LEARNING MADE EASY

ReversingLabs
Special Edition

Software Supply Chain Security

for dummies®
A Wiley Brand

Secure third-party and open source software

Establish a proactive threat-hunting program

Create a comprehensive security plan

Brought to you by:

REVERSINGLABS

Paul F. Roberts
Charlie Jones

About ReversingLabs

ReversingLabs (RL) is the trusted authority in software and file security. RL's Spectra Assure™ is a modern cybersecurity platform used by some of the world's leading software producers and enterprises to verify safe software binaries. Trusted by the *Fortune* 500 and leading cybersecurity vendors, ReversingLabs Spectra Core engine powers software supply chain and file security insights, tracking over 40 billion files daily. Only ReversingLabs provides the final exam to determine whether a single file or full software binary presents a risk to your organization and your customers.

ReversingLabs Spectra Assure™, powered by AI-driven complex binary analysis, identifies compliance issues, exposures, and threats like malware, tampering, vulnerabilities, mitigations, exposed secrets, and license issues — all without the need for source code. Providing the "final build exam," RL's Spectra Assure gives software producers and consumers the comprehensive risk analysis that enables them to identify, assess, and resolve critical issues, delivering the trust and assurance needed to ship, update, or deploy critical software.

Software Supply Chain Security

ReversingLabs Special Edition

by Paul F. Roberts and Charlie Jones

for dummies®
A Wiley Brand

Software Supply Chain Security For Dummies®, ReversingLabs Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-81295-1 (pbk); ISBN 978-1-119-81297-5 (ebk); ISBN 978-1-394-28363-7 (ebk)

Publisher's Acknowledgments

Development Editor: Rebecca Senninger

Project Editor: Dan Mersey

Acquisitions Editor: Traci Martin

Senior Managing Editor: Rev Mengle

Client Account Manager: Jeremith Coward

Production Editor: Saikarthick Kumarasamy

Cover Photo: © Shutter2U / Adobe Stock

Special Help: Ashlee Benge

Table of Contents

Introduction

All organizations are becoming increasingly reliant on software and cloud-based services. As such, the focus of cybercriminal and nation-backed hackers has shifted from attacking individuals to attacking the software and services that power modern organizations. After all, why expend the time, effort, and resources targeting one organization for a marginal profit when an attacker can expend the same effort to extort hundreds or thousands of users of a piece of software for a much larger sum?

These basic economics are driving an explosion in attacks on software supply chains: the extensive web of proprietary, open source, and commercial software that every modern organization depends on. In just one measure of that, since 2020, the number of malicious packages discovered on common open source repositories has jumped 1,300 percent. During that time, high-profile compromises of both commercial and open source software applications have made headlines.

With the increase of threats, it's even more important that your organization boasts a robust software supply chain security ecosystem, one that's both understandable and achievable to maintain for you and your organization.

About This Book

As with any issue related to cyber security, there is no "silver bullet" or magic fix when it comes to securing your software supply chain. Instead, your organization needs to accurately assess its risks and then plan and invest for the long term to safely manage those risks.

In this book, we discuss the many elements of a modern and effective software supply chain program and talk about steps your organization can take to both secure development environments, search for evidence of malicious activity within your development pipeline, and defend your larger software supply chain.

Software supply chain risk is a large and growing problem that impacts organizations at many different levels. That's why all

professionals can benefit from this book, including CISOs or other senior security leads; software development, quality assurance, or testing professionals working on new or legacy development projects; and security professionals working on internal network or application security teams.

Icons in This Book

We use a few icons you'll find in the margins of the book to point out useful information:


REMEMBER

Remember icons contain nuggets of information that are worth memorizing.


TECHNICAL STUFF

If you're the kind of person who likes to have the background info or extras, read the paragraphs marked with the Technical Stuff icon. But you can safely skip them if you want to.


TIP

A Tip icon gives you some extra help or provides some additional insight.


WARNING

Pay extra attention to paragraphs with a Warning icon. They give information that, if left unheeded, can cause your organization harm.

Beyond the Book

Software represents the largest under-addressed attack surface in the world, and classic application security tools cannot address the full scope of threats impacting the software supply chain.

This book provides ideas and guidance on the kinds of investments required to secure your organization's software from malicious compromise and put you on a solid footing with both your suppliers and your customers.

If you or your organization want to have more trust in its software, be sure to visit `reversinglabs.com` to learn more about the Spectra Assure software supply chain security solution.

IN THIS CHAPTER

- Understanding the growing risks to software supply chains
- Explaining how software supply chain attacks work
- Examining how application security tools fail to detect supply chain threats

Chapter 1
Exploring the Landscape of Software Supply Chain Risks and Threats

Over the past two decades, malicious campaigns evolved from the loud but indiscriminate attacks of the early 2000s (such as the ILOVEYOU virus) to targeted attacks on public-facing, Internet-connected private and public sector organizations (such as the infamous hack of the retailer Target in 2013). But as digital transformation has taken root across global economies, and enterprise cyber protections have slowly improved, the focus of both cybercriminal and nation-backed hackers has shifted to software supply chains: the software and services that power modern organizations.

In this chapter, we take a look at why bad actors are attacking the supply chains, identify the vulnerabilities in the supply chain, and discuss why it's important that your organization address these issues.

Understanding Why Software Supply Chain Attacks Are a Growing Threat

These days, sophisticated cyber actors are targeting software integrated development environments (IDEs) used to write software, as well as continuous integration/continuous delivery (CI/CD) software that development teams and build systems use to create software artifacts.

WARNING

These are the plumbing of software development organizations and they have been largely beyond the purview of security teams and security vendors. But as attackers have been shifting their focus to the software supply chain, that can no longer be the case. Leaving IDEs and CI/CD software vulnerable gives attackers a way into your organization and its data. By one estimate, software supply chain attacks have increased by an average of over 700 percent annually since 2019.

Open source code

Open source software packages are frequent targets of malicious actors — and for good reason. An estimate by the Linux Foundation shows the percentage of open source code in modern applications is in the neighborhood of 70 to 90 percent (`linuxfoundation.org/blog/blog/a-summary-of-census-ii-open-source-software-application-libraries-the-world-depends-on`). That's a lot of open source code making its way into your organization, and, if a malicious actor has added their own code to it, can infect your software if left unchecked.

Here are some ways that bad actors can leverage what is otherwise useful open source code:

- **Find security holes.** For example, Log4Shell, an exploitable security hole in an Apache open source software library, Log4j, impacted tens of thousands of organizations globally and underscored the extent of organizations' exposure to vulnerabilities in common, open source packages.
- **Add harmful code to projects and repositories.** Anyone can contribute to open source, regardless of their intentions. That means bad actors can and will infiltrate legitimate open source projects or package repositories to push malicious code to developers and development organizations.

THE FINANCIAL REPERCUSSIONS

The financial repercussions associated with a software supply chain compromise can be enormous. SolarWinds reports that in a single quarter following its software supply chain breach, it spent at least $18 million on remediation and expected costs to be significant in future quarters (`reuters.com/technology/solarwinds-says-dealing-with-hack-fallout-cost-least-18-million-2021-04-13`). But it is not just the breached company that is impacted by a software supply chain compromise. A survey of SolarWinds customers revealed that on average their remediation costs were about $12 million (`techrepublic.com/article/cybersecurity-study-solarwinds-attack-cost-affected-companies-an-average-of-12-million`).

» **Exploit human error.** Threat actors also carry out low-skill *typosquatting* campaigns, the posting of "lookalike" malicious packages to an open source repository that mimic commonly used, benign open source packages.

Software build systems

Within the spectrum of software supply chain targets, the build system represents the Holy Grail for malicious actors. If a build system is compromised, an attacker has control over the final software that is signed and delivered to customers. Absent post-build security assessments, malicious behavior may be included in the final software package. If undetected, the final software package may be signed by the legitimate developer. This creates a false sense of security in users of the compromised software package. If the software in question is subsequently deployed via automatic updates to their users, it provides the attacker with a potentially huge number of victims.

An example of this kind of attack is the compromise of desktop client software developed by 3CX, a voice over IP provider, in 2023. Researchers at ReversingLabs analyzed the malicious 3CXDesktopApp and found evidence of a software build pipeline compromise that resulted in the addition of RC4 encrypted shellcode to the signature appendix of `d3dcompiler.dll`, a standard library

used with OpenJS Electron applications, an open source platform on which 3CXDesktopApp was built. Another standard Electron application module, `ffmpeg` was modified with code to extract and run the malicious content from the `d3dcompiler` file. A subsequent review by the maintainers of the open source libraries in question determined no malicious changes had been made, pointing to compromise of 3CX's build system in which the malicious functionality was added during the build and deployment process.

Increasing Software Supply Chain Risks from DevOps

The complexity of modern development pipelines also threatens the security of software supply chains, as shown in Figure 1-1. Many modern development teams rely on third-party and open source libraries, both of which carry risks and licensing requirements. We talk more about the development pipeline and how you can secure it in Chapter 2.

The risks from open source and third-party, commercial code include vulnerabilities that they carry as well as other risks, such as the use of code linked to sanctioned countries or firms, which may have political or financial repercussions for the software producer.

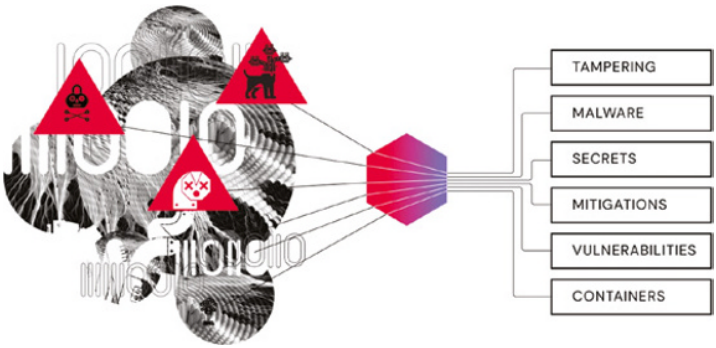

FIGURE 1-1: The complexity of modern software applications and the development pipelines adds risk to the security of deployed software. Courtesy of ReversingLabs.

Historically, the goals of security teams and development teams have been at odds, for obvious reasons:

- **Adding security slows the development process.** The development team, focused on releasing features and updates as quickly as possible, have adopted processes, such as CI/CD, to prioritize efficiency. Their concern isn't security.
- **Properly vetting software before release takes time.** Deploying software without proper vetting puts your organization at risk. Third-party software and platforms are necessary, but reduce your organization's ability to remediate risks. Breaches of a third-party software can lead to a compromise of your own organization.
- **The increasing number of secrets, use of containers, and other ephemeral systems, add to the challenges.** The accidental inclusion of API keys or code signing certificates in public code repositories or final build packages is another common problem. Exposure of these secrets allows anyone who finds them to access sensitive information via an API or to use an organization's signing certificate to sign a file of dubious origin.

Filling in Blind Spots with Comprehensive Security

Software supply chain attacks pose a big risk because current security product offerings are ill prepared to protect the software supply chain or establish a chain of trust connecting software producers with their suppliers and end-user organizations.

Existing application security tooling attempts to address this gap, but offerings such as static and dynamic application security testing (SAST and DAST) only work at one phase of the development pipeline, rather than offering a comprehensive solution.

Comprehensive software supply chain security requires investments at each stage of your development process: from the initial design of applications, to securing development infrastructure, to analysis of first-, second-, and third-party code and vetting completed software artifacts before they are shipped to end-user organizations.

REMEMBER

Comprehensive software supply chain security is a significant time and resource commitment for development teams. However, it is an investment that will pay dividends for your organization.

As noted by the National Institute of Standards and Technology (NIST) in its Special Publication 800-161 (`csrc.nist.gov/pubs/sp/800/161/r1/final`), establishing and sustaining a cybersecurity software supply chain risk management capability comes with benefits. They include:

- Enhancing your ability to detect, respond, and recover from supply chain compromises or other events that might result in significant business disruptions.
- Improving the operational and enterprise efficiencies of your organization.
- Weeding out low-quality suppliers and software, while promoting trustworthy suppliers and applications that are more likely to stave off attacks and provide high levels of availability.

IN THIS CHAPTER

- Making your development pipeline secure
- Assessing your cyber risk at different stages of the software development lifecycle
- Limiting cyber risk with frameworks and reproducible software builds

Chapter 2
Software Supply Chain Security for Modern Development Programs

Securing software development pipelines from malicious actors is a task that has historically been a low priority. But changes on multiple fronts give the task of shoring up software supply chain security new urgency.

In this chapter, we outline the steps your organization needs to take to secure your software development pipelines from end to end, encompassing both the work of your development organization and the end users who consume and deploy that software.

Securing Source Code

The first and most important element of your software supply chain security program is the source code itself.

REMEMBER

Attackers usually take advantage of weaknesses or vulnerabilities in the underlying source code that powers applications and services. Any deficiencies in the security of that code can give malicious actors the ability to leverage it in sophisticated attacks.

Common attacks that can lead to unauthorized or damaging modifications of source code include:

- The compromise of an active developer account by a malicious actor.
- The compromise of an inactive (or terminated) developer account by a malicious actor.
- Use of exposed secrets to compromise of services or data accessed by the application.
- Unauthorized changes made by an internal malicious actor (for example, a developer or engineer).
- The compromise of insecure development systems by an internal or external malicious actor.

A variety of measures can ensure that source code you are actively developing maintains its integrity:

- Have clearly defined feature descriptions and requirements.
- Use a defined and orderly process for code review and check-in.
- Harden development environments.
- Conduct both manual and automated reviews of code to identify unexplained features, behaviors, or code changes.
- Deploy static and dynamic application security testing (SAST and DAST) to identify code vulnerabilities and other risks.


TIP

Securing source code includes identifying and addressing software vulnerabilities. However, vulnerabilities are not the sum of software cyber risk. You also need to pay attention to cyber risks that do not involve exploitable vulnerabilities. Those include compromised or malicious software dependencies, code tampering, and compromises of your development infrastructure.

Securing Third-Party and Open Source Components

Your organization most likely relies heavily on commercial, third-party software components as well as open source software. These third-party and open source components provide

ready-made features and functionality, and greatly accelerate the development of new applications.

However, these third-party components and the many software dependencies that come with them introduce cyber risks that can be difficult to grasp, and open potential avenues of compromise. This isn't a new problem. As far back as 2003, for example, an unknown hacker added a backdoor to the Linux kernel.

More recently, malicious packages planted on open source repositories have become a common avenue of attack. For example, the North Korean state-sponsored hacking group known as Lazarus introduced Trojan horse malicious code into open source software, including apps such as PuTTY, KiTTY, TightVNC, and Sumatra PDF Reader for use in targeted attacks aimed at cyber espionage in 2022, according to Microsoft.

Securing third-party and open source components in your software supply chain requires you to employ a variety of measures to both prevent and detect suspicious or malicious code lurking in third party components. These include:

- Deploying software composition analysis (SCA) technology to scan your open source packages and updates for known vulnerabilities.
- Educating developers about common attack techniques such as typosquatting, repo-jacking, and dependency confusion attacks.
- Requiring detailed, machine-readable software bills of materials (SBOM) from suppliers of proprietary (closed source) software components.
- Scanning compiled binaries to detect suspicious, unauthorized, or unexplained behaviors and modifications.

Securing Your Continuous Integration/ Continuous Delivery System

The central role that continuous integration/continuous delivery (CI/CD) platforms play in DevOps environments, and the rapid pace of software delivery they enable, make CI/CD platforms a target for malicious actors.

Risk factors to CI/CD environments include both the disclosure of developer secrets, as seen in the CircleCI attack, as well as malicious tampering with CI/CD processes in ways that introduce malicious updates to developed code.

Securing your CI/CD process demands many of the same controls present throughout your development pipeline. That includes strict access control on CI/CD systems and the use of multifactor authentication, as well as close monitoring of your system logs and other data to detect suspicious or malicious activity.

Fortunately, NIST (National Institute of Standards and Technology) provides recommendations for securing your CI/CD pipeline. For CI pipelines, the NIST guidelines include:

- Securing software builds.
- Securing pull/push operations on repositories.
- Ensuring the integrity of evidence generation during software updates.
- Securing code commits.

For CD pipelines, the NIST guidelines advise you to implement controls including:

- Verifying that artifacts originate from a secure build process.
- Scanning software images for vulnerabilities prior to deployment.
- Removing secrets in code ready for deployment.

Finally, as threat actors increasingly target code-signing certificates to mask their malicious wares, your development team needs to strengthen its code signing processes. That includes the introduction of secure hardware for generating and storing private keys to prevent key compromises and centralizing code signing, rather than allowing distributed code signing across development groups.

Securing the Development, Build, and Release Processes

Traditional application security products broadly scan for vulnerabilities during the development, build, and release phases of software creation. Static application security testing (SAST) technology analyzes an application's source code, bytecode, or binary code for security vulnerabilities such as buffer overflows, SQL injection flaws, and other common vulnerabilities. Dynamic application security testing (DAST) involves runtime testing of applications to spot vulnerabilities such as cross-site scripting (XSS), SQL injection, and more. Software composition analysis (SCA) creates and maintains an inventory of open source and third-party components in a given codebase. It then scans those software modules looking for software vulnerabilities and license compliance issues.

WARNING

These traditional technologies are critical and necessary to secure code. However, they're insufficient to address all the possible software supply chain risks that you face, as successful attacks like those on SolarWinds and 3CX show. If you want to fully secure software supply chains, you must have both broad and deep detection of the threats that encompass development, build, and deployment processes.

Here, we talk about how you can secure each of these phases. In the next section, "Standardizing with Software Supply Chain Security Frameworks," we talk about the frameworks that can help your organization with its development, build, and release processes.

Development

Securing source code within your development pipeline is a multi-faceted endeavor that must consider both the design of the software and secure coding practices (which are mostly beyond the scope of this book) and then protections against some of the common attacks on source code. Here's how you can secure code during the development phase:

- **Staffing:** The development of secure code and software applications begins with hiring qualified software developers

for your team who are trained in secure software design and development. Once hired, these developers should receive regular secure development training as an ongoing part of their experience.

- **Design for security:** Securing your code extends to the implementation of "secure by design" principles at the very earliest stages of product conception, as well. That includes the use of memory-safe languages and thread-safe operations, the embrace of open design and attention to least privilege and the clear separation of roles for users. You should also employ threat modeling and attack surface analysis at the earliest stages of design and planning.
- **Secure developer systems:** Beyond these measures, your organization needs to apply a range of measures throughout the code development process that act as a check on malicious actors who seek to tamper with or alter source code. That starts with securing developer systems and development build configurations (more on that later).

TIP

Restricting the use of generalized productivity applications such as email, messaging, games, or other entertainment applications on your development systems is a way to reduce the risk of exposing these sensitive systems to attack. Keeping your development systems isolated as much as possible and devoid of non-development related applications is key to ensuring that such systems are not used to compromise development environments.

Build

Production build systems are where software deliverables are built from discrete software modules and components. Build environments typically encompass any system involved in the development and build process. That includes source code repositories, engineering workstations, build systems, as well as signing and deployment servers. These may run locally on physical servers managed by the development organization or (more commonly) on cloud-based platforms.

REMEMBER

Build systems are the crown jewels of software producers and a frequent target of software supply chain hackers.

In compromising build environments, malicious actors typically infiltrate developer networks first, possibly via a targeted attack

on an individual developer or privileged administrator. Network scanning and lateral movement enable the attackers to locate code repositories and build systems and assess them for exploitable security holes, such as misconfigurations or exploitable software vulnerabilities. By exploiting those weaknesses and gaining access to build systems and code repositories, malicious actors have almost unchecked ability to compromise the source code of the application or insert malicious libraries and code packages into the built application.

In the case of SunBurst, the hack of the SolarWinds Orion application, attackers believed to be affiliated with Russia compromised SolarWinds build environment and inserted malicious code directly into the Orion application. Their long-term access to the build server enabled the attackers to observe SolarWinds developers, and then blend their malicious code additions with the sanctioned Orion code. That code, which executed a backdoor that gave malicious attackers access to SolarWinds customer environments, was built with the regular Orion software, and delivered as a signed Orion software update, thereby avoiding detection.

TIP

To protect your organization's build environments, ensure that all systems making up that environment are:

- Protected with strong, multifactor authentication and user least privilege configuration.
- Configured to log all user/account access to build systems.
- Segregated from remote access or access from corporate and business networks and systems.
- Continually monitored for malicious activity including data exfiltration and malicious command and control (C2) communications.
- Monitored and audited to identify unexplained code additions or modifications or other unusual behavior.
- Regularly audited to identify rogue, abandoned, or over privileged user and service accounts.

Release

In addition to securing developer systems and build environments, you need to secure the distribution systems you use to deliver software packages to your customers.

Software packages include a compiled and signed software application or software update, as well as package metadata such as the software version number. While the vast majority of software produced and distributed is free of malicious modifications or other evidence of tampering, incidents such as the hacks of SolarWinds and Voice over IP (VoIP) vendor 3CX show that compromises of development pipelines and build systems may escape notice but leave telltale signs within the release package itself.


WARNING

Even software releases that haven't been compromised may still contain information that exposes you to risks. For example, compiled software releases may contain confidential information such as hard coded credentials, code signing certificates, or personal data.

Protections for your software release packages include:

- Applying complex binary analysis and software composition analysis to identify potential issues in release packages including unexplained behaviors, exploitable vulnerabilities, or software license issues. (See Chapter 3 for more details on complex binary analysis.)
- Applying continuous security monitoring to package repositories to prevent compromises.
- Generating and distributing a detailed, machine-readable SBOM for all published software.
- Cryptographically signing software packages and verifying signatures using a package manager.
- Applying appropriate transport layer security to software distribution systems in accordance with NIST SP 800-52 rev. 2.

Standardizing with Software Supply Chain Security Frameworks

The best way to begin working towards identifying anomalies in your software supply chain is to standardize your development processes — in particular, your build process. The following

software supply chain security specifications can help guide that process:

- Supply Chain Levels for Software Artifacts (SLSA) are Linux Foundation-backed guidelines for supply chain security.
- In-toto is a framework for securing software supply chains hosted at the Cloud Native Computing Foundation. The In-toto specification provides a generalized workflow to secure different steps in a software supply chain.

Supply Chain Levels for Software Artifacts (SLSA)

Supply Chain Levels for Software Artifacts (SLSA) breaks down securing the software supply chain into manageable milestones with an end goal of standardized, tightly secured build processes. This level of secure, verifiable build is also known as *provenance*.

REMEMBER

Making your build process standardized and reproducible also makes it auditable, which is essential if a security incident occurs.

In its simplest form, SLSA provides protections against tampering with software during the development, build, and release phases, akin to the kinds of quality checks that exist in other industries (such as food production or manufacturing).

The SLSA framework encourages the use of tamper-resistant evidence at each step of the software production process. This gives software producers and consumers assurance that a finished software product contains only the intended software "ingredients" (first party, open source, and third party) and that those ingredients have not been altered or tampered with. The risks that SLSA addresses specifically are:

- **Code modification:** It adds a tamper-evident seal to the code after source control.
- **Uploaded artifacts that were not built by the expected CI/CD platform:** Software artifacts are marked with a factory stamp that shows which build platform created it.
- **Threats against the build platform:** SLSA provides best practices for build platform services.

TECHNICAL STUFF

SLSA VERSUS SBOM

SLSA is complementary with another fixture of software supply chain security: SBOMs (Software Bills of Materials), which provide a list of ingredients for software artifacts. SLSAs provide granular instructions on how those ingredients can be combined and used. While SBOMs help determine whether a particular software component (say Log4j) is included in a particular build, they do not identify threats or risks linked to software — such as compromised or malicious software dependencies. SLSA helps close that loop: ensuring that proper handling practices are used at each step of the software production process.

The framework has ascending levels representing the use of approved security practices. Higher levels signify more security guarantees against software supply chain threats. Lower SLSA levels suggest only modest security guarantees, with SLSA 0 as the starting point (no steps taken to secure software supply chains). Currently, SLSA recognizes Levels 1–3, though higher levels of compliance are planned for the various security tracks.

In-toto

The SSCS platform relies on signed seals to guarantee the integrity of software. But where do those seals of approval come from? That's where In-toto comes in. The In-toto framework focuses on standardizing software supply chain provenance. A key element of the framework is a metadata system that allows actions to be verified, ensuring that the appropriate entity was responsible for a given change, and that the code in question has not been tampered with.

TECHNICAL STUFF

Specifically, In-toto provides APIs that can be used with tools like Sigstore cosign, the SLSA GitHub Generator, and the In-toto Jenkins plugin to generate SLSA Provenance metadata.

WARNING

Provenance does not make your organization foolproof. The In-toto metadata framework does not alert you to insider threats to your software supply chain. But establishing a chain of responsibility for each action taken across a software supply chain can help ensure no unwanted changes are made to a codebase.

Developing Reproducible Processes

A key indicator of software supply chain integrity is the ability to use reproducible processes to create software artifacts. By following a consistent process for building, testing, and releasing software, organizations can ensure their software has not been tampered with or otherwise modified in ways that may introduce cyber risks.

One approach to ensuring that your organization is following reproducible processes is through a *verified reproducible build*. Reproducible builds allow software producers to verify the integrity of their CI/CD build systems by comparing the behaviors of software artifacts that were constructed in totally separate build environments. The process involves the use of two or more independent build platforms to generate software artifacts, as shown in Figure 2-1. Those different platforms are then used to corroborate the provenance of a build.

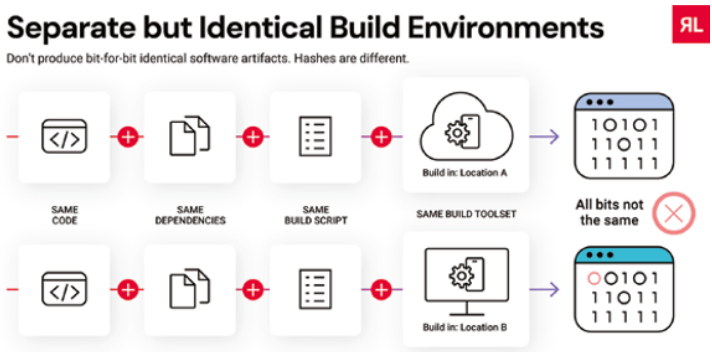

FIGURE 2-1: Use multiple independent platforms to establish a build's provenance. Courtesy of ReversingLabs.

Verified reproducible builds address the risk of build environment tampering by adversaries of the kind observed in the attack on SolarWinds Orion. Development organizations maintain physically separate, isolated, and identical build environments to generate separate binaries that are then compared statically in terms of their behaviors, rather than based on a signature hash (which would not be identical, in any case). Differences in behaviors between the two artifacts warrant investigation for possible tampering or compromise within a build environment.

IN THIS CHAPTER

- Developing a successful third-party risk management program
- Keeping third-party software secure throughout its lifecycle
- Moving beyond traditional AppSec controls for complex binary analysis

Chapter 3
Managing Third-Party Commercial Software Risk

Third-party commercial and open source code is increasingly exploited by malicious actors (which we discuss in Chapter 1). If your organization is like most, it is dependent on software suppliers, which deliver products and services using proprietary, third-party and open source code.

WARNING

And unfortunately, many organizations lag in shoring up their supplier base, even though the frequency and sophistication of attacks by malicious actors grows. A survey by ReversingLabs reveals that despite that 98 percent of surveyed organizations recognize that software supply chain issues pose a significant business risk, only six out of ten feel their software supply chain defenses are up to the task of warding off such attacks (`https://www.reversinglabs.com/blog/tools-gap-leaves-the-software-supply-chain-exposed-why-you-need-to-upgrade-your-application-security`). This gap in protection is contributing to a major uninsured risk for organizations. And, depending on circumstances, it

may create catastrophic damages for organizations and even lead to legal and regulatory actions with implicit fiduciary responsibility for executives.

Emerging regulations like the E.U.'s Digital Operational Resilience Act (DORA) and Cyber Resilience Act will require that your organization demonstrate that all software packages, containers, and updates that it consumes are verified for the absence of malware, patch-mandated vulnerabilities, and address other security threats.

In this chapter, we show you how to develop a third-party risk management (TPRM) program to assess the security risks of third-party software your organization consumes.

Sharing the Load: Managing Software Supply Chain Risks

One way to avoid critical risk management activities from slipping through the cracks is to clearly define the roles and responsibilities within the organization to identify, detect, respond, and recover from software supply chain security issues that may arise throughout the software lifecycle. This includes the acquisition, deployment, maintenance, and ongoing monitoring of software. Table 3-1 shows a high-level RACI (responsible, accountable, consulted, and informed) matrix, which demonstrates how your organization might distribute responsibilities across business functions for managing the security risk presented by third-party software.

TABLE 3-1 Software Use Lifecycle Stages

	Acquisition	Deployment	Maintenance	Monitoring
Procurement	A			
TPRM	R	C	C	I
Application Security	C	A	I	
IT Operations		R	A/R	I
Threat Intel				R
SOC				A

Key: **R** = *Responsible;* **A** = *Accountable;* **C** = *Consulted;* **I** = *Informed.*

Figure 3-1 shows the key stages across the software development and consumption lifecycle, outlining the diversity of stakeholders and stages that require security risk monitoring and management coverage.

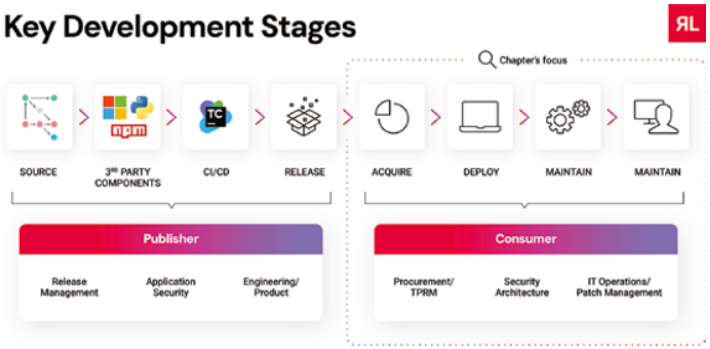

FIGURE 3-1: The key stages of development and lifecycle of software that require security. Courtesy of ReversingLabs.

We go into more detail about each of these lifecycle stages later in this chapter in the "Achieving Software Assurance Throughout the Lifecycle" section.

Aligning Software Supply Chain Security Policies to Internal Standards

A successful TPRM program starts with clearly defining what security requirements your organization expects third parties to uphold.

REMEMBER

Make sure the requirements are aligned with your organization's internal security standards, as well as any legislative and regulatory requirements that your organization and suppliers operate within.

TIP

Given the recent emergence of software supply chain security risk, not all organizations have formally defined these standards. In this case, your organization should update existing documentation to include requirements that allow the organization to continually assess the risk presented by third-party software, throughout its

lifecycle. These policy requirements should be written at a high level, to ensure multiple testing methodologies can be leveraged to achieve the objective of overall software supply chain risk management. Additionally, it should be backed by industry standard practices, to support rationale for implementation.

These control requirements, included within industry standard framework NIST CSF 2.0, can be leveraged as supporting rationale for proposed modification to existing corporate policy:

- **GV.SC-07:** The risks posed by a supplier, their products and services, and other third parties are identified, recorded, prioritized, assessed, responded to, and monitored over the course of the relationship (formerly ID.SC-02, ID.SC-04).
- **GV.SC-09:** Supply chain security practices are integrated into cybersecurity and enterprise risk management programs, and their performance is monitored throughout the technology product and service lifecycle.

Once established and clearly communicated, these requirements must be continually applied to ensure that compliance by your organization and its suppliers is tracked and reported. By ensuring that only these requirements are tested, your organization can avoid testing scope creep, which may detract from an efficient and effective risk management strategy.

Understanding what software to test first

Your organization most likely uses hundreds or even thousands of software products. Given the size of this software supply chain, it's important to identify and analyze which of these products provide critical business services that keep your organization running, including any third-party software.

When determining how to segment your software estate based on risk, consider a variety of risk elements, including

- Criticality of software to business operations
- Breadth of deployment of software
- Location that software is deployed or integrated
- Connectivity and privilege required for installation

- Level of current and historical investment and maintenance for software

Applying these considerations to business-critical software and services enables your organization to quickly identify the IT assets and services that have the most financial and operational impact if breached via a software supply chain attack.

Shifting away from a "one size fits all" mentality

Once you identify a target population of software and suppliers for monitoring, evaluate your overall TPRM assessment methodology to make sure it addresses the unique risk presented by the specific supplier type. Comparing the security maturity of two suppliers that are inherently different may negatively influence procurement if the comparison is built off a correlation with no significance.


TIP

One way to ensure that your TPRM assessments are properly calibrated to risk is to shift away from a reliance on questionnaires. These primarily focus on business process-oriented controls such as policy and procedure governance. More effective measures provide product- and service-specific assessments that address the root of software supply chain security risk. We talk more about this in the later section, "Going Beyond AppSec Controls to Evaluate Third-Party Risk."

Evaluating the Security of Third-Party Software

Targeted assessments of third-party software, including Commercial Off the Shelf (COTS) products, should include inquiries that are related to the unique functionalities of the software in question. They include:

- **Obtaining a software bill of materials (SBOM).** SBOMs list the software "ingredients" of applications and services, including open source and third-party software dependencies, which enables streamlining incident response efforts in the event of emergent software supply chain risks or threats.

- **Scanning to identify malware and software tampering.** Any embedded malware within software poses business risks or evidence of a malicious attack. Unexplained behaviors or modifications may indicate that underlying code has been altered.
- **Identifying and tracking package behaviors across release versions.** This enables you to spot unexplained changes in behavior that may indicate a compromise of the supplier's build system.
- **Verifying digital signature integrity.** Tampering with digital signatures for open source and proprietary packages may indicate malicious tampering.

Achieving Software Assurance Throughout the Lifecycle

The security risks posed by software suppliers are not stagnant. As such, it's important that your organization broaden the scope of software assurance activities beyond checking it one time during procurement. It is necessary to continually assess your software throughout its lifecycle within your organization.


TECHNICAL STUFF

This concept of continuous supplier monitoring is captured in many best practices guidelines, including NIST 800-161 Control CA-2, the European Banking Association's (EBA's) Outsourcing Guidelines, DORA, and the Biden Administration's Executive Order 14028.

A simple way of achieving continuous monitoring is by deploying security controls at each stage of the software consumption lifecycle. This section defines the stages of the lifecycle, and the stakeholder groups responsible for ensuring security assurance at each stage. Figure 3-2 outlines each stage.

Stage 1: Acquisition

As your organization researches prospective software vendors, you must consider the cybersecurity risk that the software will expose your organization to. The *acquisition* stage represents the pre-contract assessment activities that you can perform to identify cyber risks that might be introduced if you use the software.

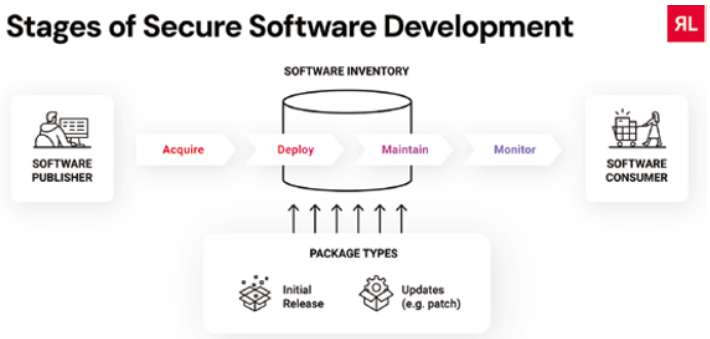

FIGURE 3-2: Each stage of a software's lifecycle needs to be secure. Courtesy of ReversingLabs.

TIP

Ask the vendor for access to the software binary and conduct security testing of that binary.

To properly manage the cyber risks that go along with software acquisition, consider involving the following groups and functions:

- **Procurement:** Responsible for defining, communicating, and managing vendor research and qualification, and ensuring the results from the software analysis are considered throughout the selection and onboarding processes.
- **Legal:** Responsible for ensuring the ultimate agreement with the vendor includes contractual terms that allow for the continuous analysis of software (such as releases, patches, or hotfixes).
- **Third-party risk management:** Responsible for obtaining the software package for analysis as well as communicating results, including key findings to the product owner, senior management, and vendor.
- **Product/business owner:** Responsible for approving or rejecting software as well as any risks identified during assessment that cannot be fixed by the vendor or otherwise covered by compensating controls prior to product go-live.

Stage 2: Deployment

The *deployment* stage represents the post-contract operational activities that are required to securely integrate a third-party software solution.

To properly manage the cyber risks that go along with software deployment, consider involving the following groups and functions:

- **Security architecture:** Responsible for determining the compensating technology and controls available to assist in the mitigation of risks identified during analysis.
- **IT operations/IT service management (ITSM):** Responsible for the secure deployment of the software binary into the production environment using any mitigation controls identified during the analysis.

Stage 3: Maintenance

Commercial software solutions inevitably change over time as bugs are fixed and new features are released. Of course, these *maintenance* activities introduce risk.

REMEMBER

New releases, bug fixes, and patches always have the potential to introduce risk. This is why risk assessment isn't a one-time activity, but continuous.

To properly manage the cyber risks that go along with software maintenance, consider involving the following groups and functions:

- **Third-party risk management:** Responsible for obtaining new software release binaries for analysis by software supply chain security tools, as well as communicating results, including key findings to product owners, senior management, and vendors.
- **Product/business owner:** Responsible for approving or rejecting new release versions as well as risks identified that cannot be fixed by the vendor or otherwise covered by compensating controls prior to deployment of a new release version.
- **IT operations/ITSM:** Responsible for updating software inventory with approved release versions and revoking access to legacy versions already deployed in production as needed.

Stage 4: Monitoring

The *monitoring* stage provides proactive identification of cyber risks, and vulnerabilities in the software supply chain, and supports efficient and effective incident response efforts. To properly monitor the security of your software inventory, consider involving the following groups and functions:

- **Threat intelligence:** Responsible for monitoring threat intelligence feeds for newly emerging threats to the software supply chain and notifying the SOC where potential impact lies.
- **Security operations center (SOC):** Responsible for triaging, investigating, and analyzing emerging threats. Responsible for notifying IT operations/ITSM to revoke impacted software where necessary.

Going Beyond AppSec Controls to Evaluate Third-Party Risk

Dozens of tools and platforms in the Application Security Testing (AST) marketplace promise to identify, mitigate, and manage risk throughout the entire software development lifecycle. However, hacks of SolarWinds, 3CX, MOVEIt, and others show that these technologies are not up to the task of identifying and preventing all software supply chain threats.

Static application security testing (SAST) technologies, for example, enable software producers to analyze software source code and identify embedded vulnerabilities. However, SAST requires access to raw source code. Software publishers that rely on prebuilt, third-party software modules don't have access to raw, uncompiled source code, nor is it available to enterprise software buyers.

Software composition analysis (SCA) technologies support the identification of risks in open source software. However, SCA typically overlooks the sea of licensed, commercial software libraries and other closed source components that may also present risks to the security of an application.

WARNING

The result is that existing application security solutions limit your ability to evaluate the security risk of the COTS software your organization uses.

Compensating controls and technology shortfalls

Aside from AST tooling, organizations rely on a number of alternative compensating controls and technology to mitigate the security risk of COTS software. These approaches similarly leave gaps in the protections they provide:

- **Vendor questionnaires:** This traditional "pen and paper" approach to managing vendor security risk provides limited levels of assurance and relies on trusting that the vendor is fully aware of its security risk and has been truthful and complete in its self-attestation statements.
- **Anti-virus/anti-malware:** Anti-virus and anti-malware technologies were not designed to detect malicious code lurking within large and complex software packages that sport valid signatures and that otherwise appear normal. In fact, anti-malware technologies often maintain file size limitations that impact their ability to provide analysis coverage for modern COTS applications.
- **Sandbox/virtual environments:** Sandboxes are resource intensive, and can be easily evaded using malicious techniques. For example: Malware authors employ time-based payload execution delay methods, where malicious code is embedded and programmed to cause damage at a later time.
- **Security rating services:** Although these services provide valuable insights into the general security posture of the vendor, they overlook the security risk posed by the actual product (for example, COTS software package) that is being consumed by the customer.

Complex binary analysis: A final exam for software

How can your organization evaluate the security risk presented by externally procured software? One way is through the greater use of complex binary analysis as a kind of final exam to spot threats before software is released to downstream consumers.

Complex binary analysis is the process of recursively unpacking large and complex software binary files, extracting metadata from embedded objects (such as executables, libraries, and icons), and analyzing the contents to uncover internal threat indicators. Figure 3-3 shows how this process typically works. In complex binary analysis, the extracted files are not executed, which means that a detailed analysis can be performed in an efficient and cost-effective manner.

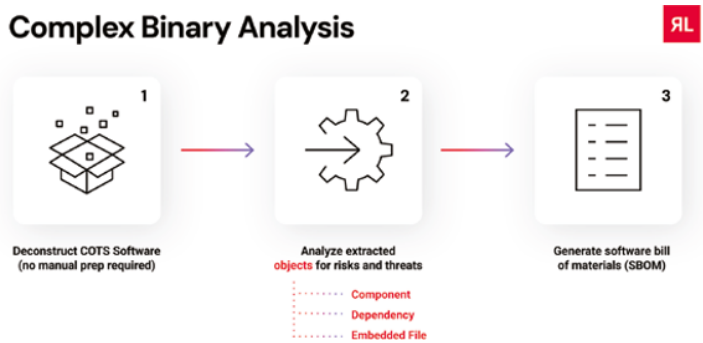

FIGURE 3-3: The process of complex binary analysis. Courtesy of ReversingLabs.

Complex binary analysis technology allows your organization to deconstruct and analyze both open source and closed source, commercial software packages. And complex binary analysis does not have constraints like requiring access to the source code; requiring the cooperation of the software vendor; or needing to engage in manual testing. This enables both software producers and end-user organizations to obtain the assurances they need through an independent verification of trust.

Although an approach such as complex binary analysis is a departure from traditional approaches to third-party software risk management, it is essential to gain targeted risk insights at the product level, where malicious actors are implanting threats. Achieving these insights in an automated manner enables your organization to securely onboard third-party software at the speed your business operates.

IN THIS CHAPTER

- Understanding how threat intelligence fits into a threat-hunting strategy
- Conducting threat hunting for known and unknown threats
- Securing the software development pipeline with threat hunting

Chapter 4
Hunting for Threats in Your Software Supply Chain

As the sophistication of cyber-attacks and malicious cyber actors increased in the last decade, threat hunting emerged as an indispensable tool for organizations. *Threat hunting* describes the efforts to identify unknown and un-remediated cyberthreats operating within your organization's IT environment. In contrast to "incident response," threat hunting is proactive, rather than reactive, and involves disciplined searches for threats, including evidence of compromises on any systems, networks, and infrastructure.

Historically, threat hunting has focused on identifying evidence of traditional network compromises and threat actors, such as the presence of malware inside email attachments, evidence of communication to malicious command and control (C2) networks from end user systems, and so on. Today, threat-hunting activities are shifting "left" to development pipelines, third-party software, and software stored in the cloud, reflecting the kinds of systems targeted by threat actors.

Threat hunting today is a critical capability to thwart software supply chain attacks. In this chapter, we examine the steps you

need to take to develop a robust threat-hunting operation that can bolster the security of your software supply chain.

Using Threat Intelligence

Just as cyber threat-hunting capabilities are key to securing modern software supply chains, cyber threat intelligence is a key component of effective software supply chain threat-hunting operations.

What is threat intelligence?

The term *threat intelligence* refers to the aggregation and application of comprehensive threat data. Threat intelligence falls into two categories:

- **Atomic indicators:** These are discrete pieces of threat intelligence, such as cryptographic hash values associated with known malware. Atomic indicators are useful for intelligence-based hunting.
- **Tactics, Techniques, and Procedures (TTPs):** These are the more holistic aspects of a malicious actor's behavior. A *tactic* is a high-level description of attacker behavior; a *technique* is a detailed description of behavior in the context of a tactic; a *procedure* is a low level, detailed description in the context of a technique.

Both types of threat intelligence are useful in *hypothesis hunting*, the process of working backwards from indicators of attack (IoAs) or indicators of compromise (IoCs) to identify specific threat actors or groups who may be operating within a compromised IT environment.


TIP

Use frameworks such as MITRE's ATT&CK to map TTPs and other forms of threat intelligence. This framework connects threats with known threat actors and recommends mitigations.

Gathering intelligence

As with traditional IT security assessments, a good way to start accessing your software supply chain security is by looking at your organization's *attack surface* (the areas that are vulnerable to an attack). This includes an understanding of your critical IT

assets and possible avenues along which cyber-attacks on your software supply chain may travel. This process starts with identifying your high-risk development and software supply chain assets: the places where highly sensitive data is stored; that perform critical functions; that are exposed to the Internet; and systems that are critical to your business operations.

REMEMBER

When finding your high-risk assets, don't forget about anything that touches them. If one of these systems is vulnerable, your high-risk asset is vulnerable, too, and a likely target for threat actors.

Identifying high-risk assets

Securing your software pipeline from end to end is essential to prevent compromises. Consider these targets of software supply chain attackers:

- **Privileged employee workstations:** Many hacks begin with the compromise of a single developer or employee workstation. Harden endpoints used by staff, and monitor both privileged accounts and workstations for suspicious activity.
- **Open source and third-party software packages:** Scanning open source packages and closed source, third-party components for malicious or suspicious code, dependencies, and behaviors is key to preventing supply chain attacks.
- **Integrated development environments (IDEs):** Malicious actors aren't just focused on seeding open source package managers with malicious code. IDEs are also fair game, with campaigns designed to trick developers into installing malicious extensions into their IDE.
- **Continuous integration/continuous delivery (CI/CD) software:** Malicious actors use their access to CI/CD environments to exfiltrate sensitive data such as credentials and access tokens, or otherwise tamper with the contents of targeted software packages.
- **Build servers:** Malicious actors that are able to compromise build can inject malicious code and functions into signed software artifacts.
- **Final builds:** The goal of software supply chain attackers is to weaponize the software that will be installed by end-user organizations. Examine all final, signed builds for unexplained behaviors, features, and characteristics before releasing them.

Sources of threat intelligence

Now that you know where to look, the next question is "what should we look for?" Evidence of suspicious or malicious activity related to key development assets can take many forms. Here are some examples to consider:

- **Exploitable software vulnerabilities:** Software flaws are a frequently used avenue of attack. Scan code to identify known and exploitable vulnerabilities.
- **File rot:** Old and outdated files or code, over time, accumulate software vulnerabilities or use outdated security methods that attackers break easily.
- **Malicious or suspicious code:** Malicious or suspicious (obfuscated) code, backdoors, or other unexplained functionality are common to many software supply chain attacks.
- **Behavioral anomalies:** Unexplained and suspicious behaviors in compiled binaries are a sign of possible unauthorized tampering with sanctioned code.
- **Third-party dependencies:** Attacks often leverage third-party dependencies to sneak malicious or vulnerable code into otherwise hardened applications.
- **Open source (OSINT) and dark web intelligence:** Malicious software supply chain campaigns can often be stopped in the planning stages via careful monitoring of both open source intelligence and the cyber underground where evidence of planning or active campaigns against targeted firms first turn up.

Proactively Hunting Threats

Whether focused on traditional IT environments or development pipelines, threat hunting involves identifying both known and unknown threats. For example, the revelation of a new zero-day vulnerability in critical software that is being actively exploited, such as the infamous EternalBlue flaw in Microsoft's implementation of Server Message Bus, might prompt a threat-hunting expedition within sensitive environments to look for evidence of malicious actors leveraging that flaw.

TIP

Don't wait for a specific threat or risk. Threat hunting can take place independently regardless of a known threat or risk. Be proactive and look for evidence of compromises involving known threats or in relation to sensitive IT assets.

WARNING

Most established threat-hunting guides are focused on defending against an older generation of common threats to IT networks and assets. Software development, build and release infrastructure, and processes are not typically discussed in those guides. But the increasing frequency and scope of malicious software supply chain campaigns make it clear that your development environment and pipeline are part of the cyber terrain, where hackers are lingering, waiting for a way into your organization.

How does your organization go about threat hunting within your software supply chain? Here are some suggestions.

Looking for evidence of malicious activity

The first stop for any threat-hunting team — whether focused on traditional network compromises or software supply chain hacks — is to look for evidence of malicious activity within sensitive and protected environments. These include looking for and finding evidence of:

- **Reconnaissance:** Attacks on development environments may start with efforts to identify and target privileged developer and administrator accounts within target organizations or trusted suppliers to those organizations. Work with your development team and trusted third parties to find evidence of malicious actors' reconnaissance or activity, including unsolicited email and social media messages, the transmission of suspicious files or web links, and so on.
- **Initial access:** Successful software supply chain compromises begin with attackers gaining a foothold on sensitive or privileged IT assets that have direct or remote access to protected assets and networks. Scanning for unusual patterns of user access, unknown accounts, unexplained network activity (for example, lateral movement), or the download and execution of unknown or unexplained files can turn up evidence that an attacker may have gained access to your development environment.

- **Command and control (C&C) infrastructure and exfiltration:** Software supply chain attackers almost certainly need to communicate outside your environment at some point during their compromise. That could include communications to and from malicious command and control infrastructure, downloading second- or third-stage malware, or the exfiltration of sensitive information. Scanning logs and monitoring for unexplained encrypted or unencrypted communications into or out of your environment is one way to spot an intrusion.

Using SBOMs to understand software composition

Before you can identify threats in your software supply chain, it is important to understand the composition of the software applications and services that your organization is trying to defend. Generating software bills of materials (SBOMs) and requiring them from your software suppliers are critical first steps in constructing a software supply chain threat-hunting plan. SBOMs come in many different formats, but most include standard elements such as the software component name, the publisher's name, the component version, filenames, software licenses, and any software dependencies.

By providing a list of third-party and open source software, statically linked packages, and internally developed software, SBOMs provide a roadmap for threat-hunting teams to assess software supply chain risks and identify possible avenues of compromise. With detailed SBOMs, threat-hunting teams can begin the work of identifying and assessing open source and commercial, third-party code dependencies and identify avenues of attack and exploitation that a malicious actor might use to compromise your organization's software supply chain.

Identifying evidence of compromise

With a solid understanding of your development environment, and the makeup of the applications and services your threat-hunting team is defending, you can begin the process of looking for evidence of emerging, ongoing, or past compromises of your software supply chain. For example, your software supply chain threat-hunt team may:

- **Leverage threat intelligence.** High-quality threat intelligence feeds can help point your threat-hunting team in the direction of accounts or IT assets that are on the radar of malicious actors and that may be targeted in attacks.
- **Look for evidence of exploitation of known software vulnerabilities.** Identify exploitable software holes in your current application code or development infrastructure and, if found, look for evidence of active or past attempts to exploit those holes. A positive finding could be evidence of a security breach.
- **Scan software binaries for behavioral anomalies.** Complex binary analysis (which we discuss in Chapter 3) can identify behavioral anomalies (you can compare the anomaly to the known version), which may indicate the presence of backdoors or other unexplained functionality.
- **Assess open source and third-party dependencies.** For open source code, verifying the integrity of open source packages and scanning for the presence of obfuscated code, call outs to unknown IP addresses, expired certificates, or other suspicious features can identify possible risks. For closed source, third-party code, requesting detailed SBOMs from suppliers, verifying code signatures, and conducting complex binary analysis on compiled and signed software artifacts can help identify risks.

Hunting for Developer Threats

TIP

Part of securing your development pipeline is making sure your engineers are highly skilled and trained properly. As we noted, a mature threat-hunting capability is critical to maintaining a secure software development pipeline and staving off successful software supply chain attacks. Such a capability requires your organization to remain vigilant to both sophisticated and unsophisticated, developer-centric threats. These include being attentive to common "ease of development" features like developers' use of temporary backdoors that may find their way into production code. It also demands attention to more subtle manipulations by a malicious insider that has access to raw, uncompiled code or other development infrastructure.

To address such threats, you should:

- **Prioritize reviews of critical code.** Pay particular attention to code that employs elevated privileges, accesses sensitive resources, or implements cryptographic functions.
- **Conduct automated static and dynamic testing.** Test newly checked-in code for known vulnerabilities.
- **Map newly created code.** Identify new code and look for the inclusion features and behavior that may indicate malicious activity.
- **Implement authenticated code check-ins.** Guard against compromises of development systems by requiring developers to authenticate prior to checking in code.
- **Implement strong access control.** Secure access to development-related infrastructure with strong and multifactor authentication as well as user policies that restrict access to sensitive data and features.
- **Implement continuous monitoring of developer systems.** You also need to monitor for suspicious activity that may indicate the presence of an insider threat, or the compromise of a privileged user account.
- **Assess the final build.** Before releasing software to customers, assess your final build artifact for the presence of embedded threats such as malware and suspicious changes in software behaviors. These can indicate that your software supply chain has been compromised.

IN THIS CHAPTER

- Building a successful software supply chain security program
- Protecting software through its lifecycle

Chapter 5
Ten Tips for a Successful Software Supply Chain Security Program

Building a robust software supply chain security program requires time to plan and realize, as well as getting buy-in from stakeholders up and down the organizational chart. As daunting as that may sound, "the journey of a thousand miles starts with a single step," as the saying goes. With that in mind, here are ten tips for building a successful software supply chain security program at your organization.

Broaden Your AppSec Program

The application security testing status quo is focused narrowly on vulnerability identification and management as well as code quality. But that leaves software producers short of the goal line in the larger context of software supply chain security. To stand up a successful software supply chain security program, broaden the focus of your application security testing to incorporate threats such as code tampering or the introduction of malware into first-party, open source, or commercial third-party code.

Secure and Protect Development Infrastructure

Prioritize the security of your development infrastructure, from developer workstations to build and release servers. Tightly managing access to development assets through robust access control and least privilege policies can keep malicious actors from getting a foothold on development systems. Close monitoring of the security configurations and network activity associated with developer systems, continuous integration and continuous delivery/deployment (CI/CD) systems, and build servers is key. Endpoint detection and response (EDR) software, threat monitoring, and security information and event management (SIEM) technologies can help by preventing local compromises from spreading within organizations and affecting development environments.

Beware of File Rot

File rot is the presence of old and outdated files or code in applications. As time passes, and absent updates, code tends to accumulate vulnerabilities and inefficiencies, making file rot an implicit risk to application security, not to mention application availability and performance.

REMEMBER

While old files are not necessarily evidence of file insecurity, the file rot phenomenon suggests that old and outdated files should be a focus for attention and close monitoring by your application security and development teams.

Give Your Software Package a Final Exam

More and more production and customer environments are being hacked by malicious actors that place back doors and other code in software updates from trusted suppliers. This makes it clear that traditional forms of third-party risk assessment (such as voluntary vendor compliance surveys and questionnaires) can't be the only factors in deciding whether software is safe. One supplement

to vendor questionnaires is to use complex binary scanning and analysis that verifies the integrity of a compiled package.

Enhance Your Risk Analysis

The exponential growth of threats in open source repositories in recent years highlights the risk of development organizations' dependence on open source and third-party, commercial software. To address this risk, you need to apply a consistent level of scrutiny to software provided by trusted internal teams, third-party contractors, and commercial software suppliers.

REMEMBER

You must do more than simply analyze open source software components or generate a software bill of materials (SBOM). Employ comprehensive software supply chain risk analysis, to ensure that remediation efforts are effective.

Use Threat Intelligence

Incorporating threat intelligence into your software supply chain program enhances your ability to monitor development activity for indicators like the presence or execution of a known-bad binary or communications to and from a malicious domain or other known command and control (C2) infrastructure.

Emphasize Secure Development Practices

The security of software supply chains rests on the foundation of secure code. The best way to raise the quality and security of application code is by emphasizing secure design principles at the earliest stages of product conception and creation. That means employing threat modeling and attack surface analysis during product design and planning. Development practices should also reduce risks, for example through the use of memory safe languages; thread safe operations; attention to least privilege concepts and user role separation; as well as the use of encryption for data at rest and in transit.

Pay Attention to Open Source Risks

Between 70 and 90 percent of the code in modern applications is open source, by one estimate. However, the use of open source software comes with risks: providing malicious actors with an avenue into development environments. Securing your software supply chain means developing a detailed understanding of your use of- and dependence on open source modules. Tracking open source licenses and use within your organization as well as identifying and tracking vulnerable and/or compromised open source components makes it easier to stay on top of these risks.

Invest in Proactive Threat Hunting

Government guidelines require you to expand the scope of your defensive measures beyond vulnerability discovery and patching. Invest in tools that can identify malicious components hiding in open source, commercial, and third-party software packages.


TIP

You can identify those risks with complex binary analysis and file reputation tools. Also, apply open source YARA rules internally to detect malicious software components such as malware downloaders, viruses, trojans, exploits, and ransomware within your IT or development environments.

Monitor and Track Development Secrets

The increasing number of development secrets poses a risk to your development organization, especially when coupled with complex deployment pipelines and ephemeral systems such as virtual containers. The ability to monitor your source code and development pipeline for the presence of developer secrets such as coded credentials and API keys is critical to preventing compromises of your development organization.

Defend your software supply chain against malicious actors

Organizations that don't prioritize software supply chain security leave themselves vulnerable to malicious actors. Hackers today look to exploit weaknesses in software development pipelines to insert malware and launch attacks against software producers and the customers that deploy their software. Ensuring the software you sell or buy is free of malware and tampering is critical.

Inside, you'll find out how to secure your entire software supply chain and its first-party, third-party, or open source software, and how to be proactive with a threat-hunting program to thwart software supply chain attacks.

Inside…

- Understand why securing the software supply chain is important
- Secure the development pipeline end to end
- Protect software through its lifecycle
- Ensure the software you deploy across your organization is safe
- Develop a robust threat-hunting program
- Perform complex binary analysis on final builds

Paul F. Roberts is the Director of Editorial and Content at ReversingLabs, with 20 years' experience covering the cybersecurity space. **Charlie Jones** is a Director of Product Management and a subject matter expert in supply chain security.

ISBN: 978-1-119-81295-1
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.