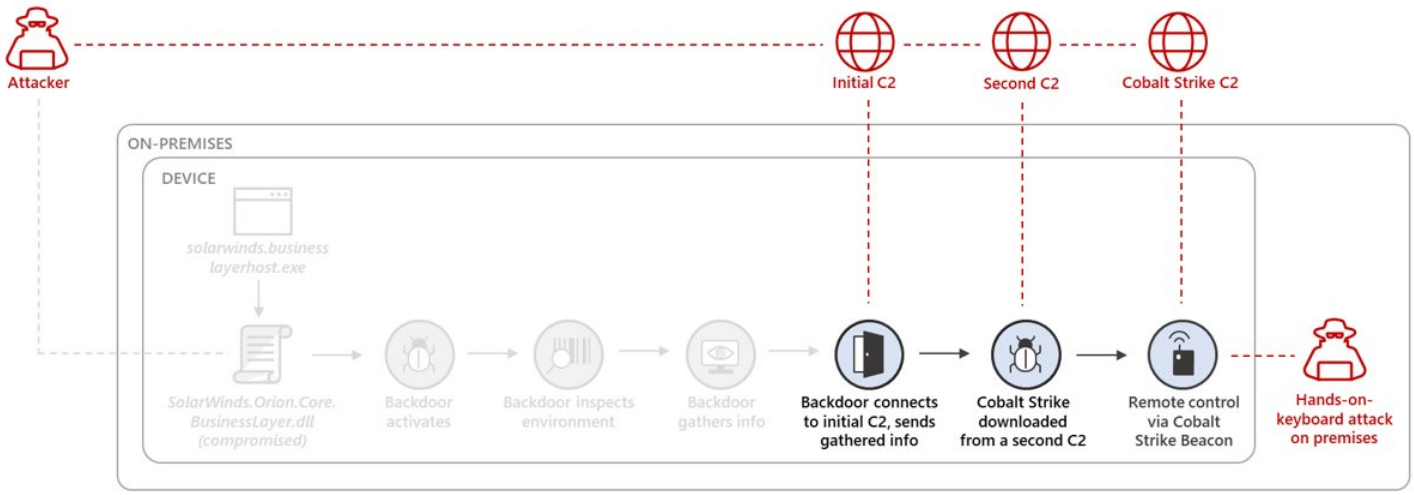
Twitter Thread by Ramin

Ramin
@MalwareRE

As part of our commitment to keeping our customers/community protected & informed, we are releasing a blog that shines light on transition between Stage 1 and 2 of #Solorigate/#SUNBURST campaign, custom Cobalt Strike loaders, post-exploit. artifacts, IOCs: https://t.co/b0ReHMa63u

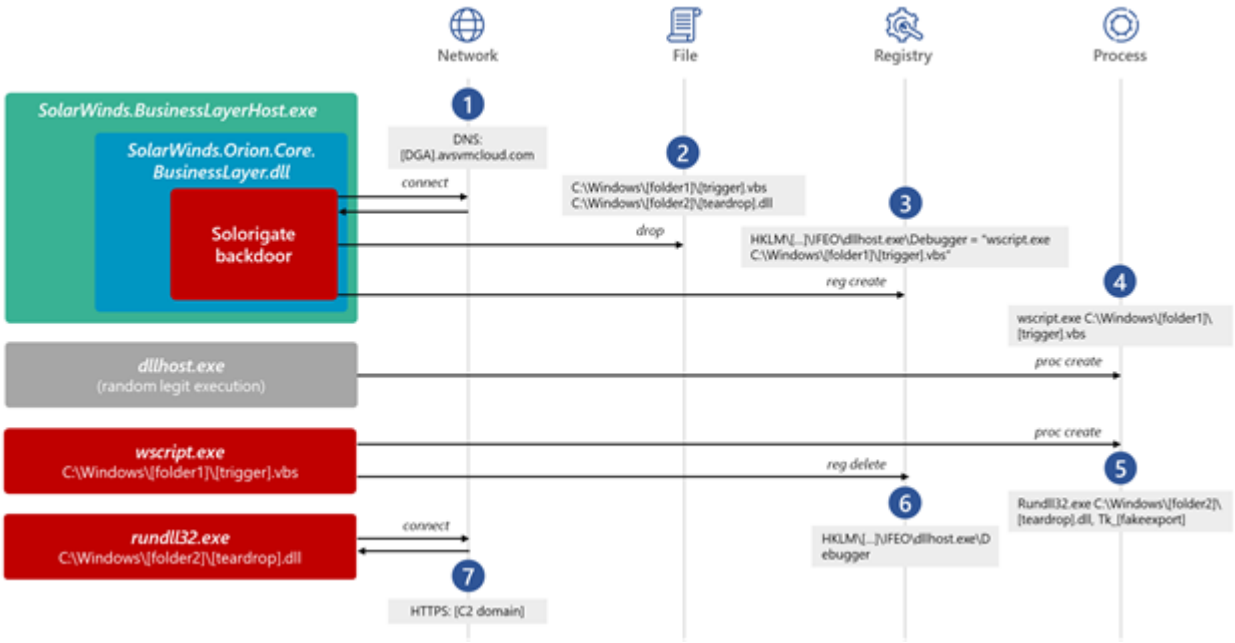


Figure·2.·Diagram·of·transition·between·Stage·1·and·Stage·2·of·the·Solorigate·attack¶

Here are some highlights:

The missing link between the Solorigate backdoor and the custom #CobaltStrike loaders observed during the #Solorigate is an Image File Execution Options (IFEO) Debugger registry value created for the legitimate process dllhost.exe (ATT&CK ID: T1546.012).

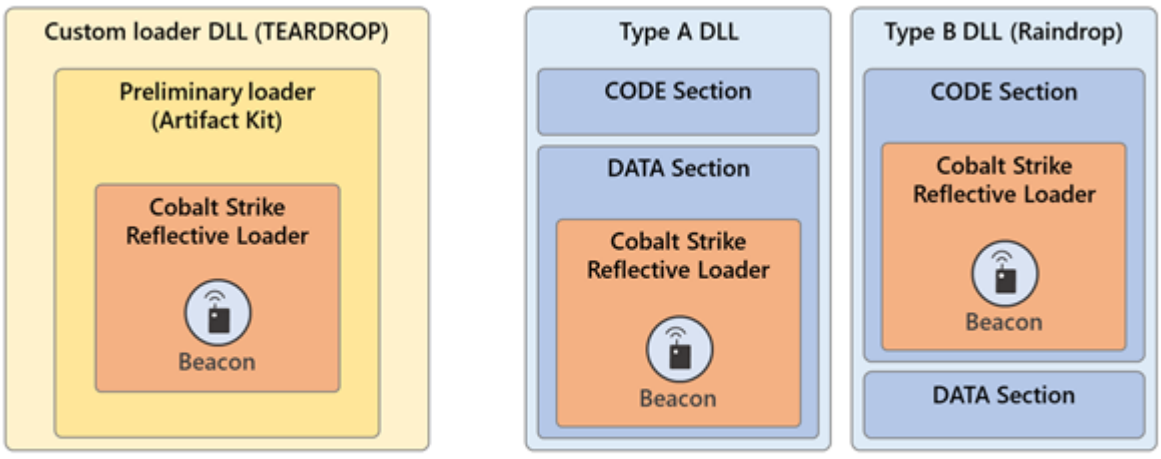
Once the registry value is created, the attackers wait for the occasional execution of dllhost.exe, which might happen naturally on a system. This execution triggers a process launch of wscript.exe configured to run the VBScript file dropped by the SolarWinds backdoor (Stage 1).



The VBScript in turn runs rundll32.exe, activating the Cobalt Strike loader DLL 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.

On the custom Cobalt Strike Loaders: 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.

#TEARDROP, #Raindrop, and the other custom Cobalt Strike Beacon loaders observed 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.

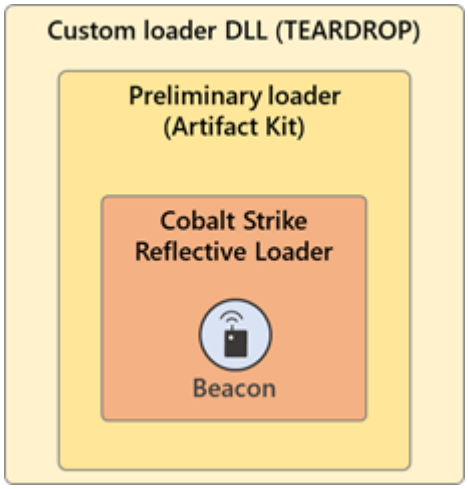


Variant 1: TEARDROP

Variant 2: Other custom Cobalt Strike Beacon loaders

The TEARDROP variants have an export that contains the trigger for the malicious code (executed in a new thread created by the export). The malicious code attempts to open a .jpg file (festive_computer.jpg, upbeat_anxiety.jpg, gracious_truth.jpg, confident_promotion.jpg, etc.).

Next, TEARDROP proceeds to decode & subsequently execute an embedded custom preliminary loader (likely generated using a Cobalt Strike Artifact Kit template e.g., bypass-pipe.c). In its true form, the preliminary loader is a DLL that has been transformed & loaded like shellcode.



We came across additional custom loaders for Cobalt Strike's Beacon that 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.

Variant 2 custom loaders also contain an attacker-introduced export (using varying names) whose only purpose is to call the Sleep() function every minute.

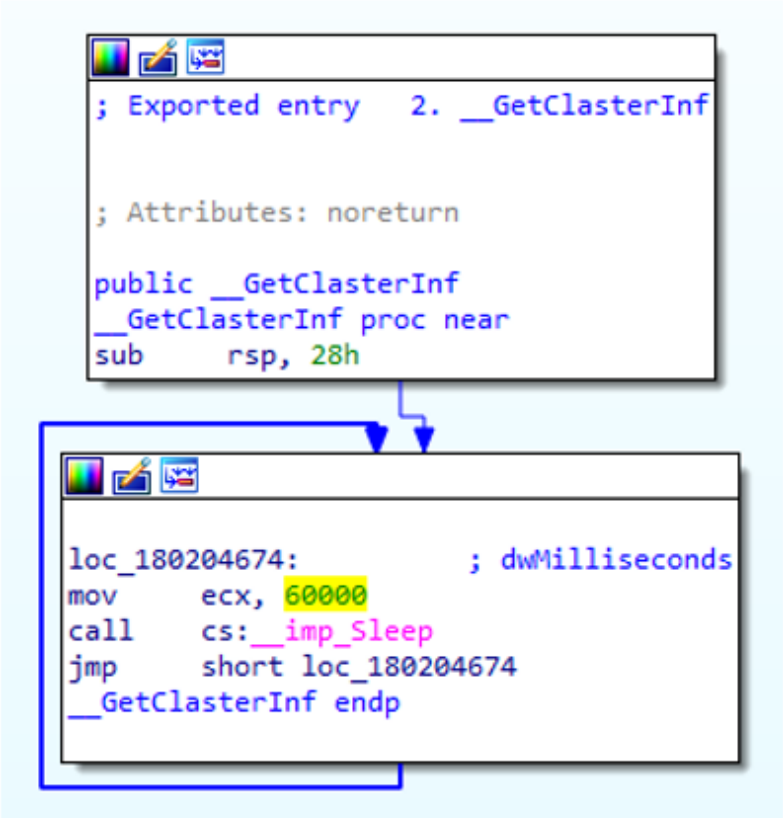


Figure 7. Example of a custom export function from a Variant 2 loader

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.

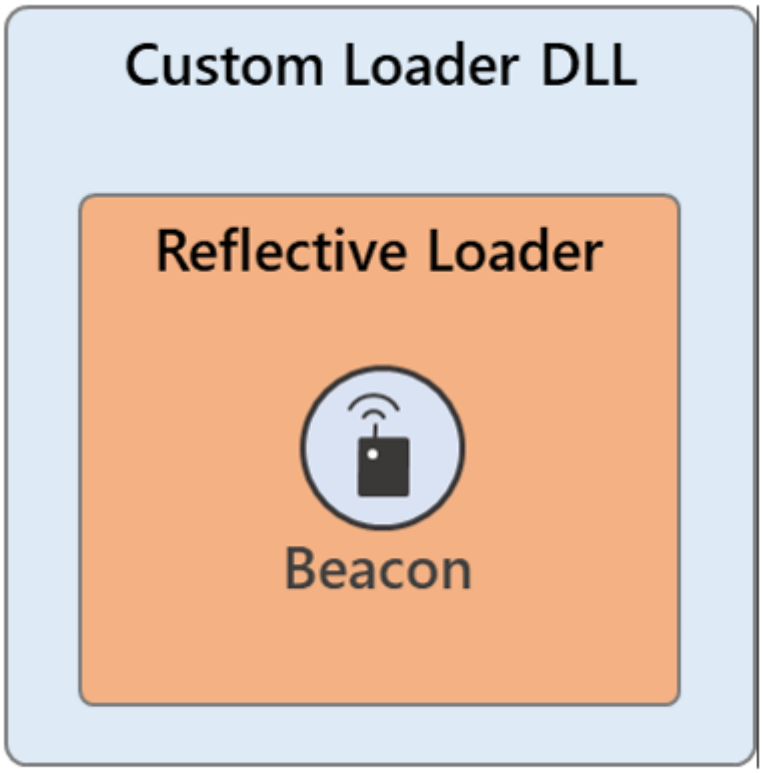


Figure 8. Structure of Variant 2 custom Loader

These custom loaders can be divided into two types:
Type A: Decodes/Loads CS's RL from the DLL's DATA section (detected as Trojan:Win64/Solorigate.SC!dha)
Type B: De-obfuscates/Loads RL from the DLL's CODE section (aka #Raindrop, detected as Trojan:Win64/Solorigate.SB!dha).

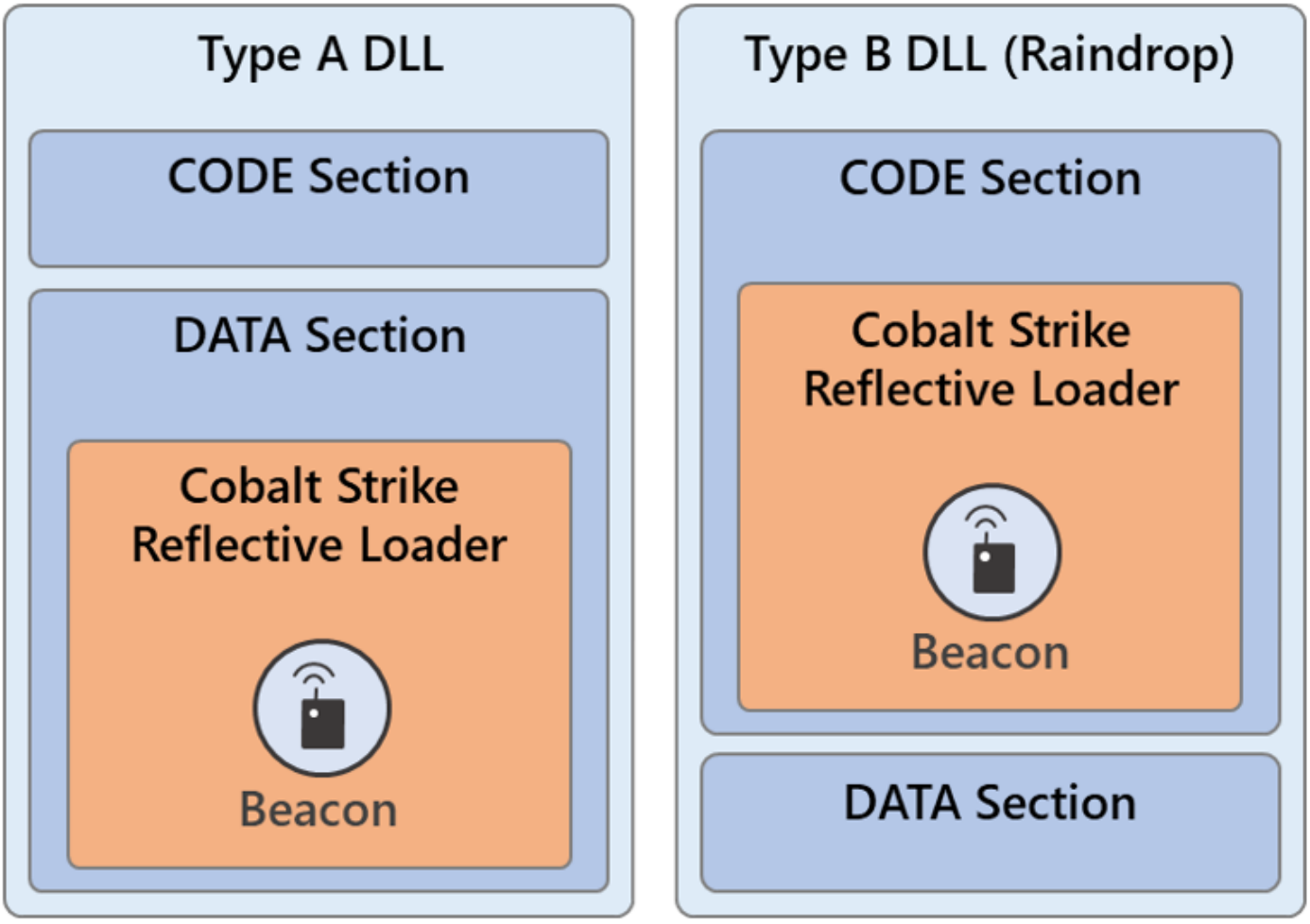


Figure 9. Two subtypes of the custom Loader

Some observations:
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 2nd stage custom loader (TEARDROP) was introduced to the environment by SolarWinds.BusinessLayerHost.exe at ~ 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. (synthetic compile timestamps via custom Malleable C2 profiles?)

2020? The actor did not timestamp the compile time of the custom loader DLLs? 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...

Most custom loader DLLs were configured with PE version information that masquerades version information belonging to legitimate applications and files from Windows (e.g., NETSETUPSVC.DLL), 7-Zip (e.g., 7z.dll), Far Manager (e.g., Far.dll), LibIntl (e.g., libintl3.dll), etc.

Certain development artifacts were left behind in the custom loader samples. e.g. the following C++ header (.hpp) path was observed in a loader compiled from a modified Far Manager source code: c:\build\workspace\cobalt_cryptor_far

(dev071)\farmanager\far\platform.concurrency.hpp

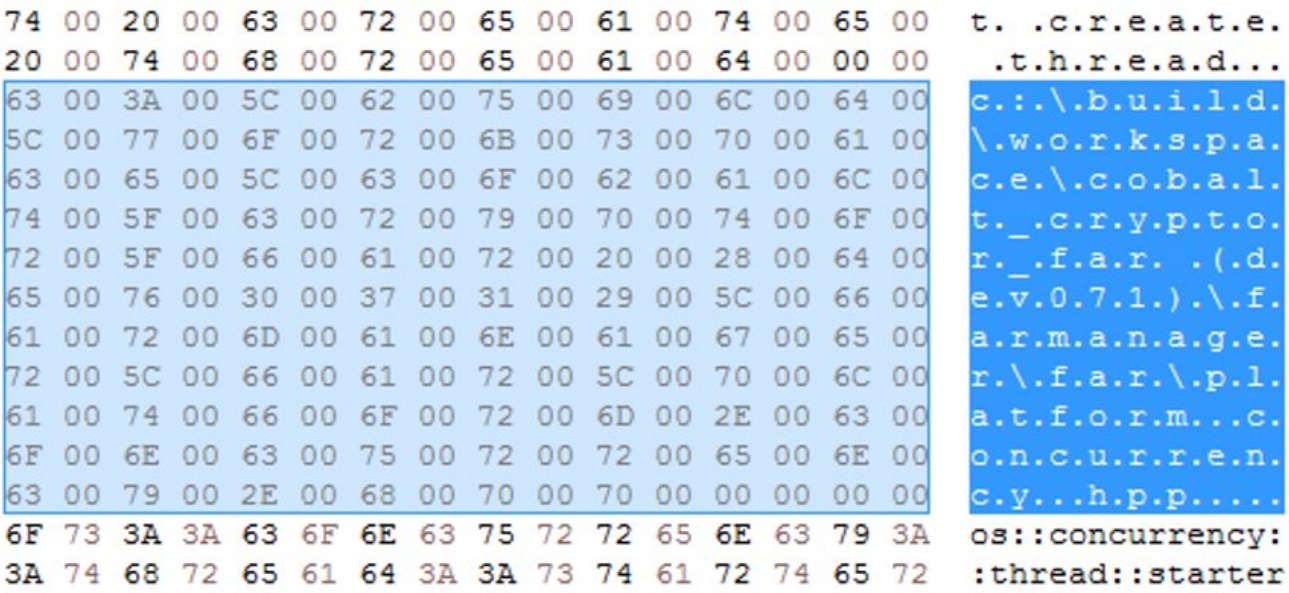


Figure 10. File path for a C++ header file (.hpp) observed in custom Cobalt Strike loader samples

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, User-Agent , HTTP POST/GET transaction URI, sleep time & jitter factor

Each Beacon instance carries a unique Watermark value. Analysis of the Watermark values revealed that all Watermark values start with the number ‘3’.

0x30343131	0x34353633	0x38303535	0x38383238
0x32323638	0x35373331	0x38353138	0x38383430

The post-exploitation related artifacts, TTPs and MITRE ATT&CK techniques (an extensive list) are best covered/described under the "Additional attacker tactics, anti-forensic behavior, and operational security" section of the blog: https://t.co/b0ReHMrGV2

Leaving No Stone Unturned: This blog is a collaboration between multiple security, threat intelligence, product, forensic, SOC, Identity & legal teams from across Microsoft. For more information refer to our dedicated Solorigate Resource Center: https://t.co/8Swnphedko.