Unpacking the Hype: What You Can (and Can't) Do to Prevent/Detect Software Supply Chain Attacks

Jake Williams

SANS

A SANS Whitepaper

Unpacking the Hype: What You Can (and Can't) Do to Prevent/Detect Software Supply Chain Attacks

Written by **Jake Williams**

February 2021

In December 2020, the security community was shocked when FireEye, a staple in the InfoSec community, announced that it had been breached and its offensive security tools stolen. Less than a week later, while the world was still reacting, we learned that SolarWinds had been breached and used to infect FireEye through its Orion Network Management System (NMS) software. Investigators later determined that the attackers used backdoored versions of Orion to infect multiple other government and commercial targets. Although the scope of the threat actors' operations went far beyond Orion, this paper focuses on the SolarWinds compromise and what it can teach us about detecting software supply chain compromises.

SolariGate[1] Background

On December 8, 2020, FireEye notified the cybersecurity community, through a blog post, that it had been the subject of a cybersecurity incident.[2] FireEye stated its belief that the attackers compromised and exfiltrated its Red Team tools, useful for offensive operations. FireEye immediately released signatures for these tools to limit their usefulness by an adversary. Figure 1 presents a timeline of events for SolariGate.[3]

[1] "SolariGate" (also Solarigate and Solorigate) is Microsoft's label for the attack in Windows Defender. "SolarWinds Breach Used to Infiltrate Customer Networks (Solarigate)," https://isc.sans.edu/forums/diary/SolarWinds+Breach+Used+to+Infiltrate+Customer+Networks+Solarigate/26884

[2] "FireEye Shares Details of Recent Cyber Attacks, Actions to Protect Community," www.fireeye.com/blog/products-and-services/2020/12/fireeye-shares-details-of-recent-cyber-attack-actions-to-protect-community.html

[3] For a more detailed timeline, see https://github.com/CyberSecOps/SolarWinds-Sunburst-Solorigate-Supernova-FireEye

Continuing its investigation, FireEye discovered the **SUNBURST** backdoor and notified SolarWinds of the issue on December 12, 2020. Knowledge of the **SUNBURST** backdoor became public the following evening (December 13). SolarWinds released patched software on December 15, 2020. FireEye attributed this activity to threat group UNC2452. Other researchers attribute the intrusion to a Russian government threat actor, though there is some question in the community about whether it was the Russian Federal Security Service (FSB) or the Russian Foreign Intelligence Service (SVR). This distinction is probably not important for the majority of organizations.

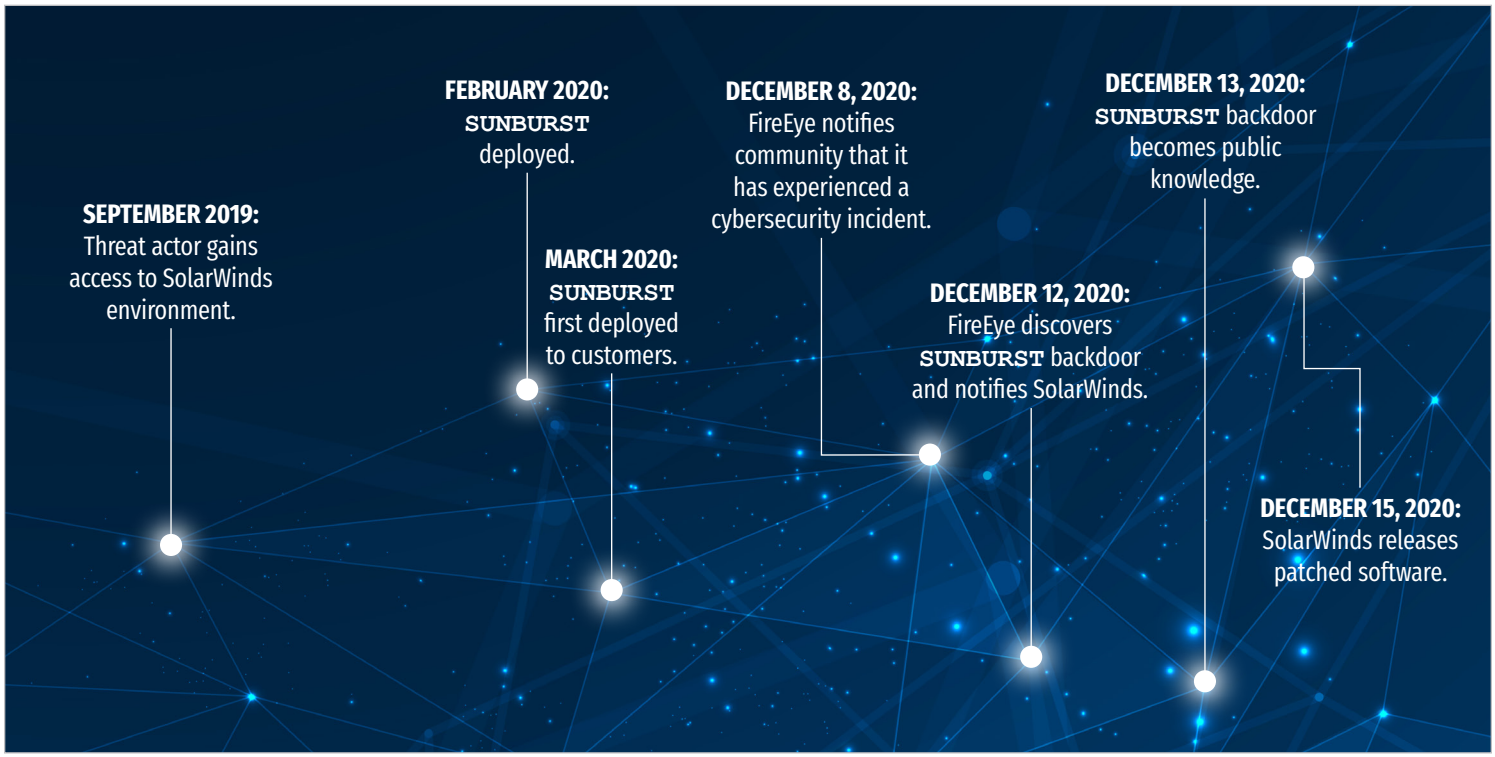

Figure 1. SolariGate Timeline

Additional malware, initially linked to the Orion supply chain compromise, called **SUPERNOVA** was discovered. Researchers (including Microsoft) have determined, however, that the **SUPERNOVA** malware is unrelated to the SolariGate compromise, primarily due to its low level of sophistication.[4] Also, **SUPERNOVA** was not attributed to a supply chain compromise because, unlike **SUNBURST**, it was not shipped with Orion.

Malware named **SUNSPOT** was eventually discovered in the SolarWinds environment. This malware was used to infect the Orion build process with the **SUNBURST** backdoor. Based on the investigation, it is believed that the threat actor first gained access to the SolarWinds environment in September 2019 and **SUNBURST** was deployed in February 2020, with its first deployment to customers in March 2020. **SUNSPOT** was only deployed in the SolarWinds environment, so most organizations do not need not to worry about it, specifically.

[4] "Analyzing Solorigate, the compromised DLL File that started a sophisticated cyberattack, and How Microsoft Defender helps protect customers," www.microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect

SolarWinds stated to the SEC that it believes the backdoored Orion software was deployed on 18,000 servers. Of those, a small number of affected organizations were targeted for follow-on activity. In particular, U.S. government agencies including the Department of the Treasury were targeted. Commercial organizations, most notably Microsoft, were targeted for additional exploitation.

Microsoft reported that some of its source code was accessed by attackers, but it claims this did not measurably impact the security of its products. Microsoft implements a secure development life cycle—and has since at least the development of Vista—so its claim is likely correct. Further, any number of governments already have at least partial access to audit source code, so it is unlikely that any source code compromise would be a significant enabler for discovering new vulnerabilities. However, read-only source code access could be useful for building rootkits and understanding the heavily obfuscated Kernel Patch Protection (**`PatchGuard`**), a kernel-mode rootkit prevention mechanism.

If attackers had gained writable access to Microsoft source code, as was the case at SolarWinds, the situation would have been very different. While Microsoft did not reveal which products' source code was accessed by attackers, it is safe to presume that most Microsoft products have a larger user base than Orion. Had attackers been able to backdoor Microsoft's source code, the scope of this operation would have been exponentially larger.

Some have cast doubt on Microsoft's claims that it can verify its source code was only accessed and not modified. Although this would be a legitimate concern with many organizations, it is likely that Microsoft does have significant enough logging to conclusively ensure the integrity of its source code.

Once inside target networks, the SolariGate attackers also employed a mechanism for persisting access to Active Directory Federation Services (ADFS) that has been called **`Golden SAML`**. This technique has been publicly discussed since at least 2017, but this was the first large-scale public use of the technique. With **`Golden SAML`**, attackers export the private key used by domain controllers to sign **`SAML`** assertions and can maintain access to the ADFS environment until the key is rotated. For those familiar with Active Directory Golden Ticket attacks, the **`SAML`** private key holds much the same significance as the **`krbtgt`** password hash.

JetBrains Involvement

The *New York Times (NYT)* published a story indicating that JetBrains software may have been involved in the SolarWinds breach.[5] Two other news outlets also reported the story but appear to have been picking up the *NYT* story without independent sourcing. Nothing additional has materialized about JetBrains' involvement, and the company itself states that it has no knowledge of an investigation. It is likely that JetBrains, like any other software used by SolarWinds in its build process, was of interest to U.S. investigators. However, there is no credible evidence that JetBrains played any role in the SolariGate attack. As such, we will not discuss JetBrains further in this paper.

[5] "Widely Used Software Company May Be Entry Point for Huge U.S. Hacking," www.nytimes.com/2021/01/06/us/politics/russia-cyber-hack.html

FireEye Involvement

Supply chain compromises are notoriously difficult to detect. The SolarWinds breach was especially difficult because of the steps taken to counter detection of the implant. Even if analysts somehow had taken interest in researching the **`SUNBURST DLL`**, nothing made it stand out as malicious. FireEye reports that it first began investigating because attackers tried to register another device usable for multifactor authentication (MFA) on a compromised account. The attackers presumably did not know that FireEye had an audit process in place that would detect changes to MFA. Because the change was unexpected, the attackers were discovered.

After FireEye began investigating, tracking the attackers back to the SolarWinds Orion server was likely relatively easy. Not only are there are a limited number of lateral movement techniques, but FireEye also investigates these techniques regularly and doubtless has the appropriate logging in place to investigate any such operation in its own network.

For several reasons, tracking the attackers back to the Orion server does not come close to implying that there was a software supply chain compromise at SolarWinds. First, NMSs are frequent targets of attackers due to the multitude of credentials stored in them. In many networks, an NMS provides more credentials than targeting a domain controller (and often yields access to domain admin credentials). Second, even if network monitoring was configured to capture all internal traffic, the Orion server talks to many destinations on the network. This communication restricts the usefulness of any NetFlow in determining how attackers compromised the Orion server and where they may have moved laterally post-compromise.

We can logically infer that FireEye has more logging in place than most organizations. Perhaps more importantly, it has the expertise to understand what is (and isn't) normal in the logs. Between logging and analysis of collected data, FireEye was probably able to quickly conclude that normal methods of lateral movement were not used to compromise the Orion server. Note that the completeness of logs is paramount here. If coverage gaps in logs exist, Occam's razor would direct analysts to presume that attackers used a technique not covered in those logs.

Because we can assume that FireEye had relatively complete logging that revealed no indication of how the attackers arrived at the Orion server, it likely only then considered the software supply chain compromise vector. While FireEye succeeded in discovering the SolariGate attackers, most others who tracked attackers back to an Orion server likely would not. Take Volexity, for instance, which noted that it had investigated a breach at a think tank where the attacker was tracked to Orion in June and July 2020.[6] The **`SUNBURST DLL`** remained undiscovered, however, probably due to customer resourcing constraints. As detailed in the sidebar titled, "What's So Hard About Identifying a Software Supply Chain Attack?", conclusively identifying a software supply chain attack is a highly resource-intensive activity.

What's So Hard About Identifying a Software Supply Chain Attack?

Just how resource-intensive is identifying a backdoor inserted through a software supply chain attack—and why can't most organizations perform this work? Analyzing a commercial software package to discover an intentionally hidden backdoor is anything but easy. This type of analysis is something most organizations simply don't have the resources to handle. The analysis requires skilled reverse engineers and an investment in the appropriate tooling. Although some large organizations have a reverse engineer on staff, reverse engineering resources are typically cost-justified for specific malware analysis. It's this writer's experience that most in-house malware analysts have more than enough malware to analyze without shifting focus to investigate a supply chain attack. Even if the organization has the resources, the analysis activity required to discover a compromised software package is still highly speculative, meaning that organizations should generally expect a low ROI.

[6] "Dark Halo Leverages SolarWinds Compromise to Breach Organizations," www.volexity.com/blog/2020/12/14/dark-halo-leverages-solarwinds-compromise-to-breach-organizations

Unlike most targeted organizations, FireEye occupies a unique position in the overall SolariGate operation. FireEye is a target that likely has the following:

- A relatively complete record of activity on the network (all relevant logging)
- Adequate retention of log data
- In-house experience investigating these specific (and talented) threat actors
- In-house reverse engineering capabilities and tooling

In addition to having the necessary resources to investigate and identify a software supply chain attack, FireEye also has motive to do so. After notifying the public that its own offensive toolkit had been stolen, FireEye was in a position where they needed to conclusively determine the source of the breach, even if for no other reason than public relations. Other investigators had previously tracked intrusions to backdoored Orion servers but were unable to dedicate resources to investigating the source.

The purpose of detailing this background is to show what it took to discover that an attacker had inserted a backdoor in the Orion code base. Figure 2 presents this complex combination of conditions in graphical form. The narrative in this section is not intended as an indictment of expertise or commitment for the organizations that identified attacker activity and tracked it to an Orion server but then stopped short of discovering the **SUNBURST** backdoor. As we discuss detecting supply chain attacks going forward, we must understand that the needed level of expertise and resourcing cannot scale to address the multitudes of software used in a modern organization. This state of affairs is true even for a firm like FireEye, which has the expertise in-house as part of its core competencies.

Figure 2. What It Took to Discover the SolariGate Attack

[7] It is recommended that logs and NetFlow be retained for up to nine months.

Techniques for (Not) Preventing Supply Chain Attacks

Numerous techniques have been proposed by organizations to prevent software supply chain attacks. In this section, we will discuss three popular suggestions:

- Installing software in instrumented environments
- Antivirus scanning
- Third-party risk assessments

Unfortunately, none of these are likely to be effective. In this section, we discuss why.

Installing Software in Instrumented Environments

One proposed solution for identifying software and supply chain assurance (SSCA) is to install all third-party software in an isolated environment for some time prior to installing the same software in the production environment. The isolated environment must be instrumented in such a way that any backdoors would be detected before the software is redeployed in production.

This approach has several practical barriers to adoption, the most significant one being that every piece of third-party software would need to be installed twice (once in the isolated environment and once in production), making it prohibitively resource-intensive for most organizations. In addition, many software packages attempt to automatically update, complicating this process. Finally, for this approach to be maximally effective, the instrumented environment must mimic the configuration of the production environment. The **`SUNBURST`** malware checks to ensure that the Orion server it is installed on is part of a Windows domain, presumably to prevent analysis in an isolated environment. Rather than focus solely on the specifics used by **`SUNBURST`**, it is important to note that future implants will implement other self-defense mechanisms that account for any increased adoption of instrumented environments that might occur.

Another practical issue is that the instrumented environment must be able to observe any malicious activity that would-be backdoored software would exhibit. Even if an instrumented environment can observe activity, how will investigators separate the signal from the noise? This approach requires a level of knowledge about the third-party software that most vendors cannot readily muster.

Detecting beaconing activity sounds easy in theory, but most software makes legitimate outbound connections to a variety of destinations. Separating signal from noise will be difficult here as well. In addition, there are other actions compromised software could take, such as:

- Opening a listening port and waiting for an inbound connection
- Modifying other files on the operating system
- Changing registry settings to modify security posture

While instrumentation might catch many of these issues more easily, each is likely to generate significant noise through legitimate operations. We must assume, however, that attackers with sufficient capabilities to target a software supply chain would not telegraph their intent with any obvious modifications such as these.

The Verdict

Analysis of recent software supply chain attacks suggest that this method is not likely to be effective. The SolarWinds backdoor performed multiple checks for a normal environment before exhibiting any detectable activity. It also waited 10–12 days before beaconing the first time. In the case of **`MeDoc`** (the delivery vehicle for **`NotPetya`**),[8] the software also waited to beacon and, even then, did so only infrequently. Even if an organization can expend the resources necessary to pre-install all third-party software (and all updates) in isolated and instrumented environments, the odds of detecting a backdoor are low. Further, attackers will doubtless adjust their antidetection countermeasures to any techniques widely adopted by defenders.

Antivirus Scanning

Some experts recommend scanning all third-party software with antivirus prior to installation. Although that is certainly a best practice, it is unlikely to deliver meaningful results. Most software is installed through an installer package such as an **`.msi`** file. The **`.msi`** file (or other installer) contains large numbers of files that are unpacked and installed on the system. Scanning the installer file is unlikely to detect an infected file, even if the antivirus has a signature for it.

Best practices require installing the software first, then scanning the resulting installation. Even this approach is lacking, however. The scan is an analysis of the software at a particular point in time. Many software packages apply automatic updates. Even when these are disabled, so-called "configuration updates" may include executable code. An example of this would be an antivirus signature engine itself, which often utilizes scripting languages capable of executing code.

The larger issue with this approach is that the InfoSec industry, in general, understands that signature-based antivirus is largely capable of detecting only known threats. Any backdoor inserted through a software supply chain compromise would be unknown code for which signatures would not yet exist. Additionally, some scanners ignore or use different heuristics for digitally signed code, further reducing the efficacy of this technique.

The Verdict

Analysis of recent software supply chain attacks suggests that this method is not likely to be effective. The backdoors in **`MeDoc`** and SolarWinds were not detected through antivirus scanning. Additionally, antivirus scanning did not discover backdoors used in **`CCleaner`** or the **`ShadowPad/ShadowHammer`**[9] attacks. The appeal of this technique is obvious, but it is likely performative security with no realistic chance of detecting a compromise.

---

[8] "The Untold Story of NotPetya, the Most Devastating Cyberattack in History," www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world

[9] "Operation ShadowHammer: A High-Profile Supply Chain Attack," https://securelist.com/operation-shadowhammer-a-high-profile-supply-chain-attack/90380

Third-Party Risk Assessments

Some InfoSec practitioners have suggested using third-party risk assessments to identify which of their vendors are most likely to be compromised and used to deliver a software supply chain attack. The theory is not that risk assessments will prevent software supply chain risks, but rather that they will eliminate vendors that present the most risk. The organization uses the responses to questionnaires it sends to the vendor combined with publicly available data to determine whether the vendor has at least the same security posture the organization does.

The value of the questionnaire data depends on the number and quality of questions included. Unfortunately, increasing the value of questionnaire data increases the burden on the vendor, reducing the likelihood of compliance. Further, the questionnaires are typically just attestations—and vendors have little incentive to highlight poor security practices.

The Verdict

Although third-party risk assessments are a critical component of an information security program, we should not anticipate that they will be useful in detecting software supply chain compromises. SolarWinds counted the U.S. Department of Defense as a customer, which has more stringent vendor risk management requirements than the vast majority of organizations. In the case of **MeDoc**, attackers targeted a software package that the Ukrainian government required for use. In both instances, risk assessments would have been minimally useful.

When evaluating a threat, consider the intersection of intent, opportunity, and capability.

Any publicly available data used in the risk assessment is also unlikely to be a predictor of software supply chain threats. While some note that there were publicly visible signs of security lapses at SolarWinds before the software supply chain compromise, these rely on hindsight bias. There are myriad such warning signs observed every year, few of which ever materialize as supply chain compromises.

Techniques for Detecting Supply Chain Attacks

In the previous section, we laid out a compelling case that there is little hope of preventing a supply chain compromise. While this pronouncement sounds negative, there are definitely opportunities for detecting a software supply chain attack that has breached your network. Although there are no trivial approaches to obtaining 100% detection coverage, following the methods laid out in this section will provide detection for most actions taken after a supply chain attack begins.

When evaluating a threat, consider the intersection of intent, opportunity, and capability. The *opportunity* in a software supply chain attack is the installation of the software in your network. As discussed in the previous section, installation is probably not preventable. As evidenced by the level of detail put into the **SUNSPOT**-malware-modified SolarWinds build process, attackers targeting software supply chains value stealth and are capable of building sophisticated software to evade pedestrian detections.

In light of these facts, we must consider the *intent* of the attackers. Software supply chain compromises are a means to an end, not the end itself. To be useful to the attacker, the intrusion enabled by the software supply chain attack must help the attackers meet their goals (as shown in Figure 3).

Figure 3. Attacker Goals in a Supply Chain Attack

To accomplish any of the suggested goals, it is likely that the attacker will need to move laterally. Publicly available case studies of software supply chain attacks attest to this.

In the case of **NotPetya**, attackers performed a destructive cyberattack that was combined with the use of the wormable exploit **EternalBlue**.[10] Because many systems have been patched for the vulnerability that **EternalBlue** targets using MS 17-010, the payload also used credential dumping tools to obtain valid credentials that could be used to target other systems. Armed with those credentials, **NotPetya** used WMI and **PSEXEC** to perform lateral movement prior to encrypting machines.

In the SolariGate intrusion, where attackers performed follow-on activities, it is clear these activities included lateral movement. Observed lateral movement techniques were mostly WMI, but there are also reports of **PowerShell** remoting (where configured).

The other technique to focus on for detecting a software supply chain attack is command and control (C2). Both SolariGate and **NotPetya** used beaconing for command and control.

Because lateral movement and C2 are common techniques used in supply chain attacks, detecting them is critical. This section addresses the following methods for detecting software supply chain attacks through lateral movement and C2 detection:

- Configure event logging
- Use **Sysmon**
- Enable collection of East/West NetFlow
- Perform DNS logging
- Deploy network security monitoring (NSM) tools

Note that for each technique, organizations can purchase tools that will assist in data collection and analysis, but each requires knowledge about your environment to be maximally effective.

[10] https://en.wikipedia.org/wiki/EternalBlue

Configuring Event Logging

Let's face it: Windows event logging is woefully inadequate in an out-of-the-box configuration. To detect lateral movement, organizations should enable additional event logging. This section could be an entire paper on its own if all event logs were addressed. Note that many event logs useful in detecting lateral movement are not covered in this section. We've omitted them because they are either high volume or have too much noise relative to the signal that they provide.

SANS recommends enabling process auditing with command line arguments. This tactic creates logs with **`Event ID 4688`**. Additionally, organizations should enable share access auditing (but not detailed share access auditing) on most workstations and servers. Share access auditing logs a single **`Event ID 5140`** the first time a given file share is accessed in a new SMB session. Detailed share access auditing is a less attractive option because it logs every file operation for every file accessed, creating far too much noise in the event logs.

Organizations should also increase the size of the event logs on the system, ensuring that local systems have longer log retention. An attacker could clear event logs on a system, but the "deleted logs" are still easily visible by forensics tools. Ideally, organizations will use Windows Event Forwarding to send relevant event logs to a centralized location. Too many organizations take an all-or-nothing approach to event forwarding. By centralizing only select event logs, however, organizations obtain the benefit of increased visibility without overloading the centralized collection server.

In addition to the Security event log, other event logs are may be useful in detecting lateral movement, including System, WMI, Terminal Services, PowerShell, and Task Scheduler. Each of these logs can provide significant additional information, but they require investigators to sift through more noise to find the signal.

Use Sysmon

The free Microsoft utility **`Sysmon`** can be used to gather additional logging information that is not available using standard logging. Unfortunately, **`Sysmon`** has a reputation for generating ridiculous volumes of logs. While the reputation is well-earned, **`Sysmon`** supports filters to control what is logged. The most useful log events in this category for our purposes include:

- **`WmiEventConsumerToFilter`** (binding WMI consumers to events)
- DNS events (if not captured elsewhere)
- Driver load (for unexpected drivers)

Enable Collection of East/West NetFlow

NetFlow is a rich source of data for understanding what is communicating on the network. Where NetFlow collection is enabled, it is usually limited to the ingress and egress of the network (North/South). However, NetFlow inside the network (East/West) is potentially more useful when investigating a software supply chain attack because any attempts to map an internal network or move laterally within the network are not visible in North/South NetFlow but should be immediately apparent in East/West NetFlow. In the SolariGate compromise, analyzing East/West NetFlow to discover lateral movement would have been difficult (Orion servers are extremely chatty under normal conditions). In most circumstances, East/West NetFlow is a huge enabler in hunting lateral movement. In many cases lateral movement can't be readily discovered without it. In addition to identifying lateral movement, NetFlow (usually North/South) is useful in identifying C2 actions.

Perform DNS Logging

Centralized **DNS** logs are critical for detecting C2 that depends on **DNS**, as most does. For C2 that depends on Direct-to-IP Connections (**DIC**s), NetFlow has and should be used to detect the connection. Unfortunately, NetFlow is not useful for tracking connections made via **DNS**. While the connection to the returned IP will be captured in NetFlow, the domain requested will not be. This juncture is where **DNS** logging comes into play.

Domain names are often discovered as indicators of compromise (IoCs) and are used for hunting after an intrusion is discovered. Unfortunately, many organizations do not log all **DNS** requests and responses because many **DNS** servers do not offer easy logging configurations. Even when **DNS** servers are configured for logging, the data captured may be not be ideal due to **NAT** gateways obscuring the system making the request.

When logging on the **DNS** server itself cannot be easily configured, **Sysmon** (discussed earlier) and network security monitoring (NSM, discussed next) can cover the visibility gap.

Deploy Network Security Monitoring Tools

NSM systems bridge the gap between NetFlow and full packet capture (**PCAP**). While full **PCAP** provides the most granular data, it is usually impractical to capture (and query) at scale. This is particularly true for internal collection points (think East/West NetFlow). An NSM processes some higher-level protocol details without capturing content, which is extremely beneficial for long-term retention. NSM can also be useful in situations where regulatory, contractual, or other privacy concerns prohibit logging full packet content (as with **PCAP**).

Of the protocol decoding possible with NSM, the most useful are **HTTP** and **DNS**. Whereas most **HTTP** is encrypted (such as **HTTPS**), many NSM platforms are deployed in locations where another device is already terminating the **TLS** connection and then supplying the NSM with decrypted data. Even in situations where that is not the case, NSM can parse unencrypted details of certificates as well as server name identification (SNI, **TLS 1.2** and earlier). The **DNS** parsing capabilities of the NSM can also be used to log **DNS** requests and responses where logs cannot be obtained from the local DNS server for whatever reason. While **DoH** (**DNS** over **HTTPS**) complicates logging of **DNS**, it should generally be blocked at the egress firewall for most organizations and does not warrant further consideration.

Conclusion

The SolarWinds software supply chain compromise has rightfully attracted a lot of attention from the security community. Vendors undoubtedly will be coming out of the woodwork to offer solutions to ***prevent*** further software supply chain attacks. As we've outlined in this paper, the reality is that most of these won't meaningfully prevent another software supply chain attack. However, it's not all bad news. As we also discussed in this paper, there are steps organizations can take to ***detect*** future software supply chain attacks. The better news is that, as opposed to being unicorn solutions, these activities will make the detection of many attacks easier, substantially increasing security posture.

About the Author

Jake Williams is a SANS analyst, senior SANS instructor, course author, and designer of several NetWars challenges for use in SANS' popular, "gamified" information security training suite. Jake spent more than a decade in information security roles at several government agencies, developing specialties in offensive forensics, malware development, and digital counterespionage. Jake is the founder of Rendition InfoSec, which provides penetration testing, digital forensics and incident response, expertise in cloud data exfiltration, and the tools and guidance to secure client data against sophisticated, persistent attacks on-premises and in the cloud.