Sunburst Malware: Inside the SolarWinds Supply Chain Breach

Published: 2025-10-22 · Archived: 2026-04-05 19:42:25 UTC

When cybersecurity historians look back at the 2020s, the **SolarWinds breach** — driven by the **Sunburst backdoor** — will stand out as the moment the world realized how fragile software trust chains can be.

Attackers didn't just compromise one network; they turned a trusted update mechanism into a global delivery system for espionage.

As someone who has worked in **cloud forensics and supply chain defense**, I often describe Sunburst as *"a symphony of silence"* — precise, patient, and devastatingly effective.

The Trojan Inside a Trusted Update

The attack began when adversaries successfully **injected malicious code** into **SolarWinds Orion**, a widely used IT monitoring software.
This wasn't a rushed exploit; it was a carefully timed insertion inside an otherwise legitimate DLL — SolarWinds.Orion.Core.BusinessLayer.dll.

That DLL, signed with SolarWinds' valid certificate, was distributed via routine software updates.
When administrators installed the patch named SolarWinds-Core-v2019.4.5220-Hotfix5.msp, they unknowingly deployed a backdoor into their own systems.

The malware was not obfuscated; it was written in **.NET**, easily decompiled by tools like ILSpy.
This allowed researchers to reconstruct and analyze it line by line.
Once loaded, it executed quietly within the **legitimate process** SolarWinds.BusinessLayerHost.exe, blending into the software's normal behavior.

How the Backdoor Stayed Hidden

Sunburst's stealth was rooted in timing and precision.
After being loaded, the malicious code waited between **12 to 14 days** before activating.
This random delay, ranging from 288 to 336 hours, was a clever tactic — delaying detection by security teams and automated scanners.

The malware maintained an internal **reporting state** called ReportWatcherRetry, which determined its next action. If the state was "New," it began communication; if "Truncate," it shut down network operations and disabled defenses.

To appear inactive, it would "sleep" for random intervals between **30 minutes and 2 hours**.
In Hindi, one could say *"yeh malware sota bhi hai, par sapne nahi dekhta"* — it sleeps, but it never stops watching.

The Identity Behind the Attack: User ID Generation

Each infected system generated a unique **8-byte User ID** derived from:

1. The machine's **MAC address**,
2. Its **domain name**, and
3. A **Windows installation UUID**.

The result looked random but stayed consistent unless the OS or network hardware changed.
This permanent fingerprint allowed attackers to track individual hosts without obvious identifiers — a masterclass in covert telemetry.

Domain Generation and Command Control

Sunburst's **CryptoHelper class** produced domain names dynamically, blending cryptography with randomness.
The process began by querying api.solarwinds.com. If resolution failed, the malware went dormant — another

layer of self-preservation.

Domains were created using a **custom base64 alphabet** (ph2eifo3n5utg1j8d94qrvbmk0sal76c), combined with encoded host data.
A generated domain could look like:

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

This structure resembled legitimate AWS API endpoints — masking malicious traffic among genuine cloud requests.

Example of Generated Domains

Example Hostname	C2 Region	Mimicked Service
fivu4vjamve5vfrtn2huov.appsync-api.eu-west-1.avsvmcloud.com	EU	AWS AppSync
k1sdhtslulgqoagyn2huov.appsync-api.us-east-1.avsvmcloud.com	US East	AWS AppSync

Each new domain resolved to an IP address. If the IP matched specific subnet patterns, the malware interpreted it as a coded instruction — to activate, sleep, or disable tools.

Adaptive Behavior: The Address Family Logic

Sunburst assigned “address families” based on IP subnets.
For example, IPs like 20.140.0.1 mapped to the **ImpLink** family, triggering specific routines.
Addresses within private ranges (10.x.x.x or 192.168.x.x) were marked as **Atm**, used for stealth actions.

This subtle design allowed the attackers to **control malware behavior remotely** through DNS alone — a technique rarely seen in earlier backdoors.

Disabling Defenses: Privilege Escalation and Registry Manipulation

When Sunburst determined it was safe to proceed, it elevated privileges using Windows APIs:

AdjustTokenPrivileges() granted **SeRestorePrivilege** and **SeTakeOwnershipPrivilege**, enabling full control over protected services.

It then enumerated local user accounts via **WMI queries** (Select * From Win32_UserAccount) to locate the **Administrator SID** (ending with -500).
Once found, it reassigned ownership of antivirus registry keys, disabling services like **Windows Defender** by setting their Start values to 4 — effectively turning them off.

This operation was powered by the **Fowler–Noll–Vo hash algorithm**, which compared hashed process names to a predefined blacklist.
Even the hash for MsMpEng (Windows Defender's core process) — 5183687599225757871 — was explicitly embedded in the malware.

HTTP Backdoor and System Reconnaissance

If the domain resolution returned a "NetBios" family IP, Sunburst initialized its **HTTP backdoor**, communicating through encrypted HTTPS requests.

Commands like **CollectSystemDescription** gathered exhaustive system data:

- Domain and username
- OS version
- Proxy configurations
- Network adapters and IPs

A sample output sent to attackers might look like:

DESKTOP-VL39FPO | UserName | Windows 10 64-bit | 192.168.20.30 | 8.8.8.8 | DHCP: True

It also supported commands like **GetProcessByDescription**, listing processes with paths and parent PIDs — a reconnaissance method to map active defenses.

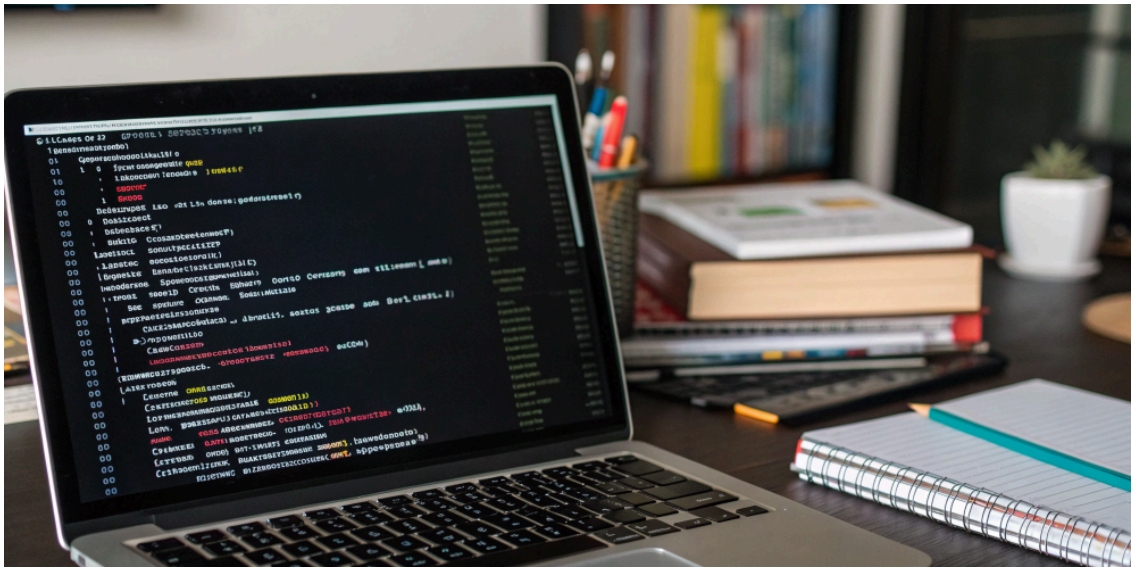
Staging the Second Attack

Sunburst wasn't the final payload.
It included commands like **WriteFile** and **RunTask**, enabling attackers to deploy and execute **second-stage malware** directly.
These payloads were often tailored for specific environments, extending control to email servers, identity platforms, and domain controllers.

If persistence was needed, registry manipulation ensured the payload would re-execute after reboot.

In short, Sunburst was the silent courier — **the delivery boy of a much larger operation**.

Lessons from Sunburst: Supply Chain and Cloud Vigilance



The Sunburst breach wasn't a simple intrusion — it was a **paradigm shift**.
By compromising a software vendor's build system, attackers reached **thousands of organizations** without breaking into a single firewall.
This proved that **trust is the new attack surface**.

Today, modern security frameworks like **Zero Trust Architecture** and **Software Bill of Materials (SBOM)** have emerged in direct response.
Vendors now integrate **code-signing validation**, **runtime monitoring**, and **DevSecOps pipelines** that scan build environments for anomalies.

As I often tell my students: *"Yeh incident ne sikhaya — verification bina trust nahi"* (This incident taught us that without verification, trust means nothing).

The Future: AI and Supply Chain Defense

AI-driven analytics now allow defenders to trace subtle behavioral shifts across millions of endpoints — something manual review could never achieve.
By mapping communication graphs and DGA patterns, machine learning can now flag anomalies before they turn into breaches.

But technology alone isn’t enough.
True resilience requires cultural change: **transparency in vendor ecosystems** and **collaboration across global CERTs**.
