Detect and prevent the SolarWinds build-time code injection attack

By Ariel Levy

Published: 2021-02-17 · Archived: 2026-04-05 13:28:42 UTC

Published February 17 2021 · 3 min. read

We have developed a patent-pending technology to detect and prevent SolarWinds-style attacks before shipping binaries to production, in both on-prem and cloud environments.

In order to understand how this new capability works technically, let's briefly examine how the attack was executed and why it went unnoticed for so long.

THE MOTIVATION

In late 2020, a complex supply chain attack against SolarWinds made headlines globally. Malicious code that implemented a back door was injected into the source code during the build process, exposing every SolarWinds customer to a significant threat. Although supply chain attacks are a known concept, this is the first disclosed detection of one at such complexity and vast scale.

Internal and external investigations of the scenario discovered that malware was running on the SolarWinds Orion IT management build environment and waiting for *msbuild.exe* (the C# compiler) to run. When a build process started, the malware verified that the build target was *SolarWinds.Orion.Core.BusinessLayer.dll*. If so, it immediately replaced the staged C# files with a version containing the back door's code **before** compilation, resulting in malicious code inside. *This dll was later digitally-signed, which resulted in the file avoiding significant scrutiny.*

Once the build environment is penetrated, there are countless methods to influence the resulting product. **While the pipeline's input is readable (and testable) code, its output is a digitally-signed binary. Comparing them in order to identify a build-level attack wasn't possible … until now.**

The challenge

Taking binary code and restoring it to its original source code is a practically impossible task. Compilation is a complex, non-reversible action. A compiled binary is packed with information, optimizations, and metadata that are continuously changing. Even if you take the same source code and compile it again a minute later, the binaries won't be identical.

In addition to the **non-readable binary** challenge, the **variety of CI/CD tools** and approaches is extremely broad. These tools are used differently by every team (where each approach handles dependencies, common code, and

additional resources in a unique way). Add to this the fact that the CI/CD pipeline is designed to be invisible to its users and is almost never inspected and you get a huge **DevSecOps blind spot**

We think about this from a totally different perspective

The solution

With a deep understanding of the source code, it is possible to determine whether or not it matches the relevant binary file (based on patent-pending technology). By the time the build process starts, Apiiro will have already learned the source code and developer experience using its risk-based AI engine. Once the Apiiro platform knows all the code components, security controls, logical flows, data types, and their relations, the next phase is to analyze the binary.

Let's take a .NET binary, for example. The Apiiro platform will parse and perform the following actions on the executable files:

- Learn all possible logic flows and symbols
- Clean out all auto-generated compiler logic
- Adjust expected differences between runtime versions
- And more…

With the normalized entity relations from the binary, Apiiro runs graph comparison algorithms against the same data it learned from the source code.

Apiiro's algorithm is also aware of all possible legitimate code changes during compilation (AOP frameworks, optimizations, etc.) and is able to **distinguish only inserted malicious functionality**, **be it a small configuration change or full back door code**.

To protect your organization from the next supply-chain attack, do not allow your vendors to deploy or upgrade their products without:

- Performing the above comparison for every compiled binary to detect source code manipulation
- Validate included dependencies, external (using a digital signature) and internal (using contextual knowledge of all the organization's repositories)
- Compare additional resource files, if present, to their counterparts from the source commit, taking into account potential build-time templating processes

These steps will be done simply by uploading the built binaries to the Apiiro platform using a dedicated API that can easily be integrated into every CI/CD pipeline and read-only access to the source control manager.

When Apiiro performs this process at the end of every build, you get end-to-end validation that no unwanted code is injected into your product before shipping to customers

---