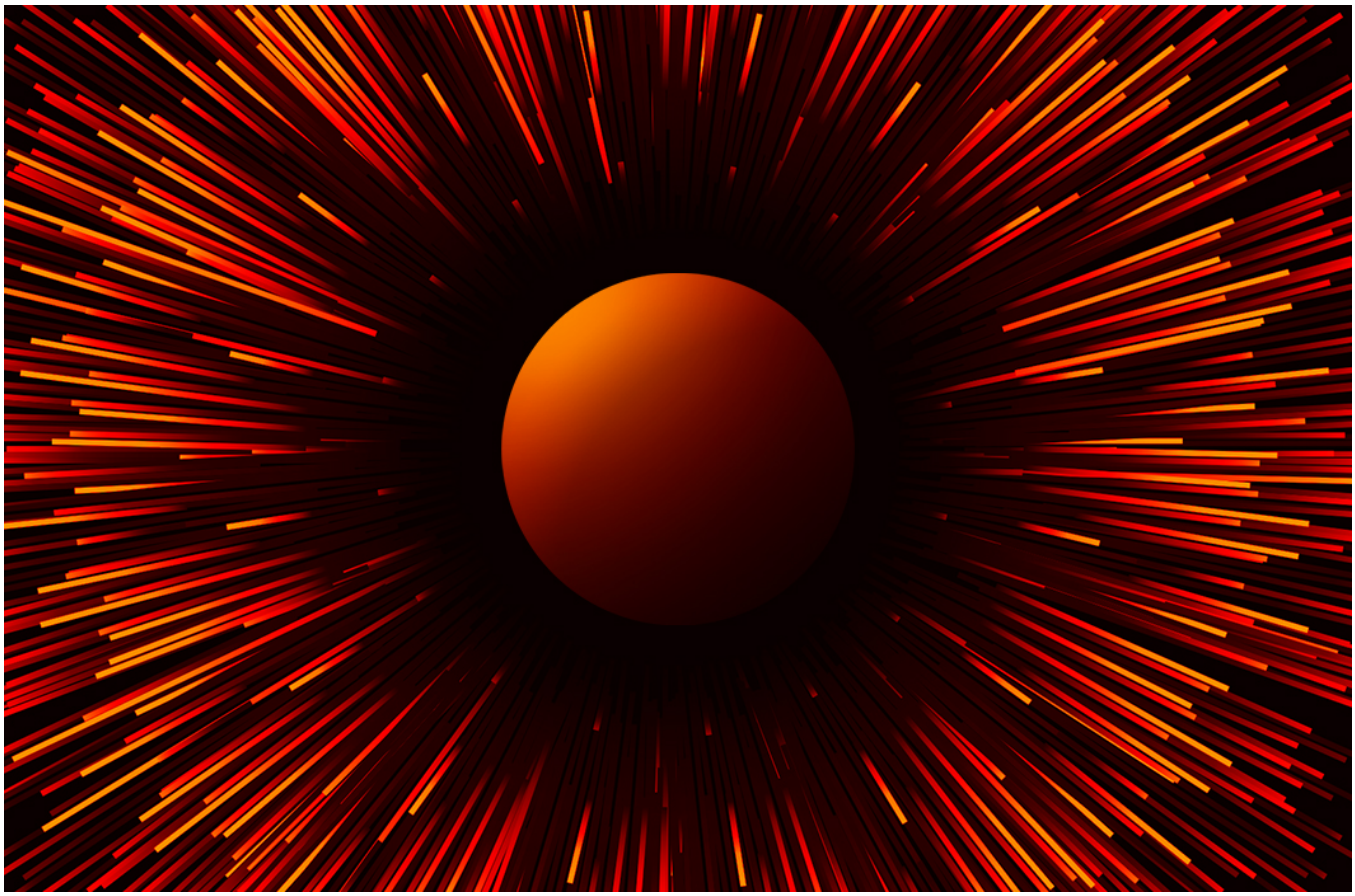
Technical Analysis

crowdstrike.com/blog/sunspot-malware-technical-analysis

CrowdStrike Intelligence Team January 11, 2021



In December 2020, the industry was rocked by the disclosure of a complex supply chain attack against SolarWinds, Inc., a leading provider of network performance monitoring tools used by organizations of all sizes across the globe. CrowdStrike and another firm have been supporting SolarWinds in its investigation and root cause analysis of the events that led to the inclusion of unauthorized malicious code into its build cycle. In coordination with SolarWinds, which has posted a blog detailing its efforts to ensure the security of its customers and build processes, CrowdStrike is providing a technical analysis of a malicious tool that was deployed into the build environment to inject this backdoor into the SolarWinds Orion platform without arousing the suspicion of the development team charged with delivering the product. At this time, CrowdStrike does not attribute the SUNSPOT implant, SUNBURST backdoor or TEARDROP post-exploitation tool to any known adversary; as such, CrowdStrike Intelligence is tracking this intrusion under the StellarParticle activity cluster.

Key Points

- SUNSPOT is StellarParticle's malware used to insert the SUNBURST backdoor into software builds of the SolarWinds Orion IT management product.
- SUNSPOT monitors running processes for those involved in compilation of the Orion product and replaces one of the source files to include the SUNBURST backdoor code.
- Several safeguards were added to SUNSPOT to avoid the Orion builds from failing, potentially alerting developers to the adversary's presence.

Analysis of a SolarWinds software build server provided insights into how the process was hijacked by StellarParticle in order to insert SUNBURST into the update packages. The design of SUNSPOT suggests StellarParticle developers invested a lot of effort to ensure the code was properly inserted and remained undetected, and prioritized operational security to avoid revealing their presence in the build environment to SolarWinds developers.

SUNSPOT was identified on disk with a filename of `taskhostsvc.exe` (SHA256 Hash: `c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230af0b3ef168`), and internally named `taskhostw.exe` by its developers. It was likely built on 2020-02-20 11:40:02, according to the build timestamp found in the binary, which is consistent with the currently assessed StellarParticle supply chain attack timeline. StellarParticle operators maintained the persistence of SUNSPOT by creating a scheduled task set to execute when the host boots.

Initialization and Logging

When SUNSPOT executes, it creates a mutex named `{12d61a41-4b74-7610-a4d8-3028d2f56395}` to ensure only one instance is running. It then creates an encrypted log file at `C:\Windows\Temp\vmware-vmdmp.log`. Individual log entries are encrypted with the stream cipher RC4, using the hard-coded key `FC F3 2A 83 E5 F6 D0 24 A6 BF CE 88 30 C2 48 E7`. Throughout execution, SUNSPOT will log errors to this file, along with other deployment information. Log entries are delineated by the hex string `32 78 A5 E7 1A 79 91 AC` and begin with the number of seconds elapsed since the first log line. Most log lines corresponding to an error contain a step number (e.g., *Step19*) requiring knowledge of the malware to understand their meaning. These steps and their mapping to the malware actions are provided at the end of this blog. The step numbering does not follow the actual execution order, suggesting the calls to the logging function were added by the developers during the creation of the malware as they progressed and needed to focus their efforts on debugging one part of the code. An extract of a log file generated by SUNSPOT in a test environment is given below.

```
0.000 START
22.781[3148] + 'msbuild.exe' [6252]

194.343[13760] + 'msbuild.exe' [6252]

324.250[14696] + 'msbuild.exe' [6252]
351.125[14696] - 0
352.031[14176] + 'msbuild.exe' [6252]

376.343[11864] + 'msbuild.exe' [6252]

439.953[9204] + 'msbuild.exe' [6252]
485.343[9204] Solution directory: C:\Users\User\Source
485.343[ERROR]
Step4('C:\Users\User\Source\Src\Lib\SolarWinds.Orion.Core.BusinessLayer\BackgroundInventory\InventoryManager.cs')
fails
```

The malware then grants itself debugging privileges by modifying its security token to add `SeDebugPrivilege`. This step is a prerequisite for the remainder of SUNSPOT's execution, which involves reading other processes' memory.

Build Hijacking Steps

Monitoring of Running Software Build Processes

After initialization, SUNSPOT monitors running processes for instances of `MsBuild.exe`, which is part of Microsoft Visual Studio development tools. Copies of `MsBuild.exe` are identified by hashing the name of each running process and comparing it to the corresponding value, `0x53D525`. The hashing algorithm used for the comparison is ElfHash and is provided in Python in Figure 1.

```
def elf_hash(name):
    # Test input: b'msbuild.exe'
    # Test output: 0x53D525
    h = 0
    for c in name:
        v = (c + (h << 4))
        msb = v & 0xF0000000
       if msb != 0:
            v ^= (msb >> 24)
        h = ~msb & v
    return h
```

Figure 1. Process Name Hashing Logic

When SUNSPOT finds an `MsBuild.exe` process, it will spawn a new thread to determine if the Orion software is being built and, if so, hijack the build operation to inject SUNBURST. The monitoring loop executes every second, allowing SUNSPOT to modify the target source code before it has been read by the compiler.

Although the mutex created during the initialization should already prevent multiple process monitoring loops from running, the malware checks for the presence of a second mutex — `{56331e4d-76a3-0390-a7ee-567adf5836b7}`. If this mutex is found, the backdoor interprets it as a signal to quit, waits for the completion of its currently running backdoor injection threads, and exits. This mutex was likely intended to be used by StellarParticle operators to discreetly stop the malware, instead of using a riskier method such as killing the process. Stopping SUNSPOT in the middle of its operation could result in unfinished tampering of the Orion source code, and lead to Orion build errors that SolarWinds developers would investigate, revealing the adversary's presence.

Command-Line Arguments Extraction from Process Memory

The malware extracts the command-line arguments for each running `MsBuild.exe` process from the virtual memory using a methodology similar to one publicly documented[1].

A call to `NtQueryInformationProcess` allows the adversary to obtain a pointer to the remote process's Process Environment Block (PEB), which contains a pointer to a `_RTL_USER_PROCESS_PARAMETERS` structure. The latter is read to get the full command line passed to the `MsBuild.exe` process.

The command line is then parsed to extract individual arguments, and SUNSPOT looks for the directory path of the Orion software Visual Studio solution. This value is hard-coded in the binary, in an encrypted form using AES128-CBC, whose parameters are given below. The same material is used for all of the blobs encrypted with AES in the binary.

```
key = FC F3 2A 83 E5 F6 D0 24 A6 BF CE 88 30 C2 48 E7 (same as the RC4 key)
iv  = 81 8C 85 49 B9 00 06 78 0B E9 63 60 26 64 B2 DA
```

The key and initialization vector (IV) are not unique and can be independently found in other binary samples of several popular video games. It is plausible the material was chosen on purpose in order to make static detections on the values impractical.

Orion Source Code Replacement

When SUNSPOT finds the Orion solution file path in a running `MsBuild.exe` process, it replaces a source code file in the solution directory, with a malicious variant to inject SUNBURST while Orion is being built. While SUNSPOT supports replacing multiple files, the identified copy only replaces `InventoryManager.cs` .

The malicious source code for SUNBURST, along with target file paths, are stored in AES128-CBC encrypted blobs and are protected using the same key and initialization vector.

As causing build errors would very likely prompt troubleshooting actions from the Orion developers and lead to the adversary's discovery, the SUNSPOT developers included a hash verification check, likely to ensure the injected backdoored code is compatible with a known source file, and also avoid replacing the file with garbage data from a failed decryption. In the exemplar SUNSPOT sample, the MD5 hash for the backdoored source code is `5f40b59ee2a9ac94ddb6ab9e3bd776ca` .

If the decryption of the parameters (target file path and replacement source code) is successful and if the MD5 checks pass, SUNSPOT proceeds with the replacement of the source file content. The original source file is copied with a `.bk` extension (e.g., `InventoryManager.bk`), to back up the original content. The backdoored source is written to the same filename, but with a `.tmp` extension (e.g., `InventoryManager.tmp`), before being moved using `MoveFileEx` to the original filename (`InventoryManager.cs`). After these steps, the source file backdoored with SUNBURST will then be compiled as part of the standard process.

SUNSPOT appends an entry in the log file with the date and time of the backdoor attempt and waits for the `MsBuild.exe` process to exit before restoring the original source code and deleting the temporary `InventoryManager.bk` file. If the Orion solution build is successful, it is backdoored with SUNBURST.

SUNBURST Source Code

The source code of SUNBURST was likely sanitized before being included in SUNSPOT. The use of generic variable names, pre-obfuscated strings, and the lack of developer comments or disabled code is similar to what could be obtained after decompiling a backdoored Orion binary, as illustrated in Figure 2, which provides a comparison between the injected source code (top) and a decompilation output (bottom).

```
private static class ProcessTracker {
   private static readonly object _lock = new object();
   private static bool SearchConfigurations()
   {
       using (ManagementObjectSearcher managementObjectSearcher = new
ManagementObjectSearcher(ZipHelper.Unzip("C07NSU0uUdBScCvKz1UIz8wzNooPriwuSc11KcosSy0CAA==")))
       {
           foreach (ManagementObject item in managementObjectSearcher.Get())
           {
               ulong hash =
GetHash(Path.GetFileName(item.Properties[ZipHelper.Unzip("C0gsyfBLzE0FAA==")].Value.ToString()).ToLower());
               if (Array.IndexOf(configTimeStamps, hash) != -1)
               {
                   return true;
               }
```

```
            }
        }
        return false;
    }
private static class ProcessTracker {
    // Token: 0x0600097C RID: 2428 RVA: 0x000435A4 File Offset: 0x000417A4
    private static bool SearchConfigurations()
    {
        using (ManagementObjectSearcher managementObjectSearcher = new
ManagementObjectSearcher(OrionImprovementBusinessLayer.ZipHelper.Unzip("C07NSU0uUdBScCvKz1UIz8wzNooPriwuSc11KcosSy0CAA==
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ulong hash =
OrionImprovementBusinessLayer.GetHash(Path.GetFileName(((ManagementObject)managementBaseObject).Properties[OrionImprovem
                if (Array.IndexOf<ulong>(OrionImprovementBusinessLayer.configTimeStamps, hash) != -1)
                {
                    return true;
                }
            }
        }
        return false;
    }
```

Figure 2. Comparison between injected source code (top) and decompiled using DnSpy (bottom)

In order to remove compilation warnings that could be generated by the adversary's own code — which could alert the SolarWinds developers — StellarParticle made their edits within `#pragma warning disable` and `#pragma warning` restore statements, hinting at what parts were edited. In particular, SUNSPOT's entry point was added to the legitimate Orion software `RefreshInternal` function by adding the following try/catch block:

```
try {   if (!OrionImprovementBusinessLayer.IsAlive) {
    Thread th = new Thread(OrionImprovementBusinessLayer.Initialize)
      { IsBackground = true };
    th.Start();
  }
} catch (Exception) {
}
```

Tactics, Techniques and Procedures (TTPs)

The following TTPs may be used to characterize the SUNSPOT activity described in this blog:

- Persistence using scheduled tasks, triggered at boot time
- Use of AES128-CBC to protect the targeted source code files and the backdoored source code file in the binary
- Use of RC4 encryption with a hard-coded key to protect the log file entries
- Log entries from different executions of the malware that are separated with a hard-coded value `32 78 A5 E7 1A 79 91 AC`
- Log file creation in the system temp directory `C:\Windows\Temp\vmware-vmdmp.log` masquerading as a legitimate VMWare log file

- Detection of the targeted Visual Studio solution build by reading the virtual memory of `MsBuild.exe` processes, looking for the targeted solution filename
- Access to the remote process arguments made via the remote process's PEB structure
- Replacement of source code files during the build process, before compilation, by replacing file content with another version containing SUNBURST
- Insertion of the backdoor code within `#pragma` statements disabling and restoring warnings, to prevent the backdoor code lines from appearing in build logs
- Check of the MD5 hashes of the original source code and of the backdoored source code to ensure the tampering will not cause build errors
- Attempt to open a non-existing mutex to detect when the malware operators want the backdoor to stop execution and safely exit

Host Indicators of Attack

The tables below detail files belonging to the SUNSPOT campaigns including filename, SHA256 hash, and build time when known.

Executables

Filename	SHA256 Hash	Build Time (UTC)
taskh `ostsvc.exe`	`c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230af0b3ef168`	`2` 020-02-20 11:40:02

Related Files

Description	SHA256 Hash
`Backdoored Orion source code with SUNSPOT`	`0819db19be479122c1d48743e644070a8dc9a1c852df9a8c0dc2343e904da389`

File System

The presence of one or more of the following files may indicate a SUNSPOT infection.

File Path	Description
`C:\Windows\Temp\vmware-vmdmp.log`	Encrypted log file

Volatile Artifacts

Name	Type	Description
`{12d61a41-4b74-7610-a4d8-3028d2f56395}`	Mutex	Ensures a single implant instance
`{56331e4d-76a3-0390-a7ee-567adf5836b7}`	Mutex	Used to signal to the malware to safely exit

YARA Rules

```
rule CrowdStrike_SUNSPOT_01 : artifact stellarparticle sunspot {

    meta:
       copyright = "(c) 2021 CrowdStrike Inc."
       description = "Detects RC4 and AES key encryption material in SUNSPOT"

        version = "202101081448"
       last_modified = "2021-01-08"
       actor = "StellarParticle"
       malware_family = "SUNSPOT"

    strings:

        $key = {fc f3 2a 83 e5 f6 d0 24 a6 bf ce 88 30 c2 48 e7}
       $iv  = {81 8c 85 49 b9 00 06 78 0b e9 63 60 26 64 b2 da}

    condition:
       all of them and filesize < 32MB

}

rule CrowdStrike_SUNSPOT_02 : artifact stellarparticle sunspot
{
```

```
    meta:
        copyright = "(c) 2021 CrowdStrike Inc."
        description = "Detects mutex names in SUNSPOT"
        version = "202101081448"
        last_modified = "2021-01-08"
        actor = "StellarParticle"
        malware_family = "SUNSPOT"

    strings:
        $mutex_01 = "{12d61a41-4b74-7610-a4d8-3028d2f56395}" wide ascii
        $mutex_02 = "{56331e4d-76a3-0390-a7ee-567adf5836b7}" wide ascii

    condition:
        any of them and filesize < 10MB

}

rule CrowdStrike_SUNSPOT_03 : artifact logging stellarparticle sunspot

{

    meta:
        copyright = "(c) 2021 CrowdStrike Inc."
        description = "Detects log format lines in SUNSPOT"
        version = "202101081443"
        last_modified = "2021-01-08"
        actor = "StellarParticle"
        malware_family = "SUNSPOT"

    strings:
        $s01 = "[ERROR] ***Step1('%ls','%ls') fails with error %#x***\x0A" ascii
        $s02 = "[ERROR] Step2 fails\x0A" ascii
        $s03 = "[ERROR] Step3 fails\x0A" ascii
        $s04 = "[ERROR] Step4('%ls') fails\x0A" ascii
        $s05 = "[ERROR] Step5('%ls') fails\x0A" ascii
        $s06 = "[ERROR] Step6('%ls') fails\x0A" ascii
        $s07 = "[ERROR] Step7 fails\x0A" ascii
        $s08 = "[ERROR] Step8 fails\x0A" ascii
        $s09 = "[ERROR] Step9('%ls') fails\x0A" ascii
        $s10 = "[ERROR] Step10('%ls','%ls') fails with error %#x\x0A" ascii
        $s11 = "[ERROR] Step11('%ls') fails\x0A" ascii
        $s12 = "[ERROR] Step12('%ls','%ls') fails with error %#x\x0A" ascii
        $s13 = "[ERROR] Step30 fails\x0A" ascii
        $s14 = "[ERROR] Step14 fails with error %#x\x0A" ascii
        $s15 = "[ERROR] Step15 fails\x0A" ascii
        $s16 = "[ERROR] Step16 fails\x0A" ascii
        $s17 = "[%d] Step17 fails with error %#x\x0A" ascii
        $s18 = "[%d] Step18 fails with error %#x\x0A" ascii
        $s19 = "[ERROR] Step19 fails with error %#x\x0A" ascii
        $s20 = "[ERROR] Step20 fails\x0A" ascii
        $s21 = "[ERROR] Step21(%d,%s,%d) fails\x0A" ascii
        $s22 = "[ERROR] Step22 fails with error %#x\x0A" ascii
        $s23 = "[ERROR] Step23 fails with error %#x\x0A" ascii
        $s24 = "[%d] Solution directory: %ls\x0A" ascii
        $s25 = "[%d] %04d-%02d-%02d %02d:%02d:%02d:%03d %ls\x0A" ascii
        $s26 = "[%d] + '%s' " ascii

    condition:
        2 of them and filesize < 10MB
}
```

ATT&CK Framework

The following table maps reported SUNSPOT TTPs to the MITRE ATT&CK® framework.

Tactic	Technique	Observable
Reconnaissance	**T1592.002** Gather Victim Host Information – Software	StellarParticle had an understanding of the Orion build chain before SUNSPOT was developed to tamper with it.

Resource Development	**T1587.001** Develop Capabilities – Malware	SUNSPOT was weaponized to specifically target the Orion build to replace one source code file and include the SUNBURST backdoor.
Persistence	**T1053.005** Scheduled Task	SUNSPOT is persisted in a scheduled task set to execute after the host has booted.
Defense Evasion	**T1140** Deobfuscate/Decode Information	The configuration in SUNSPOT is encrypted using AES128-CBC. It contains the replacement source code, the targeted Visual Studio solution file name, and targeted source code file paths relative to the solution directory.
	T1027 Obfuscated Files or Information	The log file SUNSPOT writes is encrypted using RC4.
	T1480 Execution Guardrails	The replacement of source code is done only if the MD5 checksums of both the original source code file and backdoored replacement source code match hardcoded values.
	T1036 Masquerading	SUNSPOT masquerades as a legitimate Windows Binary, and writes its logs in a fake VMWare log file.
Discovery	**T1057** Process Discovery	SUNSPOT monitors running processes looking for instances of MsBuild.exe.
Impact	**T1565.001** Data Manipulation Stored – Data Manipulation	Modification of the Orion source code to inject SUNBURST.

Logged Steps and Corresponding Errors

The following table provides a mapping of the step numbers found in the log file to the actual action performed by SUNSPOT. The step numbering does not reflect the actual execution order. Some values are also missing.

Step In Log File	Meaning
START	Logged after the initialization has completed successfully
Step1	Original file cannot be restored after tampering with the build process
Step2	Could not decrypt one of the targeted source code file's path (relative to the solution directory)
Step3	Could not create the file path for the targeted source code file
Step4	Could not get the size of the original source code file
Step5	Computation of the MD5 hash of the original source file failed
Step6	There was a mismatch between the expected target original source code file MD5 hash and the expected value
Step7	Could not successfully decrypt the backdoored source code
Step8	Computation of the MD5 hash of the backdoored source code failed
Step9	There was a mismatch between the expected backdoored source code data MD5 hash and the expected value
Step10	Could not create backup of the original source code file
Step11	Could not write the backdoored source code to disk (in the .tmp file)
Step12	Could not copy the temporary file with the backdoored source code (with the .tmp extension) to the path of the original source
Step14	Could not read the MsBuild.exe process memory to resolve its command-line arguments
Step15	The returned PEB address for the remote process is zero
Step16	Calling NtQueryInformationProcess failed
Step17	Could not create a handle to the MsBuild.exe process with SYNCHRONIZE access
Step18	Could not successfully wait for the MsBuild.exe process termination
Step19	Obtention of the address of the NtQueryInformationProcess function failed
Step20	Modification of the process security token to obtain SeDebugPrivileges failed
Step21	The number of currently running tampering threads exceeded 256, and SUNSPOT cannot track more threads

Step22	Unable to get a list of running processes
Step23	There was an error when enumerating the running processes list
Step30	Could not decrypt the solution name core.sln

Footnote:

1. https[:]//blog.xpnsec[.]com/how-to-argue-like-cobalt-strike/