Detecting Supernova Malware: SolarWinds Continued | Splunk

By Splunk

Published: 2021-01-04 · Archived: 2026-04-05 13:28:43 UTC

As organizations were catching their breath and winding down for the holidays, a fascinating twist in the SolarWinds Orion “Sunburst” intrusions began to appear.

Supernova Timeline

On December 15, GuidePoint Security posted their analysis of a .NET webshell called “Supernova” that was originally disclosed in the initial FireEye investigation. Two days later, Palo Alto Networks followed up with their analysis of Supernova. On December 18, Microsoft released a comprehensive report on “Solorigate”, the compromised supply chain DLL that was part of the Sunburst intrusion first reported by FireEye. Under the “Additional Malware Discovered” section, Microsoft calls out the Supernova malware that was uncovered during their research, as well as the hypothesis that because the malware does not conform to the other aspects of the Sunburst attack, Supernova may have originated from another APT group!

On December 24, SolarWinds released its first Security Advisory that included both Sunburst and Supernova. This advisory has been updated multiple times since then as new information has been made available. Supernova, SolarWinds clarified, appeared to be separate from the Sunburst attack and the malware leveraged a vulnerability found in the Orion platform. This new vulnerability and associated malware allows adversaries another method of access. DHS CISA has updated their initial guidance to include this new vector.

Based on this newly released research (and this really is the Reader’s Digest version, believe me), we are going to take a look at Supernova – what it is and how it is leveraged. We will also take a look at various detection methods, using data models and SPL that can identify an adversary leveraging this attack vector.

What Is Supernova?

Supernova was originally identified during the analysis of Sunburst, the SolarWinds Orion intrusions. During this initial analysis, Supernova Yara rules were created alongside other elements of Sunburst because it was initially assumed to be part of the same intrusion. Based on the new hypothesis that Supernova is distinct from Sunburst, FireEye removed the Yara rules from their GitHub repo but the original rules are still accessible.

Supernova leverages what was a zero-day vulnerability to install a trojanized .NET DLL. It is important to note that this DLL is not digitally signed like the Sunburst DLL was, which is one of the reasons multiple researchers believe that this is a different threat actor using a vulnerability to load their malicious code to vulnerable systems.

The malware that is loaded is a web shell. This MITRE ATT&CK technique, T1505, is used by adversaries to backdoor web servers and establish persistent access to systems. For a deeper dive, here is a nice primer of web shells by Acunetix. Go ahead, I will wait. What makes this web shell extra nasty is that it is built to run in-memory

which makes it more difficult for detection and forensic analysts to do additional analysis post-breach. Notice I said more difficult, but not impossible.

What Can I Do?

SolarWinds has continued to update their security advisory that lays out both the Sunburst AND Supernova issues and which versions of their software are impacted. Patches are available for both issues. In fact, depending on the patch applied, both Sunburst and Supernova can be mitigated at the same time!

How Can I Detect Supernova In My Environment?

If you have read this far, you have a foundational understanding that Supernova and Sunburst are separate issues and you are likely looking to take some action. Here are some detections that may be useful to you in your environment.

Readers of this blog may be using Splunk Enterprise or Splunk Enterprise Security. While anyone can use data models, I recognize that not everyone does. With that in mind, the detections below are provided in SPL, as well as using data models and the tstats command. Using the tstats command will provide a better overall search performance. If you are not using data models, the SPL search should still be helpful because the search criteria still applies. However, the field names may be different based on the sourcetypes in your environment compared to the examples below.

These are somewhat broad searches as they stand but can be refined further based on IP address or other attributes. Good asset management will help isolate systems to search against. If you are not using tstats and data models for your searches and just want to use SPL, don't forget to use **index=<insert index names>** to narrow down the search to only indexes that contain the applicable events.

CERT/CC issued a new SolarWinds Orion API vulnerability alert on December 26th. Assuming you have a recent vulnerability scan with this latest alert added and you are ingesting vulnerability scanning data into Splunk and populating the vulnerability data model, this search could potentially uncover vulnerable systems.

```
 | tstats count from datamodel=Vulnerabilities.Vulnerabilities where
Vulnerabilities.cert=VU#843464 OR Vulnerabilities.cert=843464 OR
Vulnerabilities.cve=CVE-2020-10148 groupby Vulnerabilities.dest
Vulnerabilities.dvc Vulnerabilities.signature Vulnerabilities.vendor_product _time span=1s
```

The SPL search will be dependent on the event source, whether that is Nessus, Qualys or others, but it could be as simple as this, provided the vulnerability vendor utilizes CERT/CC or CVE.

```
 index=<index where vulnerability data is stored>
sourcetype=<vulnerability scanner> (VU#843464 OR 843464 OR CVE-2020-10148)
```

The footprint of this intrusion is limited due to the absence of the web shell existing on disk, but file hashes do exist for the trojanized .NET DLL. VirusTotal currently has 59 engines detecting it. Signature names are available

to search against as well, but they will vary based on your anti-virus vendor.

Depending on how you are identifying file system events, the following search may allow you to identify if the file hashes associated with the trojanized DLL have been written to disk. Note that this will be dependent on the events collected being written to the Endpoint data model.

```
 | tstats count from datamodel=Endpoint.Filesystem where
Filesystem.file_name=*logoimagehandler.ashx* OR
Filesystem.file_hash=C15abaf51e78ca56c0376522d699c978217bf041a3bd3c71d09193efa5717c71
OR Filesystem.file_hash=75af292f34789a1c782ea36c7127bf6106f595e8 OR
Filesystem.file_hash=56ceb6d0011d87b6e4d7023d7ef85676 groupby
Filesystem.file_name Filesystem.file_path Filesystem.dest
Filesystem.file_hash Filesystem.vendor_product Filesystem.user _time span=1s
```

For those running Sysmon, searching for EventCode 11 (FileCreate) can also be helpful to determine if and when the DLL was written to disk.

```
 index=<index where endpoint data is stored>
sourcetype=xmlwineventlog:microsoft-windows-sysmon/operational EventCode=11
file_name=*logoimagehandler.ashx*
| table _time host Image Computer TargetFilename
```

At the very least, a search for these three hashes (SHA256, SHA1 and MD5) will provide a place to start to determine if the malware exists on your Solarwinds Orion systems.

SHA256: C15abaf51e78ca56c0376522d699c978217bf041a3bd3c71d09193efa5717c71
SHA1: 75af292f34789a1c782ea36c7127bf6106f595e8
MD5: 56ceb6d0011d87b6e4d7023d7ef85676

If you are looking for DLLs being loaded in a specific process, Sysmon Event Code 7 (Image loaded) can also be used to look for the trojanized .NET DLL being invoked.

```
 index=<index where endpoint data is stored>
sourcetype=xmlwineventlog:microsoft-windows-sysmon/operational EventCode=7
(file_name=*logoimagehandler.ashx* OR
SHA256=C15abaf51e78ca56c0376522d699c978217bf041a3bd3c71d09193efa5717c71 OR
SHA1=75af292f34789a1c782ea36c7127bf6106f595e8 OR
MD5=56ceb6d0011d87b6e4d7023d7ef85676)
| table _time Image ImageLoaded Computer
```

SentinelOne also released a blog where they developed a proof of concept that used the same technique that Supernova uses for in-memory compilation of .NET. They identified that CSC.exe and CVTRES.exe are created as child processes during execution. Please keep in mind this is a tactic to hunt, not to deploy as a signature with your SIEM. Because many .NET apps can do this, I want to caution that this is not an indicator of compromise

but, it may be worth the time to run a search that looks something like this to determine if .NET assemblies are being compiled and then hunt for additional actions occurring immediately after this behavior on vulnerable systems.

```
 | tstats count from datamodel=Endpoint.Processes where
Processes.process_exec=cvtres.exe Processes.parent_process_exec=csc.exe
groupby Processes.process_exec Processes.process_id Processes.process
Processes.parent_process_exec Processes.parent_process
Processes.parent_process_id Processes.dest Processes.user
Processes.vendor_product _time span=1s
```

```
 index=<index where endpoint data is stored>
sourcetype=xmlwineventlog:microsoft-windows-sysmon/operational EventCode=1
CommandLine=*cvtres.exe* ParentCommandLine=*csc.exe*
| table _time CommandLine ParentCommandLine User host ProcessId ParentProcessId
```

Because the web shell exists in memory, the best opportunity to see this will be found in events like web or network traffic events. Guidepoint's blog provides some pseudo query that can be adapted to be used with Stream for Splunk or Bro/Zeek or other data sets that contain information around http and URI filenames and parameters. Even in the absence of endpoint data, if web or network traffic data exists, searches like these could be used as a starting point. As previously mentioned, if you are not using Splunk for Stream, the SPL can be adapted to your specific data source.

```
 | tstats count from datamodel=Web.Web where
web.url=*logoimagehandler.ashx*codes* OR
Web.url=*logoimagehandler.ashx*clazz* OR
Web.url=*logoimagehandler.ashx*method* OR
Web.url=*logoimagehandler.ashx*args* groupby Web.src Web.dest Web.url
Web.vendor_product Web.user Web.http_user_agent _time span=1s
```

```
 index=<index where network/web data is stored> sourcetype=stream:http
(url=*logoimagehandler.ashx*codes* OR Web.url=*logoimagehandler.ashx*clazz*
OR Web.url=*logoimagehandler.ashx*method* OR
Web.url=*logoimagehandler.ashx*args*)
| table _time src_ip src_port dest_ip dest_port url transport status
```

```
 | tstats count from datamodel=Web.Web where Web.http_content_type=text/plain
Web.dest=(insert your SolarWinds IP here, we are looking for inbound traffic)
Web.url=*logoimagehandler.ashx* groupby Web.src Web.dest Web.url
Web.vendor_product Web.user Web.http_user_agent _time span=1s
```

```
 index=<index where network/web data is stored> sourcetype=stream:http
dest_ip=(insert your SolarWinds IP here, we are looking for inbound traffic)
url=*logoimagehandler.ashx*
| table _time src_ip src_port dest_ip dest_port url transport status
```

I recognize that dealing with another vulnerability and its associated malicious code so soon after Sunburst is probably not the way anyone wanted to wrap up the year and start a new one but hopefully, this provides a way forward to jump-start your detections as your organization patches its vulnerable SolarWinds systems.

Thanks!

John Stoner
