SolarWinds: a case study of Software Supply Chain attacks

Team 6: Ashlesha Kishor Mahajan, Eleonora Martella, Rosa Maria Nucera

ashleshakishor.mahajan01@universitadipavia.it

eleonora.martella01@universitadipavia.it

rosamaria.nucera02@universitadipavia.it

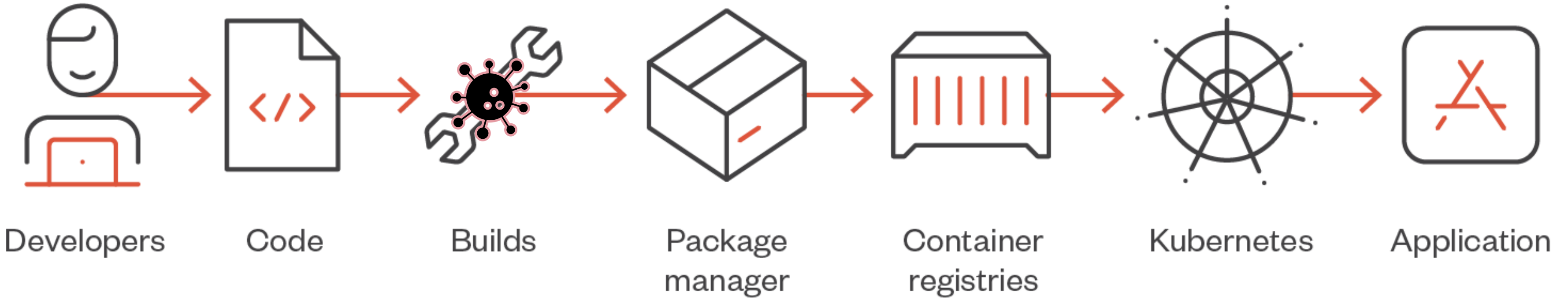
University of Pavia – Department of Electrical, Computer and Biomedical Engineering

Outline

- Software Supply Chain Overview
- SolarWinds attack overview
- Attack Analysis by Cyber Kill Chain® phases
- Impact of the attack
- Mitigation
- Conclusion

Context: Software Supply Chain Attacks

- **The Supply Chain**: damage one to hit them all



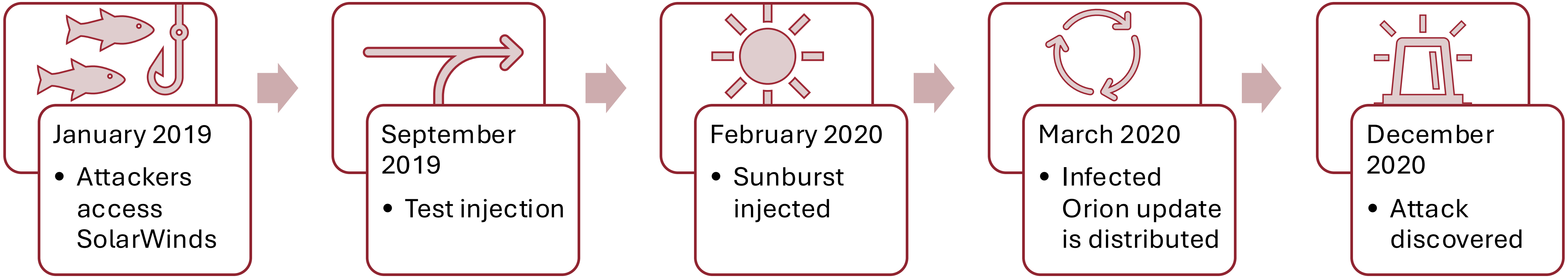
© MICRO TRENDS

- **Massive attack surface**
  - If one component is damaged, all products utilizing it will be damaged.
- Attackers use **trust exploitation** in order to deal huge amounts of damage
  - Tendency to trust reputable sources, or tools that have already been in use for a long time

The attack on solarwinds

- Texas-based company, for businesses - managing networks, systems, and IT infrastructures
- Victims being 18.000 - 33.000 companies, including **government agencies** and **major corporations**
- Perpetrated by government funded **Russian group APT29** → **Supplying chain attacks as political weapons**

Timeline:



January 2019
- Attackers access SolarWinds

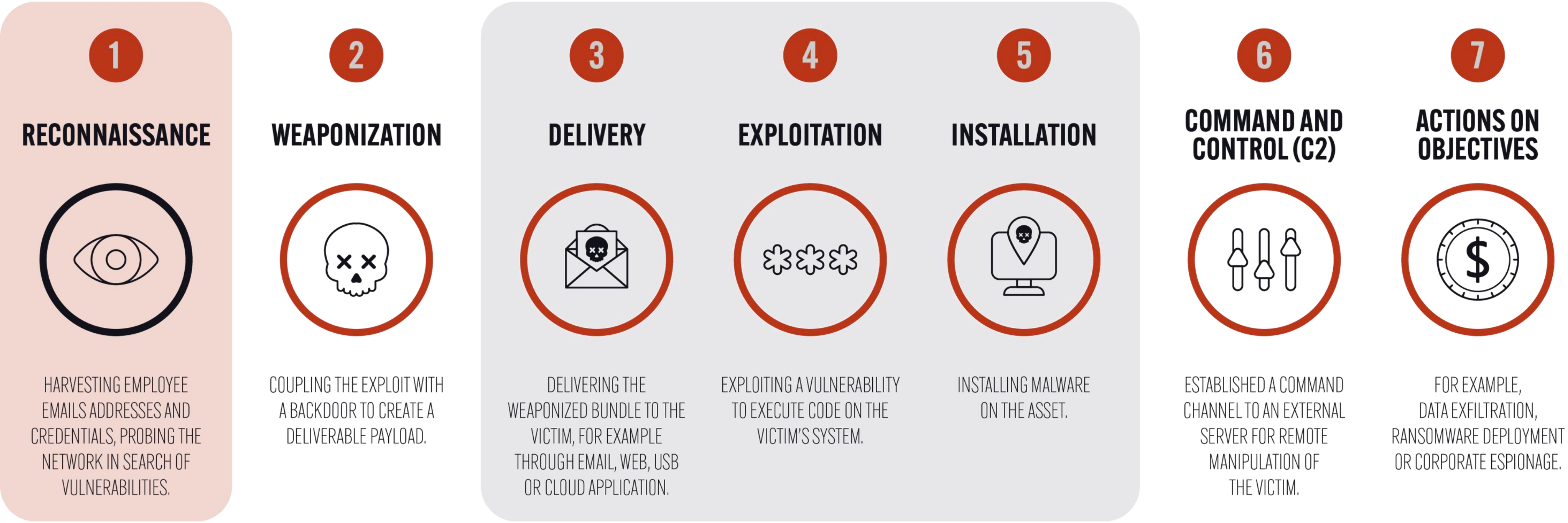
September 2019
- Test injection

February 2020
- Sunburst injected

March 2020
- Infected Orion update is distributed

December 2020
- Attack discovered

Attack Analysis: Cyber Kill Chain®



1 RECONNAISSANCE

HARVESTING EMPLOYEE EMAILS ADDRESSES AND CREDENTIALS, PROBING THE NETWORK IN SEARCH OF VULNERABILITIES.

2 WEAPONIZATION

COUPLING THE EXPLOIT WITH A BACKDOOR TO CREATE A DELIVERABLE PAYLOAD.

3 DELIVERY

DELIVERING THE WEAPONIZED BUNDLE TO THE VICTIM, FOR EXAMPLE THROUGH EMAIL, WEB, USB OR CLOUD APPLICATION.

4 EXPLOITATION

EXPLOITING A VULNERABILITY TO EXECUTE CODE ON THE VICTIM'S SYSTEM.

5 INSTALLATION

INSTALLING MALWARE ON THE ASSET.

6 COMMAND AND CONTROL (C2)

ESTABLISHED A COMMAND CHANNEL TO AN EXTERNAL SERVER FOR REMOTE MANIPULATION OF THE VICTIM.

7 ACTIONS ON OBJECTIVES

FOR EXAMPLE, DATA EXFILTRATION, RANSOMWARE DEPLOYMENT OR CORPORATE ESPIONAGE.

slcyber.io

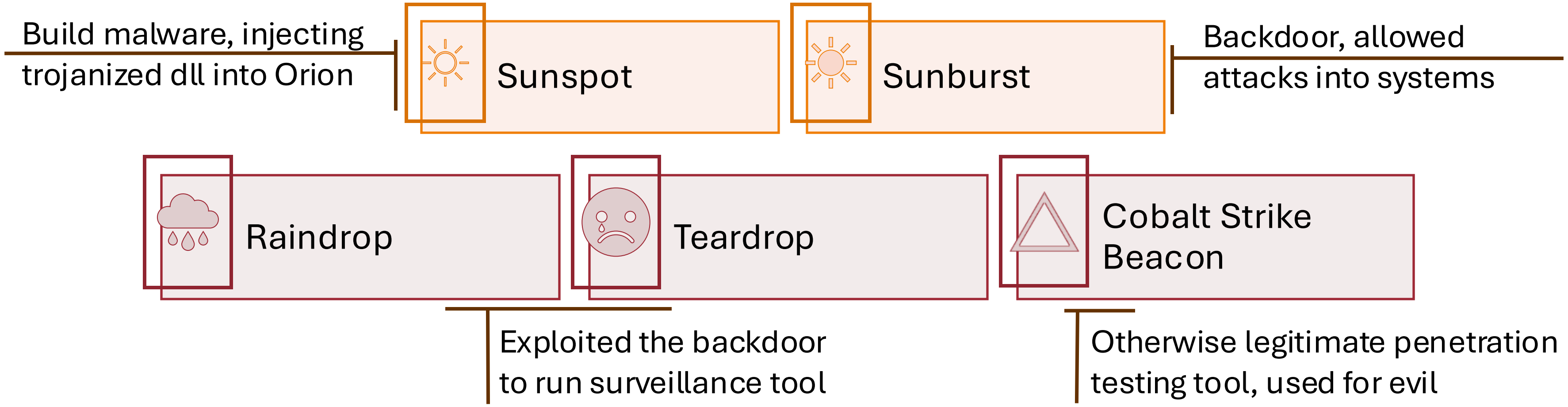
RECONNAISSANCE: **target identification - Orion software**

- **Target**: Orion software
- **Entrance method**: unknown, possibly:

Zero-day vulnerability in a **third-party application**	**Brute-force attack**, e.g. password spray attack
Social engineering, e.g. phishing attack	**Mail Server vulnerability** exploitation

- Compromising the **build server**
  - Months of passive monitoring and exfiltration

WEAPONIZATION: **Creating Malicious Payload**



Build malware, injecting trojanized dll into Orion

Sunspot

Sunburst

Backdoor, allowed attacks into systems

Raindrop

Teardrop

Cobalt Strike Beacon

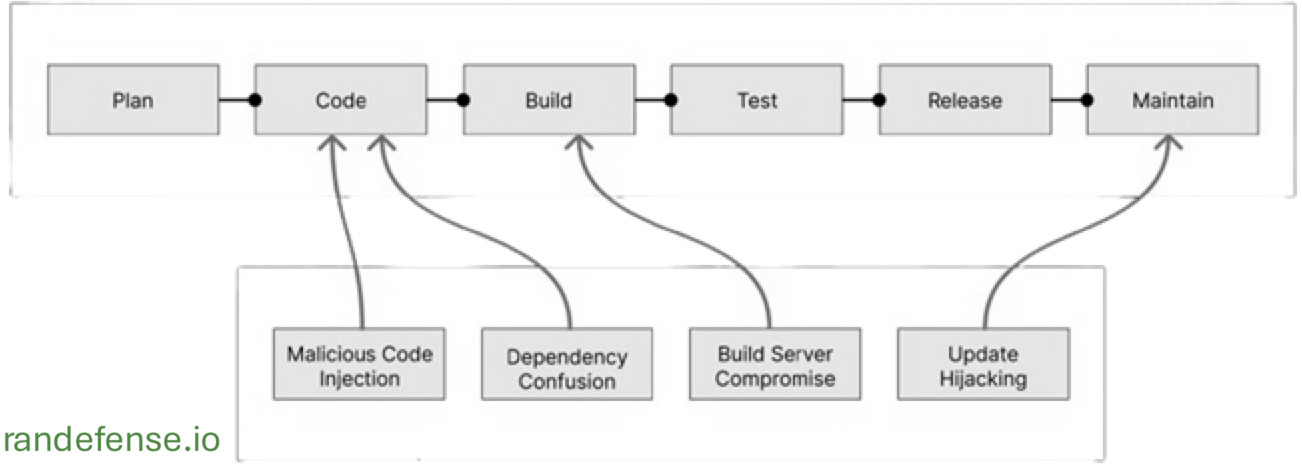
Exploited the backdoor to run surveillance tool

Otherwise legitimate penetration testing tool, used for evil

Delivery: **Malware injection – Proof of Concept**

- **Test run**: injection of innocuous test code
  - Attackers verify that their plan is doable: code appears in the following update

- **Update hijacking:** *Attackers break into a vendor’s network and inject malware into a legitimate patch. Customers automatically download and install the “update,” giving the attackers privileged access.* (brandefense.io)



brandefense.io

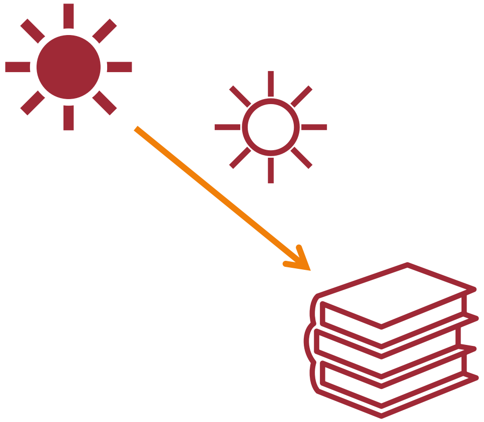
- Discoveries:

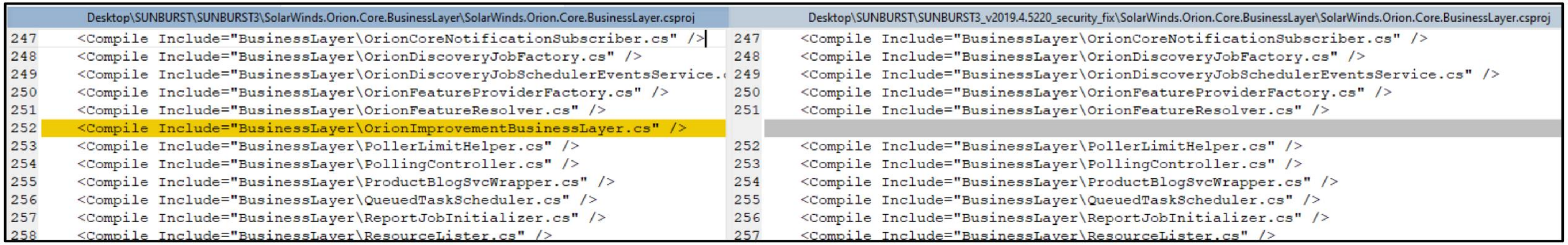
Automated build environment	Digitally signed with **legitimate certificates**
No human intervention	Exploits customers’ trust

DELIVERY: **malware injection - Sunspot**

Sunspot

- **Persistent:**
  - Scheduled task set to execute when the host boots
- **Discreet:**
  - Living in build environment, normally dormant
- **Attacks the build:**
  - Detects when a *Visual Studio* build is happening
    - Checks if it’s the Orion build
    - Checks if injection is already occurring: if yes, exit process
  - Gives itself debugging privileges → can read other processes’ memory
  - Sunburst injection: Replacing legitimate dll library with compromised clone
    - **Dependency Hijacking !**
    - Swap *during* the build → **attack against the build process !**





From CSIRT Sunburst Threat Report.

Desktop\SUNBURST\SUNBURST3\SolarWinds.Orion.Core.BusinessLayer\SolarWinds.Orion.Core.BusinessLayer.csproj

```
    <Compile Include="BusinessLayer\OrionCoreNotificationSubscriber.cs" />
    <Compile Include="BusinessLayer\OrionDiscoveryJobFactory.cs" />
    <Compile Include="BusinessLayer\OrionDiscoveryJobSchedulerEventsService.
    <Compile Include="BusinessLayer\OrionFeatureProviderFactory.cs" />
    <Compile Include="BusinessLayer\OrionFeatureResolver.cs" />
    <Compile Include="BusinessLayer\OrionImprovementBusinessLayer.cs" />
    <Compile Include="BusinessLayer\PollerLimitHelper.cs" />
    <Compile Include="BusinessLayer\PollingController.cs" />
    <Compile Include="BusinessLayer\ProductBlogSvcWrapper.cs" />
    <Compile Include="BusinessLayer\QueuedTaskScheduler.cs" />
    <Compile Include="BusinessLayer\ReportJobInitializer.cs" />
    <Compile Include="BusinessLayer\ResourceLister.cs" />
```

Desktop\SUNBURST\SUNBURST3_v2019.4.5220_security_fix\SolarWinds.Orion.Core.BusinessLayer\SolarWinds.Orion.Core.BusinessLayer.csproj

```
    <Compile Include="BusinessLayer\OrionCoreNotificationSubscriber.cs" />
    <Compile Include="BusinessLayer\OrionDiscoveryJobFactory.cs" />
    <Compile Include="BusinessLayer\OrionDiscoveryJobSchedulerEventsService.cs" />
    <Compile Include="BusinessLayer\OrionFeatureProviderFactory.cs" />
    <Compile Include="BusinessLayer\OrionFeatureResolver.cs" />

    <Compile Include="BusinessLayer\PollerLimitHelper.cs" />
    <Compile Include="BusinessLayer\PollingController.cs" />
    <Compile Include="BusinessLayer\ProductBlogSvcWrapper.cs" />
    <Compile Include="BusinessLayer\QueuedTaskScheduler.cs" />
    <Compile Include="BusinessLayer\ReportJobInitializer.cs" />
    <Compile Include="BusinessLayer\ResourceLister.cs" />
```

WHY THIS DLL? - SolarWinds.Orion.Core.BusinessLayer.dll

The choice of SolarWinds.Orion.Core.BusinessLayer.dll was strategic for different reason.

Loaded automatically at the startup of the Orion service.

Responsible for legitimate network operations.

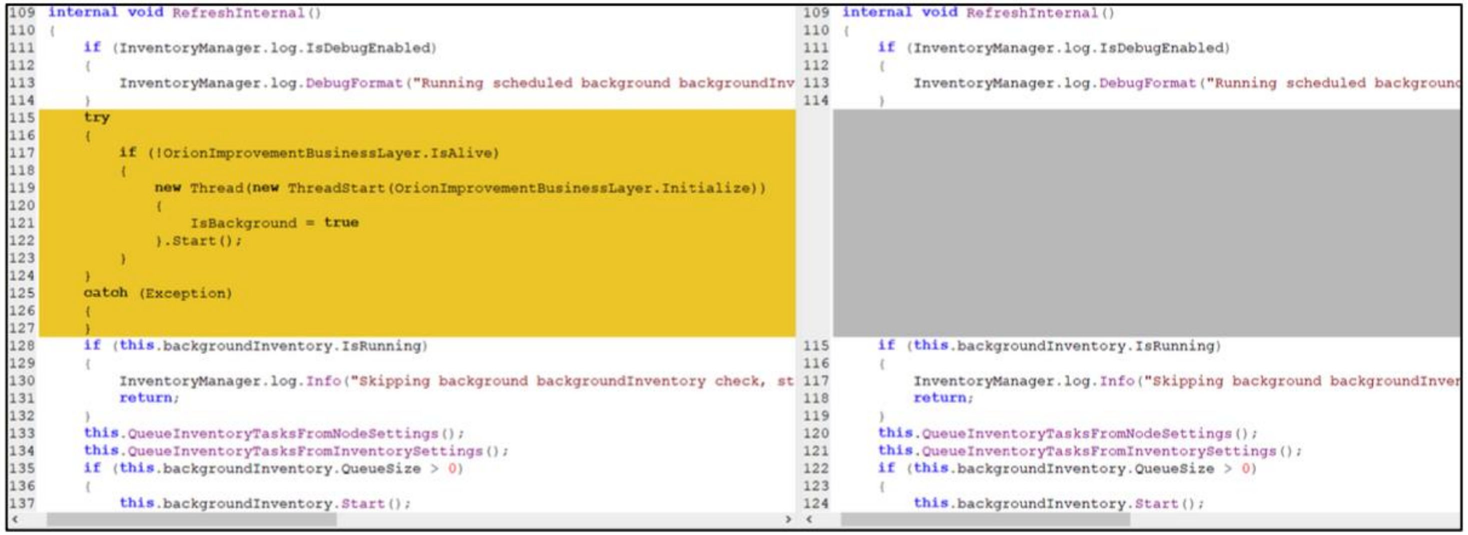
Executed with sufficient privileges to interact with the operating system.

This made it the perfect “carrier” for injecting malicious behaviour without raising suspicion.

Diffing between the legitimate DLL and the compromised one revealed two key modifications:

The attackers added a brand-new .NET class: OrionImprovementBusinessLayer containing the backdoor logic. **1**

They inserted roughly a **dozen lines of code** into an existing method **(RefreshInternal)** causing the legitimate workflow to silently spawn a thread running malicious code in the background. **2**



```
internal void RefreshInternal()
{
    if (InventoryManager.log.IsDebugEnabled)
    {
        InventoryManager.log.DebugFormat("Running scheduled background backgroundInv
    }
    try
    {
        if (!OrionImprovementBusinessLayer.IsAlive)
        {
            new Thread(new ThreadStart(OrionImprovementBusinessLayer.Initialize))
            {
                IsBackground = true
            }.Start();
        }
    }
    catch (Exception)
    {
    }
    if (this.backgroundInventory.IsRunning)
    {
        InventoryManager.log.Info("Skipping background backgroundInventory check, st
        return;
    }
    this.QueueInventoryTasksFromNodeSettings();
    this.QueueInventoryTasksFromInventorySettings();
    if (this.backgroundInventory.QueueSize > 0)
    {
        this.backgroundInventory.Start();
```

```
internal void RefreshInternal()
{
    if (InventoryManager.log.IsDebugEnabled)
    {
        InventoryManager.log.DebugFormat("Running scheduled backgroun
    }
    if (this.backgroundInventory.IsRunning)
    {
        InventoryManager.log.Info("Skipping background backgroundInve
        return;
    }
    this.QueueInventoryTasksFromNodeSettings();
    this.QueueInventoryTasksFromInventorySettings();
    if (this.backgroundInventory.QueueSize > 0)
    {
        this.backgroundInventory.Start();
```

Figure 1-On the right is the original version of the Refresh Internal method, on the left is the version modified by the attacker by adding 13 lines of code. From CSIRT Sunburst Threat Report.

WHY THIS DLL? – Exploitation phase

UNIVERSITÀ DI PAVIA

Because the modified DLL preserved its expected behaviour and network patterns, outbound HTTPS traffic to servers seems like **SolarWinds infrastructure appeared completely normal**.

The update distributed to customers therefore **contained a modular backdoor, SUNBURST**, capable of:

⌘ **RECEIVING COMMANDS** FROM THE C2 SERVER

MODIFYING CONFIGURATION VALUES

DOWNLOADING ADDITIONAL COMPONENTS

SHUTTING ITSELF DOWN TO AVOID DETECTION

This exploitation phase exploited a TRUSTED NETWORK CHANNEL

SUNBURST – **Installation phase**

ONCE DEPLOYED, SUNBURST **DOES NOT ACTIVATE IMMEDIATELY**

Instead, it enters a carefully engineered initialization phase designed to evade sandbox analysis and trigger only in high-value environments.

When the compromised DLL is loaded, Sunburst executes the **Initialize method**, which:

- **Creates a local kill-switch** through a Windows named pipe
- **Retrieves configuration parameters**
- **Generates a unique machine**

After initialization, **the malware enters the UPDATE METHOD**

SUNBURST – **Dormancy and Update method**

Sunburst compares running processes, services, and drivers against an embedded blocklist:

If a security tool **IS DETECTED**
the malware shuts down immediately

If a relevant service is present
it attempts to disable it by **MANIPULATING** the Windows Registry

Only after both Initialize and Update succeed does Sunburst
ENTER ITS 10–16 DAY DORMANCY PERIOD

THIS DELAY IS HIGHLY STRATEGIC:
automated sandbox systems typically observe behaviour for only minutes or hours, meaning Sunburst **appears perfectly benign during initial analysis**

SUNBURST – **Environment Checks**

Sunburst performs a second round of validation:

Checks for the presence of an Active Directory domain

Verifies that it was loaded by the legitimate Orion process using a proprietary hash

Re-scans the environment against its blocklist

IF ANY OF THESE CHECKS **FAIL**

Example: on isolated or home systems, or in a sandbox **THE MALWARE REMAINS INACTIVE OR TERMINATES**

This selective activation minimizes unnecessary infections

drastically reduces the chance of detection

FOCUSING THE ATTACK ON INTERESTING VICTIM.

C2 BOOTSTRAPPING: DNS STRATEGY

The **command-and-control architecture** of Sunburst is one of the most advanced aspects of the operation
The malware always starts from the fixed domain **avsvmcloud.com**, used as a dispatcher

For **each execution**, **it generates a unique subdomain** through a DGA based on: | **Internal user ID** | **Victim's Windows domain**

These strings are **appended avsvmcloud.com** such as:

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

For each generated domain, **Sunburst performs two DNS queries:**

A RECORD | **CNAME RECORD**

Before activating, Sunburst **contacts the legitimate domain api.solarwinds.com**

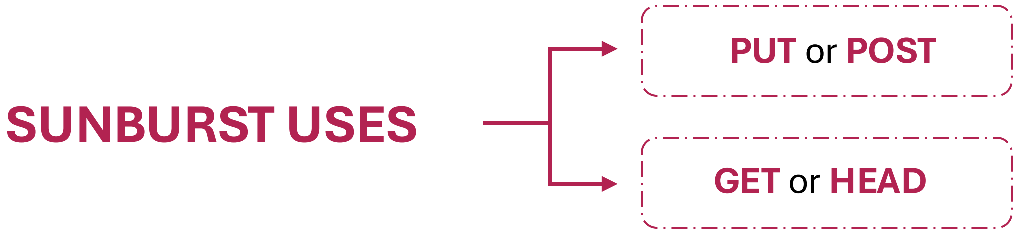
DNS KILL-SWITCH

If the IP from the A record falls within a blocked subnet **Sunburst immediately terminates**

If the IP is acceptable, the CNAME is considered valid and the C2 channel **IS ACTIVATED**.
Attackers further strengthened stealth by using IPs geolocated in the victim's own country, **defeating "impossible travel" anomaly detection**

C2 INFRASTRUCTURE: Communication Loop

Once the final C2 domain **is determined**, Sunburst launches a dedicated communication thread via **HttpHelper.Initialize**


SUNBURST USES

PUT or **POST**

GET or **HEAD**

To further hide identification data, the bot ID is embedded inside the If-None-Match header

The **C2 executes remote commands** which include:

- **Enumerating directories**
- **Modifying Registry keys**
- **Writing** or **deleting files**
- **Spawning processes**

THE COMMUNICATION INTERVAL ITSELF CAN BE ALTERED DYNAMICALLY
THROUGH THE **SETTIME COMMAND**

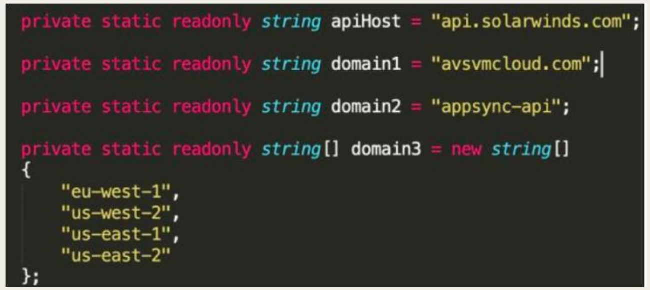
STEALTH ARCHITECTURE - Obfuscation and Digital Signature Abuse

Sunburst **integrates highly sophisticated evasion** and **anti-forensics techniques**, engineered as core components of the attack **to consistently hide the backdoor and exploit inherent weaknesses** in traditional security tools, allowing the malware **to remain undetected for months**

1

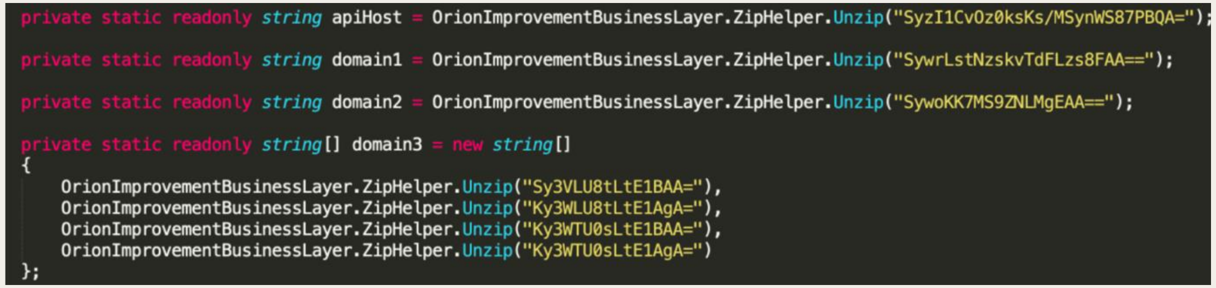
SELECTIVE OBFUSCATION AND CUSTOM HASHING

The attackers **deliberately avoided full .NET code** obfuscation In parallel, Sunburst employs a **custom hash function to verify process and driver names**



```
private static readonly string apiHost = "api.solarwinds.com";

private static readonly string domain1 = "avsvmcloud.com";

private static readonly string domain2 = "appsync-api";

private static readonly string[] domain3 = new string[]
{
    "eu-west-1",
    "us-west-2",
    "us-east-1",
    "us-east-2"
};
```

Figure 2 -Figure 3 deobfuscated strings.



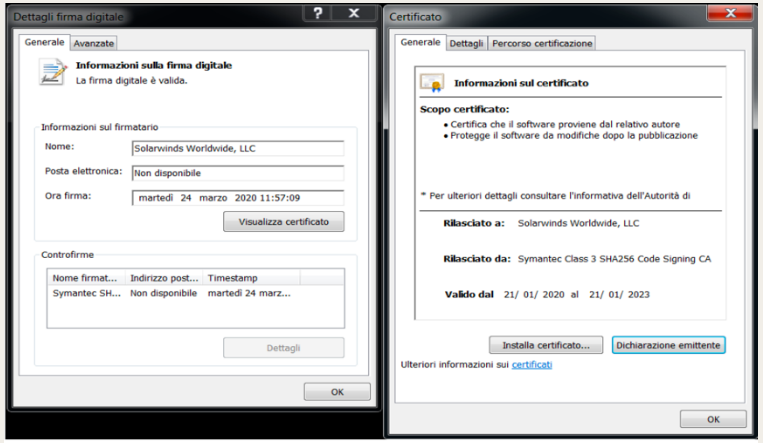
```
private static readonly string apiHost = OrionImprovementBusinessLayer.ZipHelper.Unzip("SyzI1CvOz0ksKs/MSynWS87PBQA=");

private static readonly string domain1 = OrionImprovementBusinessLayer.ZipHelper.Unzip("SywrLstNzskvTdFLzs8FAA==");

private static readonly string domain2 = OrionImprovementBusinessLayer.ZipHelper.Unzip("SywoKK7MS9ZNLMgEAA==");

private static readonly string[] domain3 = new string[]
{
    OrionImprovementBusinessLayer.ZipHelper.Unzip("Sy3VLU8tLtE1BAA="),
    OrionImprovementBusinessLayer.ZipHelper.Unzip("Ky3WLU8tLtE1AgA="),
    OrionImprovementBusinessLayer.ZipHelper.Unzip("Ky3WTU0sLtE1BAA="),
    OrionImprovementBusinessLayer.ZipHelper.Unzip("Ky3WTU0sLtE1AgA=")
};
```

Figure 3 -Obfuscated strings used by the DGA. From CSIRT Sunburst Threat Report.

2

ABUSE OF DIGITAL SIGNATURES

The malicious payload was **inserted before the SolarWinds update was digitally signed.**
The compromised package appeared authentic and trusted



Dettagli firma digitale

Generale | Avanzate

Informazioni sulla firma digitale
La firma digitale è valida.

Informazioni sul firmatario

Nome: Solarwinds Worldwide, LLC
Posta elettronica: Non disponibile
Ora firma: martedì 24 marzo 2020 11:57:09

Controfirme

Nome firmat...	Indirizzo post...	Timestamp
Symantec SH...	Non disponibile	martedì 24 marz...

Certificato

Generale | Dettagli | Percorso certificazione

Informazioni sul certificato

Scopo certificato:
- Certifica che il software proviene dal relativo autore
- Protegge il software da modifiche dopo la pubblicazione

* Per ulteriori dettagli consultare l'informativa dell'Autorità di

Rilasciato a: Solarwinds Worldwide, LLC

Rilasciato da: Symantec Class 3 SHA256 Code Signing CA

Valido dal 21/ 01/ 2020 al 21/ 01/ 2023

Ulteriori informazioni sui certificati

Figure 4 – The legitimate certificate with which the infected update packages were signed. From CSIRT Sunburst Threat Report.

STEALTH ARCHITECTURE - Evasion, Anti-Forensics, and Detection Failure

3 DORMANCY, DELAYS, AND ANTI-SANDBOX LOGIC

Sunburst remains dormant for **10/16 days** and introduces additional **long, randomized sleep cycles during execution**

6 PERFECTLY CAMOUFLAGED C2 TRAFFIC

Command-and-control communication uses:

- **HTTPS**
- **Valid AWS TLS certificates**
- **DNS with rotating subdomains**
- **Use of trusted cloud platform**
- **Patterns that resemble normal SolarWinds Orion telemetry**

4 FILE-LESS EXECUTION AND MINIMAL ARTIFACTS

Follow-on payloads execute **entirely in memory** via injection into **legitimate processes**. This results in:

- **no files on disk**
- **no detectable signatures**
- **no persistent indicators**

5 LIVING-OFF-THE-LAND AND OPERATIONAL CAMOUFLAGE

Once installed, Sunburst relies **exclusively on native Windows administrative tools** **(Living-Off-the-Land)**

7 TIMESTOMPING AND LOG MANIPULATION

Sunburst **manipulates timestamps**, **erases traces**, operates **in memory**, making **detection extremely difficult**

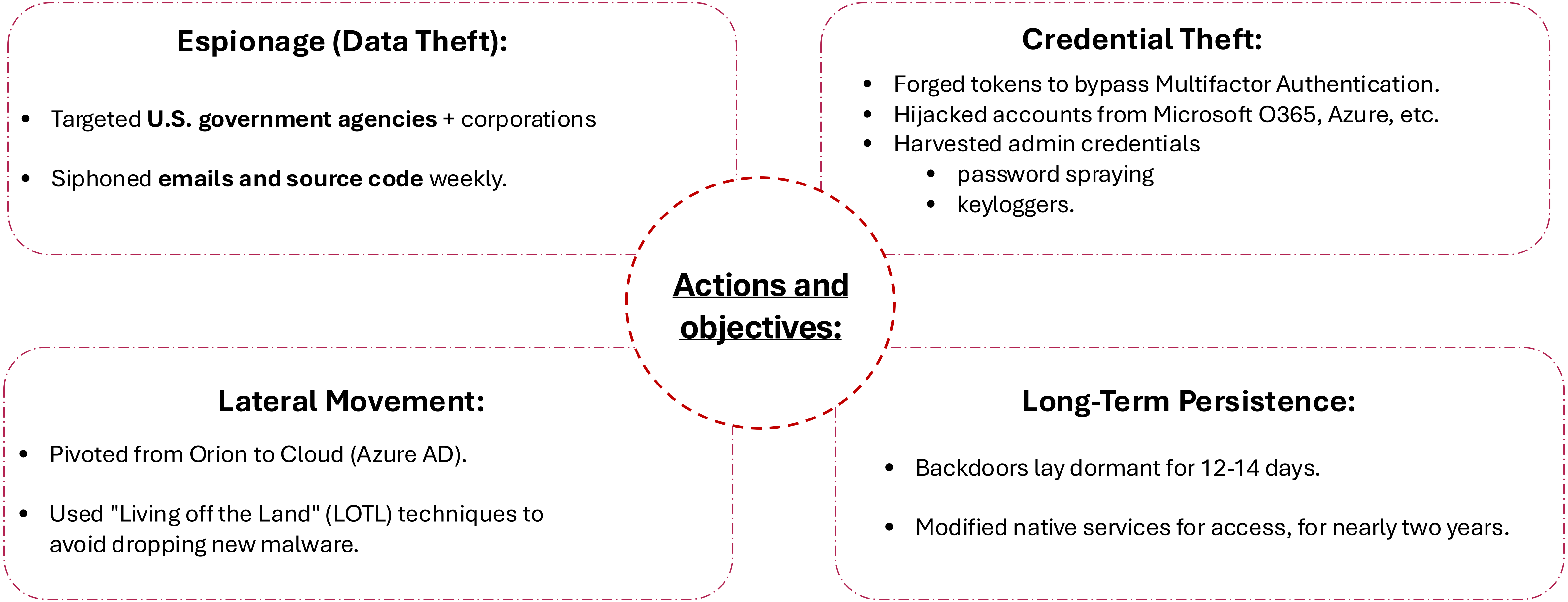
8 WHY ANTIVIRUS AND IDS TOOLS FAILED

All techniques **converge to create an operational environment**:

- **Binary appears signed and legitimate**
- **No artifacts exist on disk**
- **No malicious occurs for weeks**
- **All tools used are native to the OS**
- **Network traffic is indistinguishable from legitimate SolarWinds traffic**

Actions on Objectives & Post-Exploitation

❖ Strategic goals achieved by APT29.



Actions and objectives:

Espionage (Data Theft):

- Targeted **U.S. government agencies** + corporations
- Siphoned **emails and source code** weekly.

Credential Theft:

- Forged tokens to bypass Multifactor Authentication.
- Hijacked accounts from Microsoft O365, Azure, etc.
- Harvested admin credentials
  - password spraying
  - keyloggers.

Lateral Movement:

- Pivoted from Orion to Cloud (Azure AD).
- Used "Living off the Land" (LOTL) techniques to avoid dropping new malware.

Long-Term Persistence:

- Backdoors lay dormant for 12-14 days.
- Modified native services for access, for nearly two years.

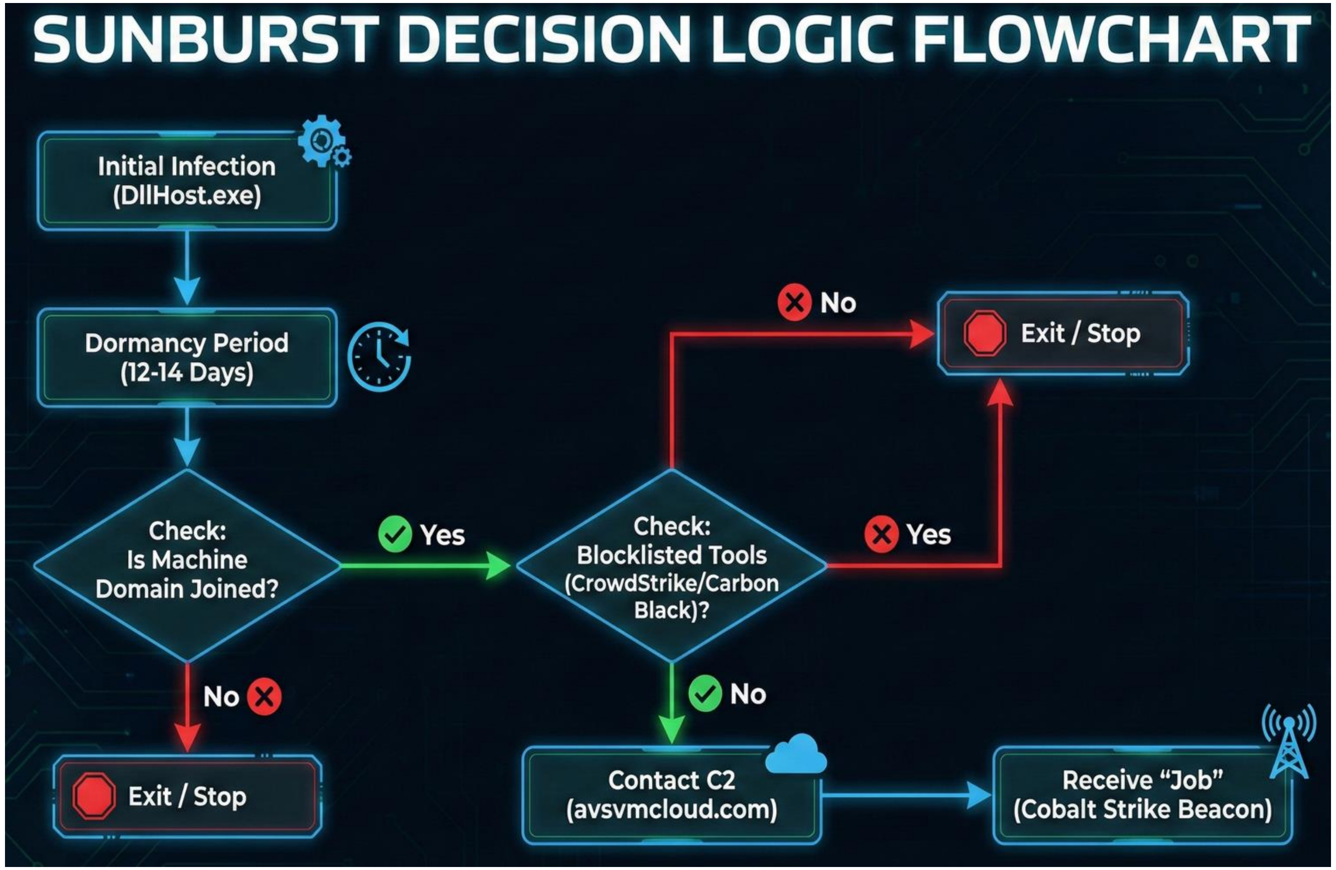
Technique: Blended malicious traffic with legitimate Orion Improvement Program (OIP) protocols to evade detection.

Actions on Objectives & Post-Exploitation

How did Sunburst select high-value victims?

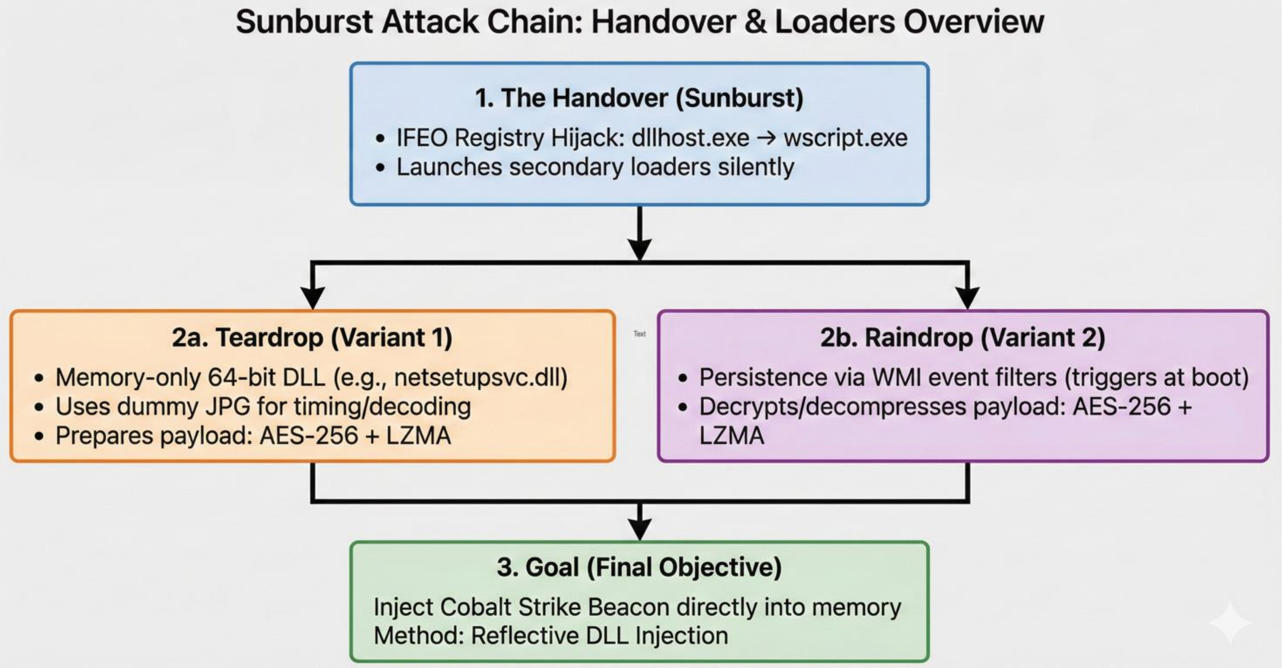
Target Selection (Environment Profiling)

- **The Dormancy Period:**
  - Sunburst lay dormant for 12–14 days post-infection to avoid sandbox detection.
- **Reconnaissance & Profiling:**
  - Executed CollectSystemDescription
    - gather hostname, domain, and OS version.
  - **The "20% Cut":** Only activated on ~20% of targets
    - focused on domain-joined enterprise environments
- **Escalation Triggers:**
  - Prioritized systems with stable connectivity.
  - Full activation only occurred if the machine was identified as a "high-value" target (Gov/Tech).


SUNBURST DECISION LOGIC FLOWCHART
Initial Infection (DllHost.exe)
Dormancy Period (12-14 Days)
Check: Is Machine Domain Joined?
Yes
No
Exit / Stop
Check: Blocklisted Tools (CrowdStrike/Carbon Black)?
No
Exit / Stop
Yes
No
Contact C2 (avsvmcloud.com)
Receive "Job" (Cobalt Strike Beacon)

Actions on Objectives & Post-Exploitation

Delivery Mechanisms



Sunburst Attack Chain: Handover & Loaders Overview

1. The Handover (Sunburst)
- IFEO Registry Hijack: dllhost.exe → wscript.exe
- Launches secondary loaders silently

2a. Teardrop (Variant 1)
- Memory-only 64-bit DLL (e.g., netsetupsvc.dll)
- Uses dummy JPG for timing/decoding
- Prepares payload: AES-256 + LZMA

2b. Raindrop (Variant 2)
- Persistence via WMI event filters (triggers at boot)
- Decrypts/decompresses payload: AES-256 + LZMA

3. Goal (Final Objective)

Inject Cobalt Strike Beacon directly into memory
Method: Reflective DLL Injection

Teardrop Deployment

- Delivered as a **64-bit DLL** that imitates real Windows files
  - Typically dropped **during lateral movement** by the attacker.
- Installed as a **Windows service** to maintain persistence.

Sunburst Handover

- Drops a **VBScript** into a trusted Windows subfolder
- Sets a **registry key**: whenever **dllhost.exe** runs, it gets redirected.
- This hijack forces **dllhost.exe → wscript.exe → rundll32.exe**, which then loads the follow-on DLL.
- Allows the attacker to load a **secondary payload** while keeping Sunburst active and hidden.

Raindrop Deployment

- Similar to Teardrop, but with a stronger focus on **WMI based persistence**.
- Uses **WMI event filters**
  - trigger when system uptime changes.
- This causes **rundll32.exe** to automatically run the payload at boot
- Uses **unique DLL names/paths per host**

Actions on Objectives & Post-Exploitation

Cobalt Strike Loading

Reflective DLL Injection (Both Loaders)

- Uses **Cobalt Strike's Loader** to run the Beacon **entirely in memory**
- Teardrop and Raindrop both loaded Cobalt Strike Beacon directly into memory.
- Teardrop used basic tricks:
  - opening a fake JPG and checking a registry value.
  - unpacked Beacon using a simple XOR step.
- Raindrop used stronger methods:
  - Encryption + decryption and XOR to unpack Beacon.
  - added delays and ran Beacon in a separate thread to avoid detection.

Beacon Initialization

- Beacon v4 uses **malleable C2 profiles** to blend in with normal traffic.
- Sends traffic with a common **User-Agent (Mozilla/5.0)**.
- Uses **HTTP GET/POST** channels, sometimes with **steganography** for hiding data.
- Each Beacon has **unique watermarks** + its own **spawnto** for precise operator control.

Actions on Objectives & Post-Exploitation

File-less Execution: In-Memory Cobalt Strike Beacon in SolarWinds Attack

Process Masquerading

Spawnto Technique

- Malware launches malicious code inside **legit processes** like *conhost.exe* or *svchost.exe* to look normal.

Blending & Evasion

- Uses **malleable C2** profiles (HTTP over 443, fake Mozilla user-agent, sometimes steganography).
- **Timestomping** and long sleep/jitter cycles to hide activity (T1036.005).

Lateral Movement

- Uses **SMB named pipes** and **PowerShell injection** to move through the network.
- Pretends to be **AD admin tools** to perform DCSync-style operations.

MITRE ATT&CK Mappings (Picus Security)

Tactic	Technique
Execution (TA0002)	T1055.001 (DLL Injection); T1059.003 (Cmd Shell)
Defence Evasion (TA0005)	T1027.002 (Obfuscation: Packing); T1564.004 (Hide: In-Memory); T1574.002 (DLL Search Order Hijacking)

Actions on Objectives & Post-Exploitation

Living-off-the-Land Techniques (LOTL) in SolarWinds Attack

PowerShell Abuse

- Attackers run hidden/Base64 PowerShell to steal credentials (LSASS dump).
- Use PowerShell to raise privileges, hide activity, and delete evidence.
- Move sideways in the network using SMB scripts for basic user/system checks.

WMI Abuse

- WMI used to gather system info (processes, network details).
- Can set WMI events to run malware automatically at startup.
- Supports lateral movement through remote system queries.

PsExec Abuse

- PsExec-like tools run commands remotely over SMB (TCP 445).
- Uses admin shares to look like normal IT activity.
- Hides using timestomping and secret named pipes; can be blocked by ASR rules.

Kerberos Abuse

- Attackers create fake Kerberos tickets (Golden/Silver Tickets).
- Kerberoasting to crack service accounts and gain domain control.
- Fix with **krbtgt** resets and monitoring for unusual ticket activity.

Actions on Objectives & Post-Exploitation

SAML Token Theft & Forgery in SolarWinds Attack

- Hackers stole the **SAML signing key** and used it to make fake login tokens.
- These fake tokens let them **skip MFA** and get into cloud apps like email.
- They added their own accounts and trust links to **stay inside the network**.
- Warning signs: **long token lifetimes**, missing MFA, cloud logins with **no matching on-prem login**, strange federation changes.
- Defenses: **rotate keys**, enforce MFA, remove unused SAML settings, and watch for **odd tokens or federation edits**.

Mitigations & Defenses

- **Response**: Reset krbtgt (2x), rotate creds (>25 chars/90 days); revoke ADFS certs/principals; disable unused SAML
- **Zero Trust**: MFA everywhere (99.9% reduction), PIM just-in-time, contextual checks (device/IP lists); cloud IDP migration
- **Enhancements**: Correlate logs for claims; enable PS/Exchange auditing; ASR for unsigned SAML; periodic Sparrow/Hawk scans

Data Exfiltration

- “Data exfiltration was slow, encrypted, and disguised as legitimate cloud traffic.
- Small data fragments — just 10 to 50 KB — were sent over HTTPS to attacker-controlled cloud-like domains, appearing as routine API telemetry.
- For larger datasets, attackers compressed files using 7-Zip LZMA and uploaded them to **OneDrive or SharePoint** through the Microsoft Graph API using **forged SAML tokens**. Because the authentication was valid, these actions looked legitimate to cloud systems and bypassed on-premises proxies.
- Sessions were infrequent, randomized, and used malleable C2 headers to mimic genuine Office 365 or Windows Update traffic. When needed, fallback methods included DNS TXT exfiltration, SMB staging, or WMI-based transfers.
- Overall, attackers weaponized cloud ecosystems and encryption to hide in plain sight.

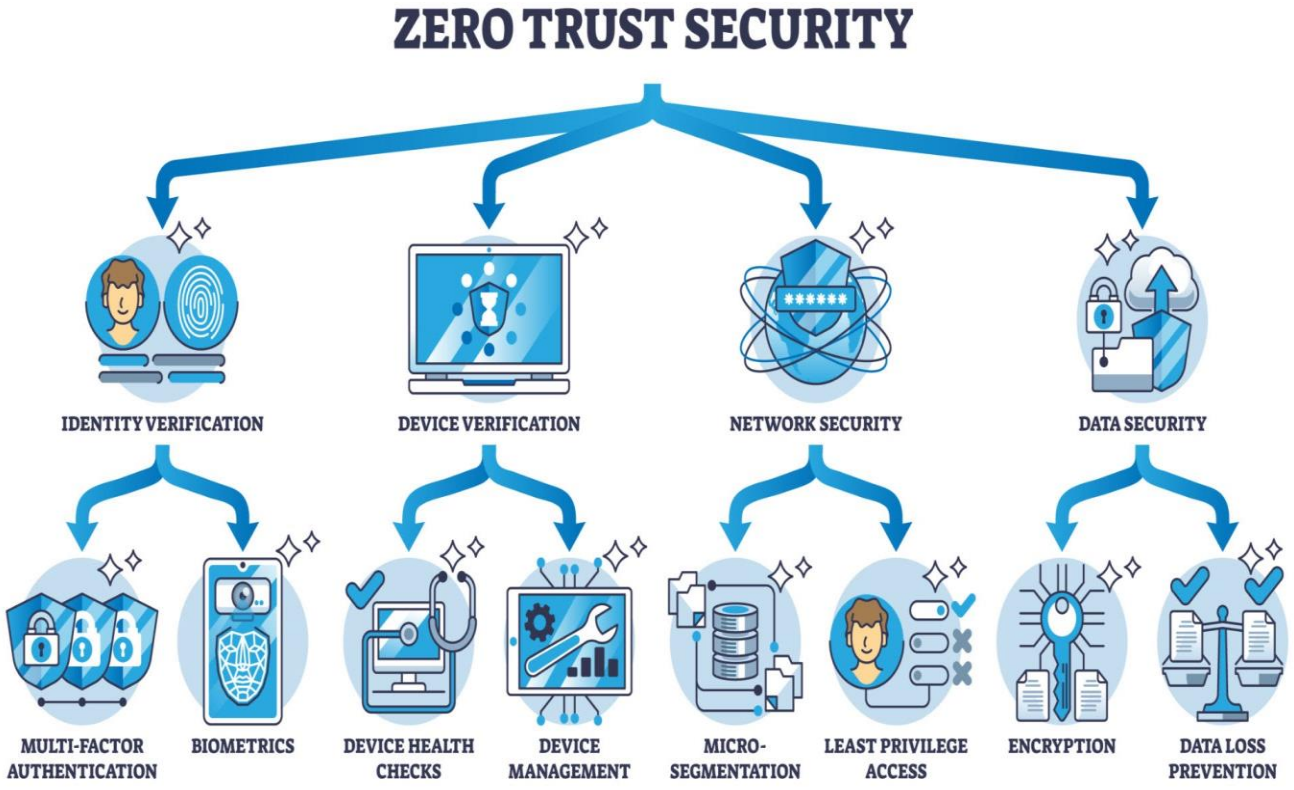
Lessons Learned & Future Direction in the SolarWinds Attack

Zero Trust

•**Continuous Verification**: Verify explicitly at every access via micro-segmentation & JIT privileges; counters lateral movement—mitigates Orion's "complete mediation" violation w/ API gateways & behavioral analytics

•**Least Privilege Enforcement**: RBAC/PBAC limits dev/build access (70% exploit reduction in sims); top "Closing the Chain" mitigation for insider risks

•**Assume Breach Mindset**: UEBA for runtime decisions; 2025 NIST SP 800-207 rev2 AI identity fabrics handle SAML forgery



network-insight.net

Lessons Learned: Future Direction after the SolarWinds Attack

Supply Chain Security

- **Vendor & Dependency Vetting:** Require SBOMs/VEX for all suppliers; use blockchain provenance (SLSA) for tamper-proof artifacts.
- **Environmental Scanning:** Automated OSS risk tools with dependency graphs + threat intel; quarterly scans to prevent maintainer-targeted exploits.
- **Boundary Protection:** Air-gapped signing and multi-attestations; isolated builds (e.g., CISA SCuBA 2025) reduce breach risk by ~40%.

Secure CI/CD

- **Pipeline Integrity:** Code signing, SBOMs, and fuzzing at all stages (SLSA L3+); prevents DLL injection and GitOps attacks.
- **Developer Tooling:** Safe libraries and automated code reviews; security task lists aligned with DevSecOps standards (GitHub Advanced Security adoption +150%).
- **Sustainable OSS:** Maintainer funding and multi-vendor reviews; OpenSSF $50M fund mitigates burnout-related exploits.

Future Directions

- **Policy & Standards:** EO 14028 updates integrate security into dev pipelines (NIST SSDF).
- **Resilient Architectures:** Protect against vulnerabilities like Log4j and XZ Utils.
- **Holistic Security:** RBAC enforcement and AI-powered scanning for dynamic threat detection.

Conclusions

- **What Happened**
  - APT29 (Russia) hid in SolarWinds software for months
  - Stole emails & secrets from ~100 big targets (gov, tech, health)
  - Used tricks like fake logins & cloud hiding to stay unseen

- **Big Lessons**
  - **Trust No One Blindly**: Check everything – use RBAC & AI alerts (cuts risks 70%)
  - **Secure the Chain**: Demand SBOMs & safe builds; fund open-source devs
  - **Act Fast**: Scan often, share intel – turn breaches into better defense

- **What's Next**
  - Build strong: EO 14028 rules + AI tools for all
  - From attack to armor: Smarter security keeps us safe

Bibliography

- CSIRT *Sunburst Threat Report*, December 2020
- Blog posts by Sudhakar Ramakrishna @ SolarWinds
- *SUNSPOT: An Implant in the Build Process*, by CrowdStrike Intelligence Team
- *Understanding Supply Chain Attack Tactics with Case Study*, brandefense
- *A Review of the SolarWinds Attack on Orion Platform using Persistent Threat Agents and Techniques for gaining Unauthorized Access,* A.Kruti, U.Butt, R.B. Sulaiman
- *SolarWinds attack explained: And why it was so hard to detect,* Lucian Constantin @ CSO
- SolarWinds hack explained: Everything you need to know, S. Oladimeji, S. M. Kerner @ TechTarget
- *SolarWinds: the most sophisticated attack in history*, by Cybersecurity Specialist Veronica Paolucci