The SolarWinds Attack and Farm-to-table Methods in the Development Process

Mitigating disasters through supply chain security

Felix Pojtinger

2022-01-06

Introduction

Introduction

On 13 December 2020, FireEye, a US-based cybersecurity company, detected a large supply chain attack directed at customers of SolarWinds using their Orion monitoring and management solution. The actors of the attack, which are now presumed to be of Russian origin, were able to gain access to data of several high-profile private and public institutions. While the attack was detected and made public by late 2020, it has been active since at least Spring 2020, resulting in widespread lateral movement and data theft. While high-profile attacks against institutions have become more frequent in recent years, the sophistication and threat vector of this attack has shown the need for more advanced modelling techniques and security measures for supply-chain security, which this paper tries to introduce the reader too.

Related Work

Related Work

FireEye provides the initial report on the SolarWinds Attack, detailing the attack's architecture, resulting vulnerabilities and information about the attackers. Sherman and Mark give an overview of the past and presents risks in the supply chain and make general recommendations on means of preventing them. Alsabbagh, Bilal, Kowalski and Stewart present a framework to model a software supply chain from a social and technical perspective, with a focus on highlighting ways to find threats in existing chains. Torres-Arias, Afzali, Kuppusamy, Curtmola and Cappos introduce a concrete implementation of a system with a wholistic approach to supply chain security, as well as an analysis of its level of protection against historical attacks.

The SolarWinds Attack

The SolarWinds Attack

In the case of the SolarWinds attack, the backdoor was built into a digitally signed component of the Orion platform (SolarWinds.Orion.Core.BusinessLayer.dll) which communicates with external servers using HTTP. After an initial dormant period of up to two weeks, the malicious component was able to receive commands such as transferring and executing files, profiling a system, disabling services and issuing reboots from command and control infrastructure. Traffic was obfuscated as by masquerading as telemetry communications similar in structure to the Orion Improvement Program. In addition to these obfuscation methods, the malware is sandbox-aware to evade detection by antivirus tools and increase the effort required to do forensics.

In order to deliver the malware to users of SolarWinds Orion, the supply chain was attacked. The attackers were able to compromise SolarWinds' keys and digitally sign a trojanized version of a

Overview of Supply Chain Security

Overview of Supply Chain Security

While supply chain attacks are not a new occurrence, the SolarWinds attack has forced both developers and security analysts to take the subject more seriously. Historically, most software did not have a particularly complex supply chain; software was limited in size, function and audience, organizations had their own developers and created, for the most part, their own software without many external dependencies. Modern software development has however led to a strong increase in supply chain complexity, as modern software is large enough for it not be manageable by a single organization.

As a result, modern software development is in large part about assembly of existing software, which leads to very long supply chains. A typical "Web 2.0" application could for example include an app server, HTTP server, XML parser, database client, C libraries & compiler, all of which also have their own supply chains

Modeling the Supply Chain

Modeling the Supply Chain

Fundamentally, software supply chains have a lot in common with physical or hardware supply chains. Supply chains are created by either deploying and using a product directly or by reproducing it as a new product in repetition, and as a result, traditional supply chains tend to have risks such as late product delivery, counterfeits and human errors. Many of these risks also apply to software supply chains and, just like they are in traditional supply chains, must be counteracted using risk management, for which a threat model is a prerequisite. Current practices mostly follow on the classical CIA triad (confidentiality, integrity, availability), but for software supply chains, the security objectives of confidentiality, (data) integrity, source authenticity, availability and non-repudiation are the most important. While organizations such as NIST publish best practices, concrete models of software supply chains are still a rather new subject.

“in-toto”, a Framework for Supply Chain Security

“in-toto”, a Framework for Supply Chain Security

in-toto provides a concrete technical implementation of a supply chain security system. It tries to protect against supply chain attacks based on version control systems (such as the breaches of the Linux kernel, Gentoo and Google), build environment (such as the CCleaner breach), software updaters (such as the breaches of Microsoft, Adobe, Google and various Linux distros such as Debian) and others.

Current supply chain security systems are mostly limited to securing individual steps. Technical measures include for example Git commit signing, which allows for controlling what developers can modify in a repo, reproducible builds, which enables (re-)building a package by multiple parties and ensuring a bit-by-bit identical result, and various methods of software delivery, such as APT, DNF or Flatpak. While securing these individual steps is useful and increases security to a certain extent, is leaves

Results

Results

When evaluating the effectiveness of supply chain security systems against historical supply chain attacks, it is important to keep the context of their application in mind. Three applications (a Linux distribution, cloud-native deployment and a language-specific package manager) have been analyzed, which each lead to different levels of security.

In the case of Linux distribution, the Debian rebuilders project was chosen by the in-toto maintainers. Debian rebuilders are part of the reproducible builds project, which intents to create bit-by-bit reproducible package definitions, meaning that a source package can be rebuilt on a separate host and result in a binary which is bit-by-bit identical to the original build. In this case, a apt—transport for in-toto metadata has been implemented, which is used to provide attestations of the validity of resulting builds using link metadata. This allows for cryptographically asserting

Summary and Conclusions

Summary and Conclusions

Assessment of protection against previous breaches was done in three categories: Control of infrastructure but not functionary keys, control of parts of the infrastructure or keys of a specific functionary, and control of the entire supply chain by compromising the project owner infrastructure, including keys. In tests, the majority of surveyed attacks (23/30) did not include a key compromise. In this case, in-toto's client inspection would have detected the tampering. The Keydnap attack, in which an Apple developer certificate was stolen and used to sign a malicious software package, inspection with in-toto would have detected the attack due to an unauthorized functionary signing the link metadata. In another attack, the developer's SSH key was used to sign a malicious Python package; this would have been prevented with in-toto as the files extracted from the malicious package would not match the source coded recorded in the first step of the