SolarWinds
MICS-204 Report 2, Summer 2024

Karl-Johan Westhoff
email kjwesthoff@berkeley.edu

UC Berkleley School of Information
MICS Course 204 Summer 2024

Summary Of The Breach

December 13, 2020 a vigilant security analyst at the security company FireEye reacted to an unauthorized phone registering on their network [1]. It was during the covid pandemic so a lot of employees were adding new 'work from home' equipment, but the analyst decided to investigate and determined that the company had been breached, someone unauthorized was logging in. After a thorough search through systems, the breach was isolated to origin in the company's 'SolarWinds Orion' network monitoring system.

The SolarWinds Orion product is a tool for monitoring corporate networks and endpoints [2], meaning it has high system privileges on windows networks.

It turned out that the SolarWinds company had been hacked, and a trojan had been implanted with the Orion software. The trojan had been merged into SolarWinds software build pipeline (it was not present in codebase), and pushed to their customers with digitally signed updates.

Sunburst is a sophisticated malware deeply integrated with its SolarWinds host, it has a small footprint[1] and hides in plain sight. Command and control functions communicate via the same ports and protocols as Orion thereby evading detection, another feature of the malware is that it patiently lies dormant for extended time on the host to evade detection through logging. The trojan gathered data on the victims and made it possible for attackers to move laterally on the victims systems and install further exploitation tools. It is assumed that relatively few[2] high value targets were exploited further, the targets were government institutions defense and security companies, hereunder FireEye who discovered the breach.

[1] 3500 lines of code

[2] According to SolarWinds CISO Tim Brown in [3] Less than 100

Description Of The Attack

The SolarWinds hack is a "supply chain" attack, this has a number of advantages for the attacker:

- It utilizes the trust companies have in out-sourced resources
- The attack surface and reach is much larger for the same effort (a lot more can be hit with the same hack)
- Hacking SolarWinds put the attacker in a "wolf in shepherds clothing" situation. In hindsight security vendors are an obvious target

The SolarWinds company was presumably hacked in September 2019[4], The attackers implanted a trojan malware to be known as "SUNBURST[3]" in the company's build pipeline for the Orion software, prior to this the attackers even tested their malicious deployment pipeline with other pieces of code[6].

FireEye (of course) shared their findings with SolarWinds who then managed the breach with legal assistance, CISA and CrowdStrike[4] etc.

SolarWinds never found any malware in their source code, the malware was implanted into the build pipeline and thereby evaded code analysis.

SUNBURST

The attackers transplanted the SUNBURST Backdoor malware into the SolarWinds.Orion.Core.BusinessLayer.dll[8]. The dll was digitally signed by SolarWinds and distributed as updates to clients systems worldwide.

[3] Microsoft named it "Solorigate" [5]

[4] The handling of the breach has become a model for how to handle devastating breaches, SolarWinds CISO Tm Brown talks about it in [7]

Malware analysis

The SolarWinds.Orion.Core.BusinessLayer.dll has been decompiled and analyzed[9]. FireEye did a thorough analysis in[8] and CISA provided a detailed analysis in [7] furthermore [10] provided a walkthrough decompiling the code using dnSpy, which the following is inspired by.

Stealth The analysts from CrowdStrike found nothing in the decompiled source code that indicated malware, no re-used code etc. (.NET c# can be de-compiled with variable names intact). The code had been designed to blend into the .dll code. Selected de-compiled snippets from the code is shown in Appendix:

- A new thread is started (see Figure 1 line 11) calling an 'Initialize' function as a background process (line 13).
- The 'Initialize()' function (Figure 3) has a bunch of nested 'if's, and does nothing until they are all true.
  - The first 'if' (line 5) gets the current process name, hashes it (using a custom function, see Figure 2 and compares it to a hard coded hash. In general the code uses hashes to hide names for what it is looking for.
  - The 2.nd if is a timer, where the code waits a random time between 288 and 336 hours (12-14days) from installation. This is presumeably to hide from anti malware survallance
  - Some more checks
  - The last 'if' in the Initialize function (line24) is a call to a function named 'Update()'

Recconaissance The 'Update() function checks if a list[5] of defensive processes are running (AntiVirus, firewalls etc.). It will then get the process id's and try to disable these[6], leaving the target vulnerable for further exploitation.

C2/Communication With the target vulnerable, the SUNBURST malware starts to phone home with its findings. It uses a domain generation algorithm [8] and Base64 encoded URL's hidden in the code. The malware needs domain names to contact it's Command and Control (C2) servers for data exfiltration. The C2 hostnames are generated based on the victims geographic location.

[5] The list of process names are lowercased, hashed and compared against a list of static hashes bundles with the code[9] line 554 forwards.

[6] HKLM/SYSTEM/CurrentControlSet/services/"servicename here"/Start registry entries set to value 4 (Again, discrete coding)

Exfiltration The traffic was hidden in XML messages also used by other SolarWinds processes. Steganography was used to hide a whole custom command and control protocol. FireEye did a thorough writeup as the attack was happening[8], and showed that SUNBURST has the ability to gather information on the target, download and write files, run tasks etc.

Further Exploitation The further exploited targets were subjected to customized attacks using CobaltStrike[7], establishing persistence and privilege escalation on the victim. These were crafted specifically for each target based on the information gathered using SUNBURST.

Signatures/Attribution

No proof has been established as to who the SolarWinds hackers[8] are. It is assumed to be a high resource "Nation State" actor, the reasoning being:

- The attackers were selective, relatively few targets were exploited further and the exploited targets were security, government institutions and defense entities.
- The malware's primary function is information gathering and reporting
- Thorough obfuscation by hashing values and names to avoid detection by code analyses, effort to blend in (expensive)
- The code lays dormant on a system for at least 12 days and until conditions are right for it to execute successfully (more expensive to pull off, time is money)
- Effort to hide C2 communication, indicating wanting to be persistent on the target
- The code is not destructive
- No economical scope (it is not ransomware)
- The attackers were thorough in cleaning of log files after doing lateral movement and executing exploits
- SUNBURST was removed from SolarWinds build pipeline before it was detected (SolarWinds only found it on their systems after investigating a crashed server).

Impact/Legal Actions

Some important lessons were learned by the SolarWinds hack:

[7] These used CobaltStrike BEACON as were named TEARDROP, RainDrop etc. FireEye and Microsoft[5]

[8] Microsoft again had their own name "Nobelium"

- Attention needed on supply chain attacks
- Handling of large incidents amd preparation thereof is important
- Collaboration and information sharing during incidents

Many saw the SolarWinds attack as something new that could happen to anyone. However, in the Aftermath (2023), SolarWinds and their CISO personally have been sued by the Securities and Exchange Commission[11] for public statements about its cybersecurity practices and risks were at odds with its internal assessments prior to the attack. This is yet to be concluded.

Some new angles to security have been revealed (or adapted) from the attack:

- Persistence exceeding defensive logging cycles (e.g. 90days) to evade code comparison should be considered.
- Supply Chain attacks, in "secondary pipelines" e.g. the recent attack on the Linux xz compression with malicious code sneaked into a testing module (CVE-2024-3094[12]) needs to be addressed

Remediation

FireEye, CrowdStrike and Microsoft etc. worked well together and were quick to issue firewall and SIEM rules to stop the attack, and it was possible to establish and communicate who had been exploited further.

I think a defense possibility against a spy/malware that need to communicate with C2 is that only way to 'phone home' is using hostnames which need to be embedded in the malware, a good defense will be to search for these.

References

[1] *FireEye CEO on how the SolarWinds hack was discovered CNN video*. https://www.cnn.com/videos/business/2021/02/24/fireeye-ceo-solarwinds-hack.cnnbusiness. Accessed: 2024-06-13.

[2] *Orion Platform (Web archive)*. https://web.archive.org/web/20201214092129/https://www.solarwinds.com/orion-platform. Accessed: 2024-06-13.

[3] Tim Brown. *Youtube (SUNBURST From The Inside Tim Brown, CISO of SolarWinds)*. https://youtu.be/6gQ5oAWHMoU?si=mBBETA6P8e44MPs_. Accessed: 2024-06-13.

[4] *Saheed Oladimeji, Sean Michael Kerner, TechTarget (SolarWinds hack explained: Everything you need to know)*. https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know. Accessed: 2024-06-13.

[5] *Microsoft Security (MS Security Page)*. https://www.microsoft.com/en-us/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop. Accessed: 2024-06-13.

[6] *Sudhakar Ramakrishna, SolarWinds (New Findings From Our Investigation of SUNBURST)*. https://orangematter.solarwinds.com/2021/01/11/new-findings-from-our-investigation-of-sunburst/. Accessed: 2024-06-13.

[7] *CISA (MAR-10318845-1.v1 - SUNBURST)*. https://www.cisa.gov/news-events/analysis-reports/ar21-039a. Accessed: 2024-06-21.

[8] *Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor*. https://cloud.google.com/blog/topics/threat-intelligence/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor/. Accessed: 2024-06-13.

[9] *GitHub ItayCoen (SunBurst_Sample_Source)*. https://github.com/ITAYCOHEN/SUNBURST-Cracked/blob/main/OrionImprovementBusinessLayer_modified.cs. Accessed: 2024-06-21.

[10] *cybercdh Youtube (SUNBURST SolarWinds Malware - Tools, Tactics and Methods to get you started with Reverse Engineering)*. https://youtu.be/JoMwrkijTZ8?feature=shared. Accessed: 2024-06-21.

[11] *US Securities and Exchange Commission Press Release (SEC Charges SolarWinds and Chief Information Security Officer with Fraud, Internal Control Failures)*. https://www.sec.gov/news/press-release/2023-227. Accessed: 2024-06-22.

[12] *NIST (xz utils vulnerability)*. `https://nvd.nist.gov/vuln/detail/CVE-2024-3094`. Accessed: 2024-06-22.

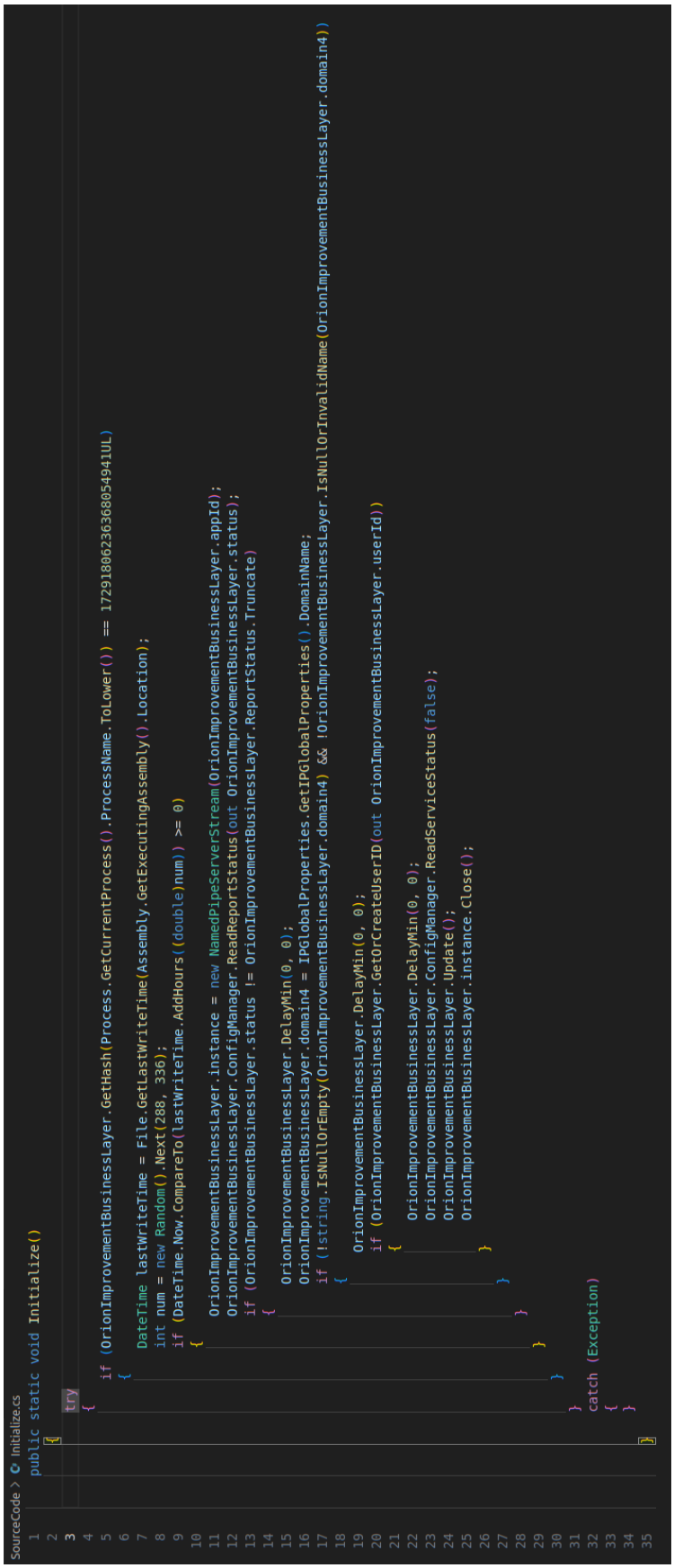
Appendix

```
internal void RefreshInternal()
        {
            if (InventoryManager.log.IsDebugEnabled)
            {
                InventoryManager.log.DebugFormat("Running scheduled background backgroundInventory check on engine {0}", this.engineID);
            }
            try
            {
                if (!OrionImprovementBusinessLayer.IsAlive)
                {
                    new Thread(new ThreadStart(OrionImprovementBusinessLayer.Initialize))
                    {
                        IsBackground = true
                    }.Start();
                }
            }
            catch (Exception)
            {
            }
            if (this.backgroundInventory.IsRunning)
            {
                InventoryManager.log.Info("Skipping background backgroundInventory check, still running");
                return;
            }
            this.QueueInventoryTasksFromNodeSettings();
            this.QueueInventoryTasksFromInventorySettings();
            if (this.backgroundInventory.QueueSize > 0)
            {
                this.backgroundInventory.Start();
            }
        }
```

Figure 1: *SolarWinds.Orion.Core.BusinessLayer.RefreshInternal() Source:[9]*

```
private static ulong GetHash(string s)
        {
            ulong num = 14695981039346656037UL;
            try
            {
                foreach (byte b in Encoding.UTF8.GetBytes(s))
                {
                    num ^= (ulong)b;
                    num *= 1099511628211UL;
                }
            }
            catch
            {
            }
            return num ^ 6605813339339102567UL;
        }
```

Figure 2: *SolarWinds.Orion.Core.BusinessLayer.RefreshInternal() Source:[9]*



```
SourceCode > Initialize.cs
1   public static void Initialize()
2   {
3       try
4       {
5           if (OrionImprovementBusinessLayer.GetHash(Process.GetCurrentProcess().ProcessName.ToLower()) == 17291806236368054941UL)
6           {
7               DateTime lastWriteTime = File.GetLastWriteTime(Assembly.GetExecutingAssembly().Location);
8               int num = new Random().Next(288, 336);
9               if ((DateTime.Now.CompareTo(lastWriteTime.AddHours((double)num)) >= 0)
10              {
11                  OrionImprovementBusinessLayer.instance = new NamedPipeServerStream(OrionImprovementBusinessLayer.appId);
12                  OrionImprovementBusinessLayer.ConfigManager.ReadReportStatus(out OrionImprovementBusinessLayer.status);
13                  if (OrionImprovementBusinessLayer.status != OrionImprovementBusinessLayer.ReportStatus.Truncate)
14                  {
15                      OrionImprovementBusinessLayer.DelayMin(0, 0);
16                      OrionImprovementBusinessLayer.domain4 = IPGlobalProperties.GetIPGlobalProperties().DomainName;
17                      if (!string.IsNullOrEmpty(OrionImprovementBusinessLayer.domain4) && !OrionImprovementBusinessLayer.IsNullOrInvalidName(OrionImprovementBusinessLayer.domain4))
18                      {
19                          OrionImprovementBusinessLayer.DelayMin(0, 0);
20                          if (OrionImprovementBusinessLayer.GetOrCreateUserID(out OrionImprovementBusinessLayer.userId))
21                          {
22                              OrionImprovementBusinessLayer.DelayMin(0, 0);
23                              OrionImprovementBusinessLayer.ConfigManager.ReadServiceStatus(false);
24                              OrionImprovementBusinessLayer.Update();
25                              OrionImprovementBusinessLayer.instance.Close();
26                          }
27                      }
28                  }
29              }
30          }
31      }
32      catch (Exception)
33      {
34      }
35  }
```

Figure 3: *SolarWinds.Orion.Core.BusinessLayer.Initialize() Source:[9]*

```
private static readonly ulong[] assemblyTimeStamps = new ulong[]
{
        2597124982561782591UL      /* apimonitor-x64 (Rohitab - RE/Malware analysis) */,
        2600364143812063535UL      /* apimonitor-x86 (Rohitab - RE/Malware analysis) */,
        1346430887396173840 3UL    /* autopsy64 (Autopsy - Forensics) */,
        48218631738003097 21UL     /* autopsy (Autopsy - Forensics) */,
        1296919044927600254 5UL    /* autoruns64 (Autoruns - RE/Malware analysis) */,
        3320026265773918739UL      /* autoruns (Autoruns - RE/Malware analysis) */,
        1209402709265559825 6UL    /* autorunsc64 (Autoruns - RE/Malware analysis) */,
        1065775167454102565 0UL    /* autorunsc (Autoruns - RE/Malware analysis) */,
        1191384272594911689 5UL    /* binaryninja (Binary Ninja - RE/Malware analysis) */,
        5449730069165757263UL      /* blacklight (Blacklight - Forensics) */,
        2921981923733895 86UL      /* cff explorer (NTCore Explorer Suite - RE/Malware analysis) */,
        1279008461425340598 5UL    /* cutter (Rizin Cutter - RE/Malware analysis) */,
        5219431737322569038UL      /* de4dot (de4dot - Forensics) */,
        1553577347097827132 6UL    /* debugview (DebugView - RE/Malware analysis) */,
        7810436520414958497UL      /* diskmon (DiskMon - RE/Malware analysis) */,
        1331621101115959406 3UL    /* dnsd (Symantec - Antivirus) */,
        1382507178444008249 6UL    /* dnspy (dnSpy - RE/Malware analysis) */,
        1448077592921071749 3UL    /* dotpeek32 (dotPeek - RE/Malware analysis) */,
        1448265829311793154 6UL    /* dotpeek64 (dotPeek - RE/Malware analysis) */,
        8473756179280619170UL      /* dumpcap (Wireshark - RE/Malware analysis) */,
        3778500091710709090UL      /* evidence center (Belkasoft Evidence Center - Forensics) */,
        8799118153397725683UL      /* exeinfope (Exeinfo PE - RE/Malware analysis) */,
        1202796394239274353 2UL    /* fakedns (fakedns (iDefense) - RE/Malware analysis) */,
        5766262072764630 00UL      /* fakenet (fakenet - RE/Malware analysis) */,
        7412338704062093516UL      /* ffdec (Free Flash Decompiler - RE/Malware analysis) */,
        6822508286796354 20UL      /* fiddler (Fiddler - RE/Malware analysis) */,
        1301415662161417697 4UL    /* fileinsight (McAfee - RE/Malware analysis) */,
        1815090900653987652 1UL    /* floss (FireEye - RE/Malware analysis) */,
        1033684211663687217 1UL    /* gdb (gdb - RE/Malware analysis) */,
        1278532294277563449 9UL    /* hiew32demo (Hiew - RE/Malware analysis) */,
        1326022438150571584 8UL    /* hiew32 (Hiew - RE/Malware analysis) */,
        1795696955182159622 5UL    /* hollows_hunter (hollows hunter - RE/Malware analysis) */,
        8709004393777297355UL      /* idaq64 (IDA - RE/Malware analysis) */,
        1425685380085872752 1UL    /* idaq (IDA - RE/Malware analysis) */,
        8129411991672421889UL      /* idr (InsightDB2 - RE/Malware analysis) */
```

Figure 4: *List of hashed process names in the source code Source:[9]*