Prevasio

prevasio.io/blog/sunburst-backdoor-a-deeper-look-into-the-solarwinds-supply-chain-malware

By

Sergei Shevchenko

December 15, 2020

Sunburst Backdoor: A Deeper Look Into The SolarWinds' Supply Chain Malware

Update: Next two parts of the analysis are available here and here.

As earlier reported by FireEye, the actors behind a global intrusion campaign have managed to trojanise SolarWinds Orion business software updates in order to distribute malware.

The original FireEye write-up already provides a detailed description of this malware. Nevertheless, as the malicious update **SolarWinds-Core-v2019.4.5220-Hotfix5.msp** was still available for download for hours since the FireEye's post, it makes sense to have another look into the details of its operation.

The purpose of this write-up is to provide new information, not covered in the original write-up. Any overlaps with the original description provided by FireEye are not intentional.

For start, the malicious component **SolarWinds.Orion.Core.BusinessLayer.dll** inside the MSP package is a non-obfuscated .NET assembly. It can easily be reconstructed with a .NET disassembler, such as ILSpy, and then fully reproduced in C# code, using Microsoft Visual Studio. Once reproduced, it can be debugged to better understand how it works.

In a nutshell, the malicious DLL is a backdoor. It is loaded into the address space of the legitimate SolarWinds Orion process **SolarWinds.BusinessLayerHost.exe** or **SolarWinds.BusinessLayerHostx64.exe**.

The critical strings inside the backdoor's class **SolarWinds.Orion.Core.BusinessLayer.OrionImprovementBusinessLayer** are encoded with the *DeflateStream* Class of the .NET's *System.IO.Compression* library, coupled with the standard base64 encoder.

Initialisation

Once loaded, the malware checks if its assembly file was created earlier than 12, 13, or 14 days ago. The exact number of hours it checks is a random number from 288 to 336.

Next, it reads the application settings value **ReportWatcherRetry**. This value keeps the reporting status, and may be set to one of the states:

- New (4)
- Truncate (3)
- Append (5)

When the malware runs the first time, its reporting status variable **ReportWatcherRetry** is set to **New (4)**.

The reporting status is an internal state that drives the logic. For example, if the reporting status is set to **Truncate**, the malware will stop operating by first disabling its networking communications, and then disabling other security tools and antivirus products.

In order to stay silent, the malware periodically falls asleep for a random period of time that varies between 30 minutes and 2 hours.

At the start, the malware obtains the computer's domain name. If the domain name is empty, the malware quits.

It then generates a 8-byte User ID, which is derived from the system footprint. In particular, it is generated from MD5 hash of a string that consists from the 3 fields:

- the first or default operational (can transmit data packets) network interface's physical address
- computer's domain name

- UUID created by Windows during installation (machine's unique ID)

Even though it looks random, the User ID stays permanent as long as networking configuration and the Windows installation stay the same.

Domain Generation Algorithm

The malware relies on its own *CryptoHelper* class to generate a domain name. This class is instantiated from the 8-byte User ID and the computer's domain name, encoded with a substitution table: **"rq3gsalt6u1iyfzop572d49bnx8cvmkewhj"**.

For example, if the original domain name is "*domain*", its encoded form will look like: "*n2huov*".

To generate a new domain, the malware first attempts to resolve domain name "*api.solarwinds.com*". If it fails to resolve it, it quits.

The first part of the newly generated domain name is a random string, produced from the 8-byte User ID, a random seed value, and encoded with a custom base64 alphabet **"ph2eifo3n5utg1j8d94qrvbmk0sal76c"**.

Because it is generated from a random seed value, the first part of the newly generated domain name is random.

For example, it may look like "*fivu4vjamve5vfrt*" or "*k1sdhtslulgqoagy*".

To produce the domain name, this string is then appended with the earlier encoded domain name (such as "*n2huov*") and a random string, selected from the following list:

- .appsync-api.eu-west-1[.]avsvmcloud[.]com
- .appsync-api.us-west-2[.]avsvmcloud[.]com
- .appsync-api.us-east-1[.]avsvmcloud[.]com
- .appsync-api.us-east-2[.]avsvmcloud[.]com

For example, the final domain name may look like:

fivu4vjamve5vfrtn2huov[.]appsync-api.us-west-2[.]avsvmcloud[.]com

or

k1sdhtslulgqoagyn2huov[.]appsync-api.us-east-1[.]avsvmcloud[.]com

Next, the domain name is resolved to an IP address, or to a list of IP addresses. For example, it may resolve to **20.140.0.1**.

The resolved domain name will be returned into **IPAddress** structure that will contain an **AddressFamily** field - a special field that specifies the addressing scheme.

If the host name returned in the **IPAddress** structure is different to the queried domain name, the returned host name will be used as a C2 host name for the backdoor.

Otherwise, the malware will check if the resolved IP address matches one of the patterns below, in order to return an 'address family':

IP Address	Subnet Mask	'Address Family'
10.0.0.0	255.0.0.0	Atm
172.16.0.0	255.240.0.0	Atm
192.168.0.0	255.255.0.0	Atm
224.0.0.0	240.0.0.0	Atm
fc00::	fe00::	Atm
fec0::	ffc0::	Atm
ff00::	ff00::	Atm
41.84.159.0	255.255.255.0	Ipx
74.114.24.0	255.255.248.0	Ipx
154.118.140.0	255.255.255.0	Ipx
217.163.7.0	255.255.255.0	Ipx
20.140.0.0	255.254.0.0	ImpLink
96.31.172.0	255.255.255.0	ImpLink
131.228.12.0	255.255.252.0	ImpLink
144.86.226.0	255.255.255.0	ImpLink
8.18.144.0	255.255.254.0	NetBios
18.130.0.0	255.255.0.0	NetBios
71.152.53.0	255.255.255.0	NetBios
99.79.0.0	255.255.0.0	NetBios
87.238.80.0	255.255.248.0	NetBios
199.201.117.0	255.255.255.0	NetBios
184.72.0.0	255.254.0.0	NetBios

For example, if the queried domain resolves to **20.140.0.1**, it will match the entry in the table **20.140.0.0**, for which the returned 'address family' will be **ImpLink**.

The returned 'address family' invokes an additional logic in the malware.

Disabling Security Tools and Antivirus Products

If the returned 'address family' is **ImpLink** or **Atm**, the malware will enumerate all processes and for each process, it will check if its name matches one of the pre-defined hashes.

Next, it repeats this processed for services and for the drivers installed in the system. If a process name or a full path of an installed driver matches one of the pre-defined hashes, the malware will disable it.

For hashing, the malware relies on Fowler–Noll–Vo algorithm.

For example, the core process of Windows Defender is **MsMpEng.exe**. The hash value of "*MsMpEng*" string is **5183687599225757871**. This value is specifically enlisted the malware's source under a variable name **timeStamps**:

```
timeStamps = new ulong[1]
{
   5183687599225757871uL
}
```

The service name of Windows Defender is **windefend** - the hash of this string (**917638920165491138**) is also present in the malware body. As a result, the malicioius DLL will attempt to stop the Windows Defender service.

In order to disable various security tools and antivirus products, the malware first grants itself **SeRestorePrivilege** and **SeTakeOwnershipPrivilege** privileges, using the native **AdjustTokenPrivileges()** API. With these privileges enabled, the malware takes ownership of the service registry keys it intends to manipulate.

The new owner of the keys is first attempted to be explicitly set to Administrator account. If such account is not present, the malware enumerates all user accounts, looking for a SID that represents the administrator account.

The malware uses Windows Management Instrumentation query "*Select * From Win32_UserAccount*" to obtain the list of all users. For each enumerated user, it makes sure the account is local and then, when it obtains its SID, it makes sure the SID begins with **S-1-5-** and ends with **-500** in order to locate the local administrator account.

Once such account is found, it is used as a new owner for the registry keys, responsible for manipulation of the services of various security tools and antivirus products. With the new ownership set, the malware then disables these services by setting their **Start** value to **4** (Disabled):

registryKey2.SetValue("Start"), 4, RegistryValueKind.DWord);

HTTP Backdoor

If the returned 'address family' for the resolved domain name is **NetBios**, as specified in the lookup table above, the malware will initialise its **HttpHelper** class, which implements an HTTP backdoor.

The backdoor commands are covered in the FireEye write-up, so let's check only a couple of commands to see what output they produce.

One of the backdoor commands is **CollectSystemDescription**. As its name suggests, it collects system information. By running the code reconstructed from the malware, here is an actual example of the data collected by the backdoor and delivered to the attacker's C2 with a separate backdoor command **UploadSystemDescription**:

1. %DOMAIN_NAME%
2. S-1-5-21-298510922-2159258926-905146427
3. DESKTOP-VL39FPO
4. UserName
5. [E] Microsoft Windows NT 6.2.9200.0 6.2.9200.0 64
6. C:\WINDOWS\system32
7. 0
8. %PROXY_SERVER%

Description: Killer Wireless-n/a/ac 1535 Wireless Network Adapter #2
MACAddress: 9C:B6:D0:F6:FF:5D
DHCPEnabled: True
DHCPServer: 192.168.20.1
DNSHostName: DESKTOP-VL39FPO
DNSDomainSuffixSearchOrder: Home
DNSServerSearchOrder: 8.8.8.8, 192.168.20.1
IPAddress: 192.168.20.30, fe80::8412:d7a8:57b9:5886
IPSubnet: 255.255.255.0, 64
DefaultIPGateway: 192.168.20.1, fe80::1af1:45ff:feec:a8eb
NOTE: Field #7 specifies the number of days (0) since the last system reboot.

GetProcessByDescription command will build a list of processes running on a system. This command accepts an optional argument, which is one of the custom process properties enlisted here.

If the optional argument is not specified, the backdoor builds a process list that looks like:

[1720] svchost
[8184] chrome

```
[ 4732] svchost
```

If the optional argument is specified, the backdoor builds a process list that includes the specified process property in addition to parent process ID, username and domain for the process owner.

For example, if the optional argument is specified as "*ExecutablePath*", the **GetProcessByDescription** command may return a list similar to:

```
[ 3656] sihost.exe       C:\WINDOWS\system32\sihost.exe      1720 DESKTOP-VL39FPO\UserName
[ 3824] svchost.exe      C:\WINDOWS\system32\svchost.exe       992 DESKTOP-VL39FPO\UserName
[ 9428] chrome.exe       C:\Program Files (x86)\Google\Chrome\Application\chrome.exe      4600
DESKTOP-VL39FPO\UserName
```

Other backdoor commands enable deployment of the 2nd stage malware. For example, the **WriteFile** command will save the file:

```
using (FileStream fileStream = new FileStream(path, FileMode.Append, FileAccess.Write))
{
   fileStream.Write(array, 0, array.Length);
}
```

The downloaded 2nd stage malware can then the executed with **RunTask** command:

```
using (Process process = new Process())
{
   process.StartInfo = new ProcessStartInfo(fileName, arguments)
   {
      CreateNoWindow = false,
      UseShellExecute = false
   };
   if (process.Start())
   ...
```

Alternatively, it can be configured to be executed with the system restart, using registry manipulation commands, such as **SetRegistryValue**.