THE SOLARWINDS TIPPING POINT

digicert®

Table of contents

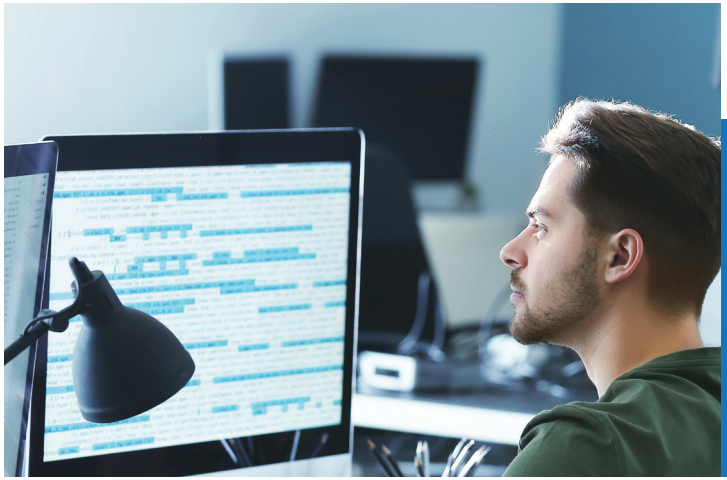
The Great Supply Chain Robbery

On Sunday, December 13, 2020, the United States acknowledged that hackers had broken into several federal government computer networks. Over the following weeks, the world learned that this sophisticated attack was almost certainly the work of Russian agents, who had managed to collect sensitive data, undetected, for months. As experts learned more, the extent of the breach became clear. This was not just a successful act of espionage. It was one of the largest malware intrusions in years[1].

The attack was traced back to a software system, called Orion, an IT management program built and sold by the Texas-based cybersecurity company SolarWinds. Within a short time, SolarWinds became synonymous with data breaches. But SolarWinds isn't the first company to face a damaging security breach, and the headlines don't tell the full story.

This breach really began long before the attack was initiated, when malicious code was slipped into the DNA of Orion. Now part of the CI/CD build, it sat as a ticking time bomb, hidden within the software itself. The hackers were patient. The timebomb had a very long fuse. They knew they could exact the most damage by waiting. Over the months that followed, Orion spread across the world, with installs in government agencies, major corporations, even other software companies.

And then, one day, sometime after March 2020, the bomb went off. By the time anyone was aware of the infiltration, it was too late. This isn't only because the malware worked, but also because Orion was everywhere. It wasn't just a matter of isolating one breach, one server. It wasn't a single patch. The attack was so large, we still don't know how many systems, businesses and organizations were affected—and we probably never will.



1 https://www.reuters.com/article/us-cyber-solarwinds-microsoft/solarwinds-hack-was-largest-and-most-sophisticated-attack-ever-microsoft-president-idUSKBN2AF03R

SolarWinds—there's the time before and the time after

What makes the SolarWinds attack so astonishing is its scale. The infected Orion software was sold to more than 33,000 customers. Sunburst, the aptly named malicious code, was distributed to as many as 18,000 organizations. For months, this Trojan Horse sat inside the firewalled networks of tens of thousands of unsuspecting businesses and government agencies.

In the aftermath, forensics discovered Sunburst intrusions in 425 of the U.S. Fortune 500 companies, each of the top ten U.S. telecommunication companies, five of the top U.S. accounting firms, and Microsoft itself. Sunburst gave foreign agents access to the White House, all five branches of the U.S. military, the Treasury Department, the Pentagon, the State Department, the Department of Justice, the Department of Homeland Security, the National Security Agency, and the National Nuclear Security Administration. Outside the United States, Sunburst hit NATO, the United Kingdom government, the European Parliament, and others. Public, private, large and small, Sunburst made it through the gates, carried by supposedly safe software that was signed.

Just a decade before, many cybersecurity experts were saying that a supply chain attack was extremely unlikely. It didn't fit the profile of the type of vulnerabilities exploited by cyber criminals. And because no individual or single company could protect every link in the build, trying to secure the supply chain was a fruitless task. This idea was so pervasive, Stacy Simpson of SAFECode, along with a group of leaders from the field, published a warning, recommending that DevOps professionals ignore these viewpoints and implement code signing best practices to close intrusion points in the CI/CD supply chain.[2]

But most people continued to believe that a supply chain attack was either impossible to pull off or impossible to prevent. After the SolarWinds breach, the world knew better.

The forensics of a preventable disaster

The type of vulnerability that led to the SolarWinds attack is nothing new. Infected code has been a concern for security-conscious professionals since the invention of the CI/CD pipeline. But until SolarWinds, there had never been such a condemnable breach, one that was massive in scale, very public, and totally preventable.

2 http://safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf

Like many companies, SolarWinds released Orion as a legitimate, signed software package. In signing the software, SolarWinds promised customers that they would be informed of any tampering or malware after the build and release. When signed software enabled a breach, the digital world was forced to pause and reexamine the entire software supply chain process. How do we protect customers and end-users if code can contain malware injected into the supply chain before signing?

Code signing itself is still proven technology, but the SolarWinds breach showed that code signing alone isn't enough to protect the supply chain. Without source code scanning, 3rd party library scanning, proper staff vetting, and a secure build environment—all in addition to code signing—software is open to attack. It's like having an advanced security system in your home, but you only turn it on for an hour at a time.

In the aftermath of SolarWinds, the world learned of new supply chain attack vectors, and that signing software may be the last step in the DevOps cycle, but it's not the only one. The entire process must be secured, controlled and validated. Without complete security, the DevOps loop is never closed.

Signing isn't the last step in the build, it's every step

When software is signed, it's secured against tampering from the moment of signing moving forward. But there's more to it than encryption. Code signing verifies the identity of the publisher, and it checks the integrity of the code after downloading, so users know the software hasn't been altered. Because signed software means that software is trusted, it's vital that DevOps teams deliver software releases that are verifiably free of malware. Signed software containing malware bypasses all the code signing checks ubiquitous to public and private software distribution ecosystems. In essence, signing software with compromised code tells the security gateways through the rest of the supply chain and at the user access point not to look for problems.

Protecting software integrity throughout the entire CI/CD process is the only way to deliver complete trust. It's not enough to sign software at the end of your team's development cycle. Code written by other teams in the supply chain, including open source code and 3rd party library code, needs to be scanned for malware and other anomalies. Only clean code should be used to build software, and then signed, using the right signing keys, which have been properly issued, managed and stored. Access to these signing keys should be limited to only the users or automated build servers authorized to use keys for signing.

This can seem like a daunting task, which is why the developers often make use of weak security practices or shortcuts in order to keep builds moving forward. We all know that speed and efficiency is the beating heart of DevOps. Cumbersome steps that lead to missed deadlines can bring the CI/CD process to a grinding halt. A solution that isn't agile—one that creates delays or interruptions—isn't a solution at all.

In many cases, these signing processes can be automated to decrease the need for human touches and give your developers time to focus on building code without sacrificing the integrity of your security posture.

There's best practice and there's not secure

Signing software is important in and of itself, but it's all too easy to overlook the management of signing policies and practices—and that's where we find security gaps that can be exploited. When it comes to software security, the choice is binary. You can choose best practices, or you can choose to leave your supply chain open to attack.

Implementing code signing best practices will help to ensure that signing isn't the weak link in your supply chain. And, if implemented correctly, code signing can also help to prevent malware injection.

1. Protect and control the use of signing keys and private keys

Private keys can be duplicated or stored in many locations. Physical keys like USB tokens can be lost or left in an unlocked location. Disgruntled or negligent employees may lose track of keys or steal them.

There are two options for fully securing private keys. You can store keys in a properly maintained HSM, or you can leverage a managed PKI service to secure the keys for you.

2. Enforce multi-factor authentication and FIPS-compliant storage with public trust deployment

A password can be forgotten or compromised. Keys can be shared, lost, or stolen. Likewise, physical USB tokens can be misplaced or misused.

To guard against user error or malicious intent, strong access controls need to be in place. Multi-factor authentication (MFA) requires two forms of credentials from users, proving identity. HSMs provide strong authentication and resist physical tampering, ensuring secured keys can be accessed only by authorized users.

3. Operate with full visibility

Without end-to-end auditing and user accountability, it is difficult or impossible to track key misuse or signing anomalies.

Both keys and users need to authenticate prior to signing, and this authentication and signing should be completed at every step in the CI/CD process. These signatures must be tracked and auditable, so users are accountable, and you know who signed what. The use of signing keys by authorized users at unauthorized times can help IT security teams detect problems before the software is moved to the next link.

4. Control your account-level security

When keys are used without permission controls, it is difficult or impossible to protect and monitor keys and signing usage.

Keys should be assigned by permission and role, and separated by product, project or team. Administrators should be able to generate private keys and certificates and distribute keys to teams. Users should be restricted to those assigned to signing-related workflows. Non-human components like build servers should have keys assigned to permit automation.

5. Control your organization-level security

Organizational policies are just as important as individual roles and permissions. Without a holistic policy and control structure, individual security processes may fall out of alignment, creating vulnerabilities.

Create and enforce internal security policies across the board, from your full organization to product development. This should include cryptographic algorithms or key size, certificate validity periods, and certificate types or approval workflows.

6. Rotate your keys

When the same key is used to sign multiple pieces of code, any compromise to one piece puts all pieces at risk.

The ability to issue multiple keys quickly and easily, and to rotate those keys, protects against an isolated intrusion turning into a widespread attack.

7. Set policies that assign certain types of keys to certain projects

Without tight control over different types of keys, organizations may lose visibility over key access and usage, opening the door for unauthorized or inappropriate key signing activities.

Re-use specific signing keys for specific products or projects restricted to specific users and teams. Where possible, use on-the-fly private keys and certificates, so each release is signed by a unique private key and certificate.

8. Never let your unsigned software leave your environment

Transferring full files for signing is not only slow and resource-intensive, it also opens the possibility for interception or tampering in transit.

Hash signing solves all these concerns. Only the hash is uploaded to the cloud for signing, so the process is nearly as fast as local signing. The code itself stays on your internal servers, so your intellectual property cannot be stolen or altered.

9. Regularly scan all your intellectual property

It may seem like a simple step, but a signature signals trust. Users assume signed code is safe code. If a virus or malware is already embedded, the developer will send supposedly trusted code to their customer.

Scan source code and 3rd party library code, as well as compiled code. Check for any malicious injections and sign only when you've verified that everything in the software is clean.

10. Make signing reproducible to validate what you're signing

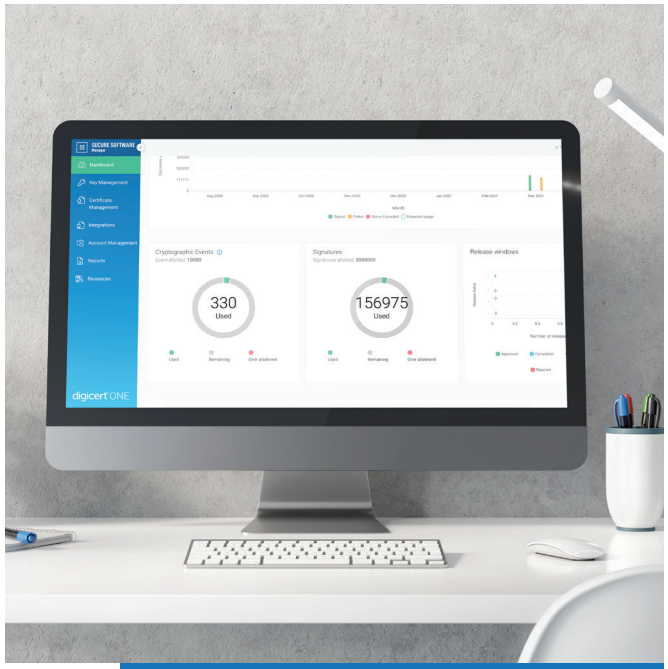
When you cannot compare builds against a baseline, you cannot be sure the binaries are identical and reproducible by a quorum of users. This increases the risk for undetected malware injection.

Compare the hashes of your binaries sent for signing against the baseline before signing. This comparison is needed to validate that the production build is generating the same outcome as the expectation in your test and QA cycles.

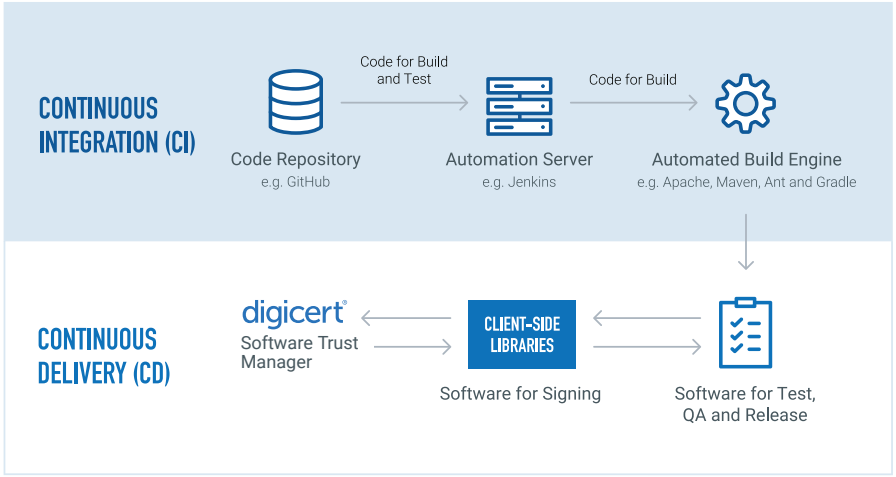
Continuous signing facilitates the need for speed

DigiCert Software Trust Manager is built to facilitate agile DevOps. By leveraging automated processes, Software Trust Manager enables continuous signing that delivers the highest level of trust, with end-to-end security, all without any extra steps or a minute lost.

Sign code, software, and binaries rapidly, easily, and at scale. Create reproducible build processes that automate baseline parameters, so malware injection is detected before compromised code can be released. Secure key management, control permission-based access, and utilize reporting and tracking capabilities for visibility and accountability. DigiCert Software Trust Manager is the single solution for implementing and maintaining code signing best practices at every stage of your CI/CD process.



Automated software signing with CI/CD processes using DigiCert® Software Trust Manager



We have been warned, and now we take action

When Stacy Simpson published the SAFECode report recommending CI/CD supply chain protection, she couldn't have known she was describing not just a potential attack, but a very real attack—one that would be considered amongst the most notable in digital history. In 2010, a supply chain attack was considered a remote possibility for a weak intrusion. Ten years later, the SolarWinds breach would be considered an example of a devastatingly successful hack and theft.

Ironically, the solution to the CI/CD supply chain threat was understood in 2010 and published in the SAFECode report. Because individual DevOps teams and single organizations cannot hope to control all the parts of the chain, the only way to combat supply chain cyberattacks is to implement best practices that continually monitor the integrity of the code while it is in development. If everyone in the build process scans and signs their code, the entire supply chain is protected against weak links.

In 2010, this may have been a prohibitive expectation. If security practices delay or interrupt the CI/CD process, then security itself can become as damaging to DevOps as the risk of a breach.

But today, code signing processes can be automated. Automation alone offers the tools needed by DevOps to severely restrict the attack surface of the entire CI/CD landscape. With automation, DevOps teams can implement continuous signing, so code, software and apps are protected using an end-to-end integrity and encryption system—all with minimal human touches.

With the success of the SolarWinds breach, we can only assume that other cyber criminals have already initiated their own supply chain attacks. Part of defending against this threat is moving as an industry toward code signing best practices. If everyone in the supply chain is monitoring and securing their code, then each link is protected during development and during transit to the next stop in the pipeline.

At the end of the day, SolarWinds isn't just the story of an astonishing act of cyberespionage. It's a tipping point, a watershed moment in the history of code and software development. The world of DevOps cannot afford to say we weren't warned. The potential risk was known. The practical devastation is now a matter of record. The tools exist to easily protect against the threat, without delay or interruption. Now, it's time for the entire industry to leverage these tools and implement code signing best practices, so the DevOps supply chain is protected from start to release.