	39	6E	63	6D	46	hVGhpcyBwcm9ncmF
00005790	74	49	47	4E	68											42	tIGNhbm5vdCBiZSB
000057A0	79	64	57	34	67											31	ydW4gaW4gRE9TIG1
000057B0	76	5A	47	55	75											41	vZGUuDQ0KJAAAAAA
000057C0	41	41	41	42	51											43	AAABQRQAATAEEAEC
000057D0	62	73	31	59	41											41	bs1YAAAAAAAAAAAAOA
000057E0	41	44	67	45	4C											41	ADgELAQYAACIAAAA
000057F0	4D	41	41	41	41											41	MAAAAAAAAAAfkAAAAA
00005800	67	41	41	41	41	59	41	41	41	41	41	42	41	41	41	41	gAAAAAYAAAAABAAAA
00005810	67	41	41	41	41	41	67	41	41	42	41	41	41	41	41	41	qAAAAAqAABAAAAAA

```

>>>
>>> "TVqQ".decode("base64")
'MZ\x90'
>>> █

```

Inside the .NET assembly, the string is stored as a DefaultValue string. The developers might have used this as a way to conceal the meaning of this long string.

```

[DefaultValue("TVqQAAMAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnN...
c1IAbDh0v0ZKUTHGAG9qZvPUINuNQ8PRTdWtkRGUoArMxmSDBFbGp2AE5KaTNxcFdhaABVNjM2SU1zQmMAbmpXQ0FqVjNiAG80c1k1UDdMVGbFdmVudF...

```

Once the string is decoded, it is written to disk and executed.

Address	Stack	Procedure / arguments
0021E758	76E35186	KERNEL32.CreateProcessW
0021E75C	0428453C	ModuleFileName = "C:\Users\user\AppData\Roaming\Microsoft\Blend\14.0\FedCache\PerfWatson.exe"
0021E760	04D06990	CommandLine = ""C:\Users\user\AppData\Roaming\Microsoft\Blend\14.0\FedCache\PerfWatson.exe" "
0021E764	00000000	pProcessSecurity = NULL
0021E768	00000000	pThreadSecurity = NULL
0021E76C	00000000	InheritHandles = FALSE
0021E770	04080410	CreationFlags = CREATE_NEW_CONSOLE CREATE_UNICODE_ENVIRONMENT CREATE_DEFAULT_ERROR_MODE 80000
0021E774	00000000	pEnvironment = NULL
0021E778	04281188	CurrentDir = "C:\Users\user\Desktop"
0021E77C	0021E7C0	pStartupInfo = 0021E7C0
0021E780	04282888	pProcessInfo = 04282888
0021E850	76E42C34	? shell132.76E34FBC
0021FA04	76F34F65	shell132.76F42B75

Injecting the Payload

The NanoCore RAT payload is never written to disk to avoid detection. Instead, it is injected into a new process. The injection routine can be summarized by these Win32 API and system calls:

- **CreateProcessW(CREATE_SUSPEND):** create the child process in suspend mode.
- **NtGetContextThread():** Used to find the PEB and to update the EIP register.
- **ReadProcessMemory():** Reads the PEB.ImageBaseAddress field.
- **NtUnmapViewOfSection():** This runs only when there is an image already mapped to 0x400000.
- **VirtualAllocEx():** Used to allocate the pages for injection.
- **NtWriteVirtualMemory():**
- 0x00400000: MZ/PE Header
- 0x00402000: .text
- 0x00436000: .rsrc
- 0x0043a000: .reloc
- **PEB.ImageBaseAddress:** Updates the base address to 0x400000.
- **NtSetContextThread():** Updates the EIP register in the thread context.
- **NtAlertResumeThread():** Causes the child process to leave suspend mode and become runnable. The process begins in suspend mode:

Procedure
KERNEL32.CreateProcessW
ApplicationName = "C:\Users\user\Desktop\b7cfc7e9551b15319c068aae966f8a9ff563b522ed9b1b42d19c122778e018c8.exe"
CommandLine = ""C:\Users\user\Desktop\b7cfc7e9551b15319c068aae966f8a9ff563b522ed9b1b42d19c122778e018c8.exe""
pProcessSecurity = NULL
pThreadSecurity = NULL
InheritHandles = FALSE
CreationFlags = CREATE_SUSPENDED
pEnvironment = NULL
CurrentDirectory = NULL
pStartupInfo = 0043E058 -> STARTUPINFOW {Size=68., Reserved1=NULL, Desktop=NULL, Title=NULL, X=0, Y=0, Width=0, Height=0, ShowCmd=0, WindowStyle=0, WindowStyleEx=0, Reserved2=NULL}
pProcessInformation = 0025E428 -> PROCESS_INFORMATION {hProcess=NULL, hThread=NULL, ProcessID=0 (0.), ThreadID=0 (0.)}
mscnruks.60C31CFA

Next, the thread context is read from the child process:

CPU - main thread, module ntdll

```

77AB4E2C C2 1000 RETN 10
77AB4E2F 90 NOP
77AB4E30 B8 84000000 MOV EAX,84
77AB4E31 BA 0003FE7F MOV EDI,7FFE0300
77AB4E32 FF12 CALL DWORD PTR DS:[EDI]
77AB4E33 C2 0400 RETN 4
77AB4E34 90 NOP
77AB4E35 B8 85000000 MOV EAX,85
77AB4E36 BA 0003FE7F MOV EDI,7FFE0300
77AB4E37 FF12 CALL DWORD PTR DS:[EDI]
77AB4E38 C2 0800 RETN 8
77AB4E39 90 NOP
77AB4E3A B8 86000000 MOV EAX,86
77AB4E3B BA 0003FE7F MOV EDI,7FFE0300
77AB4E3C FF12 CALL DWORD PTR DS:[EDI]
77AB4E3D C2 2800 RETN 28
77AB4E3E 90 NOP
77AB4E3F B8 87000000 MOV EAX,87
77AB4E40 BA 0003FE7F MOV EDI,7FFE0300
77AB4E41 FF12 CALL DWORD PTR DS:[EDI]
77AB4E42 C2 0800 RETN 8
77AB4E43 90 NOP
77AB4E44 B8 88000000 MOV EAX,88
77AB4E45 BA 0003FE7F MOV EDI,7FFE0300
77AB4E46 FF12 CALL DWORD PTR DS:[EDI]
77AB4E47 C3 RETN
77AB4E48 8D49 00 LEA ECX,[ECX]
77AB4E49 B8 89000000 MOV EAX,89
77AB4E4A BA 0003FE7F MOV EDI,7FFE0300
77AB4E4B FF12 CALL DWORD PTR DS:[EDI]
77AB4E4C C2 0800 RETN 8
77AB4E4D 90 NOP
77AB4E4E B8 8A000000 MOV EAX,8A
77AB4E4F BA 0003FE7F MOV EDI,7FFE0300
77AB4E50 FF12 CALL DWORD PTR DS:[EDI]
77AB4E51 C2 0C00 RETN 0C
77AB4E52 90 NOP

```

Inn=0008
Top of stack [0025E380]=mscorwks.6DC31D39

ntdll.NtGetContextThread+0C

Address	Hex dump	ASCII	0025E380	6DC31D39	9#In	RETURN to mscorwks.6DC31D39
01CDD160	00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00	0	0025E384	00000264	dB	Arg1 = 264
01CDD170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E388	01CDD160	7F0	Arg2 = 1CDD160
01CDD180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E38C	003F3C50	P<?	
01CDD190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E390	0025E384	BTZ	
01CDD1A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E394	0015A890	EDZ	
01CDD1B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E398	0025E38C	ITZ	ASCII "P<?"
01CDD1C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E39C	77AB4E60	*N%w	ntdll.NtGetContextThread
01CDD1D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3A0	0015A880	CZ3	
01CDD1E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3A4	0025E3CC	FTZ	
01CDD1F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3A8	0015A87F	qzS	
01CDD200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3AC	003F3C50	P<?	RETURN from mscorwks.6DC31D39
01CDD210	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3B0	0025E3E4	ETZ	
01CDD220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3B4	AED87F14	7077	
01CDD230	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3B8	002E6B84	ak.	
01CDD240	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3BC	0043E040	@oC	
01CDD250	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3C0	00000000		
01CDD260	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3C4	01CDD158	N%w	
01CDD270	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3C8	00000264	dB	
01CDD280	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3CC	0025E46C	IZZ	
01CDD290	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3D0	002D4B7B	IK-	RETURN to 002D4B7B
01CDD2A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3D4	0025E428	IZZ	
01CDD2B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3D8	02A5B6A8	zj#0	
01CDD2C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3DC	01941198	y400	
01CDD2D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3E0	0025E46C	IZZ	
01CDD2E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3E4	6DC3A2A8	z6#m	
01CDD2F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3E8	0025E7E4	z7%	
01CDD300	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0025E3EC	77AB4E60	*N%w	ntdll.NtGetContextThread

From the thread context, the address of the PEB is now known and is can be read:

```

Procedure
[
KERNEL32.ReadProcessMemory
: hProcess = 00000268
: BaseAddress = 7FFDC008
: Buffer = 0025E418 -> 00
: Size = 4
: pBytesWritten = 0025E45C -> 0
: mscorwks ANP31CF0

```

The address range for the injected image is now allocated:

Procedure

KERNELBASE.VirtualAllocEx

```
Arg1 = 268
Arg2 = 400000
Arg3 = 30000
Arg4 = 3000
Arg5 = 40
mscorwks 60C31039
```

And now a series of NtWriteVirtualMemory() to inject the RAT image and update PEB.ImageBaseAddress.

Call stack of main thread

Stack	Data	Procedure	Called by
0025E368	60C31039	ntdll.NtWriteVirtualMemory	nscorwks
0025E36C	00000268	Arg1 = 268	
0025E370	00400000	Arg2 = 400000	
0025E374	0205E6B0	Arg3 = 205E6B0	
0025E378	00000200	Arg4 = 200	
0025E37C	0025E45C	Arg5 = 25E45C	
0025E39C	00154C33	nscorwks.60C31039	00154C2E
0025E3A4	00204777	777	00204770
0025E470	002052C2	00204818	002052B5

CPU - main thread, module mscorwks

Address	Hex dump	ASCII	Comment
60C31039	8045 F4	LER EAX, [EBP-0C]	
60C3103A	5B	PUSH EBX	
60C3103B	8045 F0	LER EAX, [EBP-10]	
60C3103C	5B	PUSH EBX	
60C3103D	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	
60C3103E	FF75 9C	PUSH DWORD PTR SS:[EBP+0C]	
60C3103F	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
60C31040	E8 34481D00	CALL 60E065B3	
60C31041	C9	TEST EAX, 00000100	
60C31042	74 80	JZ SHORT 60C31D33	
60C31043	A9 00020000	TEST EAX, 00000200	
60C31044	74 95	JZ SHORT 60C31D32	
60C31045	58	POP EAX	
60C31046	59	POP EBX	
60C31047	59	POP EBX	
60C31048	59	POP EBX	
60C31049	EB 01	JMP SHORT 60C31D33	
60C3104A	59	POP EBX	
60C3104B	> 8965 EC	MOV DWORD PTR SS:[EBP-14], ESP	
60C3104C	FF65 F8	CALL DWORD PTR SS:[EBP-8]	ntdll.NtWriteVirtualMemory
60C3104D	FF75 E8	PUSH DWORD PTR SS:[EBP-18]	
60C3104E	FF75 DC	PUSH DWORD PTR SS:[EBP-14]	
60C3104F	5B	PUSH EBX	
60C31050	5B	PUSH EBX	
60C31051	FF75 F4	PUSH DWORD PTR SS:[EBP-0C]	
60C31052	FF75 F0	PUSH DWORD PTR SS:[EBP-10]	
60C31053	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	
60C31054	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
60C31055	E8 804E1D00	CALL 60E06C10	
60C31056	C9	LEAVE	
60C31057	5B	POP EBX	
60C31058	59	POP EBX	
60C31059	59	POP EBX	
60C3105A	59	POP EBX	
60C3105B	8BEC	MOV EBP, ESP	
60C3105C	5D	POP ESP	
60C3105D	8045 FC	LER EAX, [EBP-4]	

Registers (MMX)

Register	Value
EAX	00000000
ECX	6DE06714 nscorwks.6DE06714
EDX	0205E6B0
EBX	00000000
ESP	00025360
EBP	00025398
ESI	00025306
EDI	000253E4
EIP	7765EE00 ntdll.NtWriteVirtualMemory
EFL	00000246 (NO, NB, E, BE, HS, PE, GE, L)
MM0	00000000 00000000 00000000 000
MM1	00000000 00000000 00000000 000
MM2	00000000 00000000 00000000 000
MM3	00000000 00000000 00000000 000
MM4	00000000 00000000 00000000 000
MM5	00000000 00000000 00000000 000
MM6	00000000 00000000 00000000 000
MM7	FA990000 00000000 00000000 000
XMM0	00000000 00000000 00000000 000
XMM1	9979443A 00093304 B1C0F94 9FF
XMM2	9632020A 0F940FF FFFFFFFE 200
XMM3	00FFFFFF F9363694 9B509900 FFF
XMM4	40FFFFFF F9363694 9B509900 600
XMM5	40FFFFFF F9363694 9B509900 600
XMM6	3497E8E8 9204FFF FFFF9633 0AE
XMM7	20FFFFFF F9363694 7FC928FF FFF
P U	
MCSCR	00001FA0 F2 0 0C 0 Exp: 1 0 Rnd: NEAR Task: 1 1

Stack (0025E390)=7765EE00 (ntdll.NtWriteVirtualMemory) (current registers)

Address	Hex dump	ASCII	Comment
0205E360	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E36C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E370	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E374	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E378	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E37C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E380	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E384	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E388	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E38C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E390	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E394	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E398	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E39C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3A4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3B4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3C4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0205E3D4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

NtSetContextThread is invoked to update the EIP register's value:

Call stack of main thread

Stack | Data | Procedure | Called From

002E5B94	60C31039	ntdll.NtSetContextThread	mscorwks.60C31036
002E5B98	00000004	Arg1 = 25E6A5A	
002E5B9C	01C00160	Arg2 = 1C0D160	
002E5BA0	0015A003	mscorwks.60C310EA	0015A00E 002022BC 002022BC
002E5BA4	00C0D160	77	mscorwks.60C31B69
002E5BA8	002022C2	00204A18	
002E5BAC	77	77	

CPU - main thread, module ntdll

Registers (FPU)

EAX	00000000
EAX	60E06714 mscorwks.60E06714
EAX	00000000
EAX	003F3C50
EAX	002E5B94
EAX	002E5B94
EAX	002E5B94
EAX	002E5B94
EAX	002E5B94
EIP	77B59600 ntdll.NtSetContextThre
C 0	ES 0023 32bit 0 (FFFFFFFF)
C 1	CS 001B 32bit 0 (FFFFFFFF)
C 2	DS 0023 32bit 0 (FFFFFFFF)
C 3	FS 0023 32bit 77FDE000(FFF)
C 4	GS 0000 NULL
EIP	00000000
EFL	00000246 (NO,HS,E,SE,NS,PE,GE,L
MI0	0000 0000 0000 0000
MI1	0000 0000 0000 0000
MI2	0000 0000 0000 0000
MI3	0000 0000 0000 0000
MI4	0000 0000 0000 0000
MI5	0000 0000 0000 0000
MI6	0000 0000 0000 0000
MI7	F029 0000 0000 0000

Finally, execution is started with NtAlertResumeThread:

Call stack of main thread

Stack | Data | Procedure | Called From

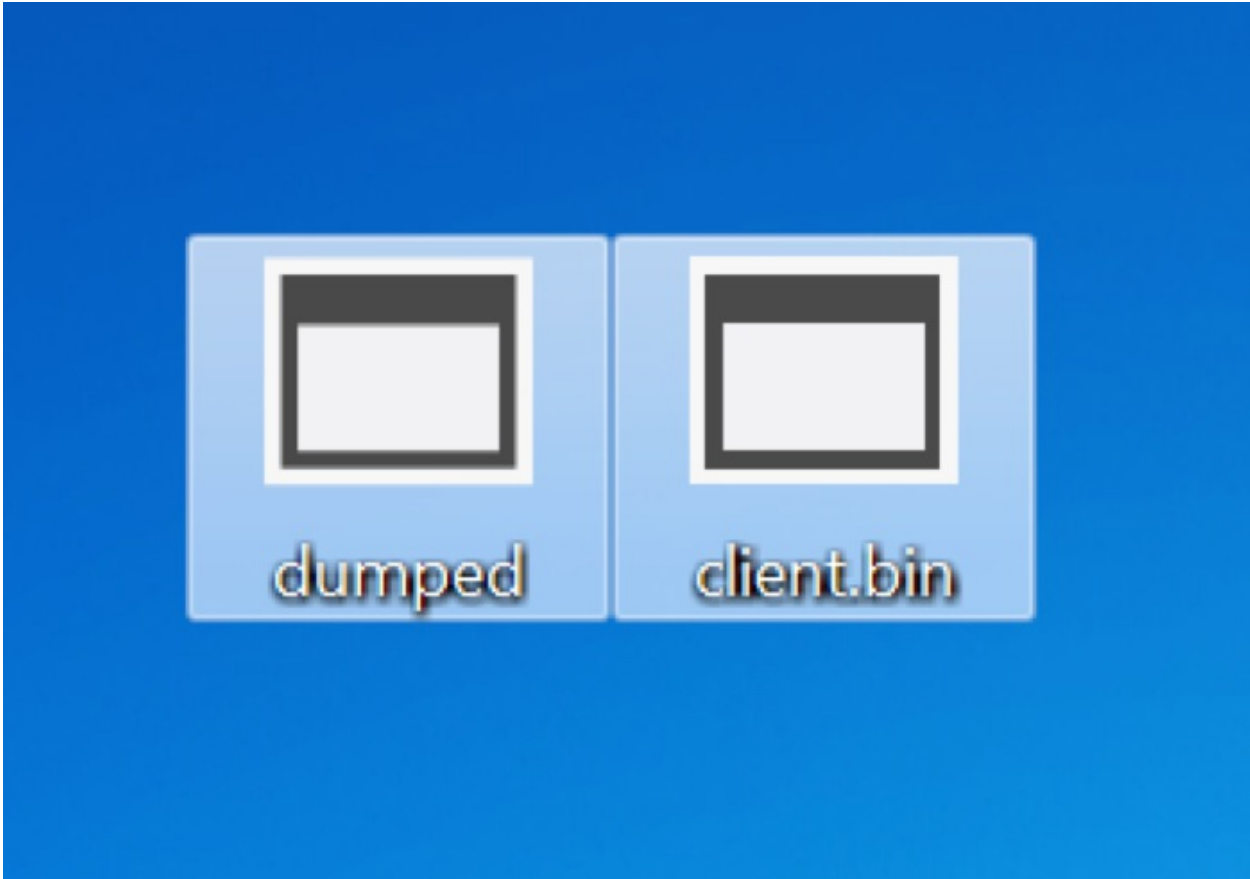
002E5B94	60C31039	ntdll.NtAlertResumeThread	mscorwks.60C31039
002E5B98	0015A77C	mscorwks.60C310EA	0015A77E 002022BC 002022BC
002E5BA0	00C0D160	77	mscorwks.60C31B69
002E5BA4	002022C2	00204A18	
002E5BA8	77	77	

CPU - main thread, module mscorwks

Registers (FPU)

EAX	00000000
EAX	60E06714 mscorwks.60E06714
EAX	00000000
EAX	003F3C50
EAX	002E5B94
EAX	002E5B94
EAX	002E5B94
EAX	002E5B94
EIP	77B59600 ntdll.NtAlertResumeThr
C 0	ES 0023 32bit 0 (FFFFFFFF)
C 1	CS 001B 32bit 0 (FFFFFFFF)
C 2	DS 0023 32bit 0 (FFFFFFFF)
C 3	FS 0023 32bit 77FDE000(FFF)
C 4	GS 0000 NULL
EIP	00000000
EFL	00000246 (NO,HS,E,SE,NS,PE,GE,L
MI0	0000 0000 0000 0000
MI1	0000 0000 0000 0000
MI2	0000 0000 0000 0000
MI3	0000 0000 0000 0000
MI4	0000 0000 0000 0000
MI5	0000 0000 0000 0000
MI6	0000 0000 0000 0000
MI7	F029 0000 0000 0000

By dumping the process to disk, we can see that the injected process is just the NanoCore client.



For more information on how SentinelOne protects against attacks such as these, visit our resources page at sentinelone.com/resources

