National Institute of Standards and Technology
U.S. Department of Commerce

Defending Against Software Supply Chain Attacks

Publication: April 2021

Cybersecurity and Infrastructure Security Agency

INTRODUCTION

A software supply chain attack occurs when a cyber threat actor infiltrates a software vendor's network and employs malicious code to compromise the software before the vendor sends it to their customers. The compromised software then compromises the customer's data or system. Newly acquired software may be compromised from the outset, or a compromise may occur through other means like a patch or hotfix. In these cases, the compromise still occurs prior to the patch or hotfix entering the customer's network. These types of attacks affect all users of the compromised software and can have widespread consequences for government, critical infrastructure, and private sector software customers.

This document provides an overview of software supply chain risks and recommendations on how software customers and vendors can use the National Institute of Standards and Technology (NIST) Cyber Supply Chain Risk Management (C-SCRM) framework and the Secure Software Development Framework (SSDF) to identify, assess, and mitigate risks.

SOFTWARE SUPPLY CHAIN RISKS

Software supply chains fit within the greater information and communications technology (ICT) supply chain framework. The ICT supply chain is the network of retailers, distributors, and suppliers that participate in the sale, delivery, and production of hardware, software, and managed services. The ICT Supply Chain Lifecycle has six phases. At each phase of the ICT Supply Chain Lifecycle, software is at risk of malicious or inadvertent introduction of vulnerabilities (see table 1 for examples).[1]

Table 1: ICT Supply Chain Lifecycle and Examples of Threats

DESIGN	**Hijacked Cellular Devices.** 2016 – A foreign company designed software used by a U.S. cell phone manufacturer. The phones made encrypted records of text and call histories, phone details, and contact information and transmitted that data to a foreign server every 72 hours.
DEVELOPMENT AND PRODUCTION	**SolarWinds.** 2020 – An IT management company was infiltrated by a foreign threat actor who maintained persistence in its network for months. The threat actor left the network only after it had compromised the company's build servers and used its update process to infiltrate customer networks.
DISTRIBUTION	**End-User Device Malware.** 2012 – Researchers from a major U.S. software company investigating counterfeit software found malware preinstalled on 20 percent of devices they tested. The malware was installed in new desktop and laptop computers after they were shipped from a factory to a distributor, transporter, or reseller.
ACQUISITION AND DEPLOYMENT	**Kaspersky Antivirus.** 2017 – An overseas-based antivirus vendor was being used by a foreign intelligence service for spying. U.S. government customers were directed to remove the vendor's products from networks and disallowed from acquiring future products from that vendor.
MAINTENANCE	**Backdoors Embedded in Routine Maintenance Updates.** 2020 – Thousands of public and private networks were infiltrated when a threat actor used a routine update to deliver a malicious backdoor.
DISPOSAL	**Sensitive Data Spillage.** 2019 – A researcher bought old computers, flash drives, phones and hard drives, and found only two properly wiped devices out of 85 examined. Also found were hundreds of instances of personally identifiable information (PII) spillage, including Social Security numbers, passport numbers, and credit card numbers.

[1] For additional details on the ICT Supply Chain Lifecycle, see the December 2018 CISA-NRMC factsheet, "Supply Chain Risks for Information and Communications Technology," https://www.cisa.gov/publication/supply-chain-risks-information-and-communication-technology.

Common Attack Techniques

Threat actors employ different techniques to execute software supply chain attacks. Three common techniques are:

- Hijacking updates
- Undermining code signing[2]
- Compromising open-source code

These techniques are not mutually exclusive, and threat actors often leverage them simultaneously.[3]

Hijacking Updates

Most modern software receives routine updates to address bugs and security issues. Software vendors typically distribute updates from centralized servers to customers as a routine part of product maintenance. Threat actors can hijack an update by infiltrating the vendor's network and either inserting malware into the outgoing update or altering the update to grant the threat actor control over the software's normal functionality. For example, the NotPetya attack occurred in 2017 when Russian hackers targeting Ukraine spread malware through tax accounting software popular in Ukraine. What would later be called the NotPetya malware spread well beyond Ukraine and caused major global disruptions in crucial industries, including international shipping, financial services, and healthcare.[4,5]

Undermining Codesigning

Codesigning is used to validate the identity of the code's author and the integrity of the code. Attackers undermine codesigning by self-signing certificates, breaking signing systems, or exploiting misconfigured account access controls. By undermining codesigning, threat actors are able to successfully hijack software updates by impersonating a trusted vendor and inserting malicious code into an update.[6] For example, APT 41, a China-based threat actor, routinely undermines codesigning while conducting sophisticated software supply chain compromises against the United States and other countries.[7,8]

---

[2] David Cooper, et al., "Security Considerations for Code Signing," NIST Cybersecurity White Paper (January 2018) https://csrc.nist.gov/publications/detail/white-paper/2018/01/26/security-considerations-for-code-signing/final.

[3] The Atlantic Council, "Breaking trust: Shades of crisis across an insecure software supply chain," https://www.atlanticcouncil.org/in-depth-research-reports/report/breaking-trust-shades-of-crisis-across-an-insecure-software-supply-chain/.

[4] CISA, Alert TA17-181A, "Petya Ransomware," https://us-cert.cisa.gov/ncas/alerts/TA17-181A.

[5] Wired, "The Untold Story of NotPetya, the Most Devastating Cyberattack in History," https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/.

[6] "Breaking trust," 14-16.

[7] U.S. Department of Justice, "Seven International Cyber Defendants, Including 'Apt41' Actors, Charged In Connection With Computer Intrusion Campaigns Against More Than 100 Victims Globally," https://www.justice.gov/opa/pr/seven-international-cyber-defendants-including-apt41-actors-charged-connection-computer.

[8] FireEye, "APT41: A Dual Espionage and Cyber Crime Operation," https://www.fireeye.com/blog/threat-research/2019/08/apt41-dual-espionage-and-cyber-crime-operation.html.

Compromising Open-Source Code

Open-source code compromises occur when threat actors insert malicious code into publicly accessible code libraries, which unsuspecting developers—looking for free blocks of code to perform specific functions—then add into their own third-party code. For example, in 2018, researchers discovered 12 malicious Python libraries uploaded on the official Python Package Index (PyPI). The attacker used typosquatting tactics by creating libraries titled "diango," "djago," "dajngo," etc., to lure developers seeking the popular "django" Python library. The malicious libraries contained the same code and functionality of those they impersonated; but they also contained additional functionality, including the ability to obtain boot persistence and open a reverse shell on remote workstations.[9] Open-source code compromises can also affect privately owned software because developers of proprietary code routinely leverage blocks of open-source code in their products.[10]

Software Supply Chain Attack Threat Profile

Software supply chain attacks typically require strong technical aptitude and long-term commitment, so they are often difficult to execute. These attacks differ from trusted relationship attacks in which threat actors infiltrate a less secure third-party organization to exploit and access an existing trusted connection that the third party has with the target organization.[11] Some criminal threat actors succeed in trusted relationship attacks and some of the less complex types of software supply chain attacks, such as modifying open-source code or app store attacks.

In general, advanced persistent threat (APT) actors are more likely to have both the intent and capability to conduct the types of highly technical and prolonged software supply chain attack campaigns that may harm national security.

Uniquely Vulnerable to Software Supply Chain Attacks

Organizations are uniquely vulnerable to software supply chain attacks for two major reasons: first, many third-party software products require privileged access; and second, many third-party software products require frequent communication between a vendor's network and the vendor's software product located on customer networks.

Privileged Access

Many common, third-party software products require elevated system privileges to operate effectively; this includes products like antivirus, IT management, and remote access software. Even when a product can effectively operate on a network with reduced privileges, products will oftentimes default to asking for greater privileges during installation to ensure the product's maximum effectiveness across different types of customer networks. Customers often accept third-party software defaults without investigating further, allowing additional accessibility vectors. Additionally, because these types of products are typically present on every system within a network, including authoritative and domain

---

[9] ZDNet, "Twelve malicious Python libraries found and removed from PyPI," https://www.zdnet.com/article/twelve-malicious-python-libraries-found-and-removed-from-pypi/.

[10] "Breaking trust," 20-21.

[11] MITRE, "Initial Access," https://attack.mitre.org/tactics/TA0001/.

management servers, vulnerabilities or malware inserted into those software products could provide malicious actors with privileged access to the most critical systems within a network.

Frequent Communication

Third-party software products typically require frequent communication with the vendor in order to update the software, fix known vulnerabilities, and provide security against new and evolving cybersecurity threats. This connectivity could allow malicious actors to send illegitimate software updates containing malware to the customer. Conversely, malicious actors could also intentionally prevent an update from reaching customers, ensuring those customers remain vulnerable to certain types of malware. Malicious actors can then exploit those vulnerabilities.

Consequences of Software Supply Chain Attacks

The consequences of a software supply chain attack can be severe. First, threat actors use the compromised software vendor to gain privileged and persistent access to a victim network. By compromising a software vendor, they bypass perimeter security measures like border routers, firewalls, etc., and gain initial access. If a threat actor loses network access, they may re-enter a network using the compromised software vendor. While gaining initial persistent access can be relatively indiscriminate, threat actors will often be more selective in choosing which victims they target for follow-on actions. Follow-on actions are highly variable but often start when the threat actor injects additional tailored malware packages into a chosen target. Depending on the threat actor's intent and capability, this additional malware may allow the threat actor to conduct various malicious activities that may include performing data or financial theft, monitoring organizations or individuals, disabling networks or systems, or even causing physical harm or death.

Software Supply Chain Compromise in the News: SolarWinds Orion

In December 2020, the cybersecurity firm FireEye discovered a backdoor – subsequently named SUNBURST - in the SolarWinds Orion platform. Researchers later discovered that a threat actor used an implant, referred to as SUNSPOT, to access the build server and insert the backdoor. After spreading the backdoor to many customers via routine updates, the threat actor targeted select victim networks for follow-on actions, including the use of additional malware. This software supply chain attack provided the threat actor access to systems and data on numerous government and private sector networks. The threat actor was patient, thorough, and maintained excellent operational security throughout the process, making their presence very hard to detect. Overall, the threat actor maintained a light malware footprint, using legitimate credentials and remote access when possible. While the SolarWinds Orion platform compromise provided access to most of the threat actor's victims, the threat actor used non-supply chain compromise techniques to gain access to a limited number of victims. The software supply chain attack conducted against SolarWinds and its customers serves as a recent example of how effective a software supply chain attack can be.

RECOMMENDATIONS

Network defenders are limited in their ability to quickly mitigate consequences after a threat actor has compromised a software supply chain. This is because organizations rarely control their entire software supply chain and lack authority to compel every organization in their supply chain to take prompt mitigation steps. Due to the difficulty of mitigating consequences *after* a software supply chain attack occurs, network defenders should observe industry best practices *before* an attack has occurred. Implementing best practices will bolster an organization's ability to prevent, mitigate, and respond to such attacks.

Recommendations for Customers

Organizations acquiring software should consider its use, as with other ICT products and services, in the context of a risk management program. Such a program should use an operationalized systems security engineering framework[12] and a formal C-SCRM approach across organization, mission/business, and system tiers.[13] A mature risk management program enables an organization to understand risks presented by ICT products and services, including software, in the context of the mission or business processes they support. Organizations can manage such risks through a variety of technical and non-technical activities, including those focused on C-SCRM for software and the associated full software lifecycle.

Risk Management Program
Some Simple Steps

1. Identify your key mission or business processes—what essential services do you provide or what drives your revenue?
2. Maintain an inventory of your organization's current and future software licenses
3. Research and document how each software license is supported by its supplier (e.g., Are patches provided? Does the supplier offer periodic email updates about the product?)
4. Understand how your software (current or future purchases) supports or otherwise relates to your key processes
5. Document how you would plan to address software for which a vulnerability is disclosed

NIST suggests eight key practices for establishing a C-SCRM approach that can be applied to software.[14]

1. Integrate C-SCRM across the organization.
2. Establish a formal C-SCRM program.
3. Know and manage critical components and suppliers.
4. Understand the organization's supply chain.
5. Closely collaborate with key suppliers.
6. Include key suppliers in resilience and improvement activities.
7. Assess and monitor throughout the supplier relationship.
8. Plan for the full lifecycle.

These practices can assist in preventing, mitigating, and responding to software vulnerabilities that may be introduced through the cyber supply chain and exploited by malicious actors.

Actions to Prevent Acquiring Malicious or Vulnerable Software

Establish a formal, organization-wide C-SCRM program to ensure that supply chain risk considerations receive attention across the organization. This includes executives and managers within operations and personnel across supporting roles, such as IT, acquisitions, legal, risk management, and security. Collectively, these roles can influence risk mitigation across an organization's suppliers through acquisition due diligence and contracting activities that:

- Apply the same policies to suppliers that are applied internally.

[12] Ron Ross, et al., "Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems," NIST SP 800-160 Vol. 1 (November 2016), https://doi.org/10.6028/NIST.SP.800-160v1.

[13] Jon Boyens, et al., "Supply Chain Risk Management Practices for Federal Information Systems and Organizations", NIST SP 800-161 (April 2015), http://dx.doi.org/10.6028/NIST.SP.800-161.

[14] Jon Boyens, et al., "Key Practices in Cyber Supply Chain Risk Management: Observations from Industry", NISTIR 8276 (February 2021), https://doi.org/10.6028/NIST.IR.8276.

- Establish a set of security requirements or controls for all suppliers varied based on the criticality of the supplier and the permissions granded to the ICT.
- Use supplier certifications to ascertain whether a supplier:
    - Uses a software development lifecycle (SDLC) and incorporates secure software development practices throughout all lifecycle phases.
    - Looks for known weaknesses and vulnerabilities in their source code and compiled code, and demonstrates the degree of rigor they apply. This may include requiring a specified level of developer testing and evaluation (e.g., static code analysis, threat modeling and vulnerability analysis, third-party verification of processes, manual code review, penetration testing, dynamic code analysis, etc.).[15]
    - Actively identifies and discloses vulnerabilities.
    - Maintains a product vulnerability response program.
    - Uses proactive exploit mitigation technologies in the code they acquire.
    - Enables patch management capabilities.
    - Submits products for third-party assessments.
    - Participates in Common Vulnerabilities and Exposures (CVE) generation, including whether the supplier participates as a CVE Numbering Authority (CNA).[16]
    - Develops, maintains, and uses approved supplier lists for its products and services.
- Require a software component inventory (e.g., software bill of materials) that articulates the components and other attributes of delivered software developed by the vendor and third parties.
- Ensure vendors enforce supply chain security requirements commensurate with those used by the organization acquiring the vendor's products and services.

Prevention
Some Simple Steps

1. Ask your software supplier/vendor (or check the vendor's website) whether the supplier:
    - Uses a software development lifecycle incorporating secure software development practices
    - Actively identifies and discloses vulnerabilities while maintaining a vulnerability response program
    - Enables patch management capabilities
    - Develops, maintains, and uses approved supplier lists for its products
2. Request a software component inventory with each contemplated software purchase
    - If a vendor cannot provide a component inventory, consider using that as a differentiator when selecting among competing products
    - Post-purchase, incorporate that information into your software inventory

As part of its standard acquisition and deployment process, an organization may be able to confirm software and firmware integrity by using common code authentication or other mechanisms.[17] In the absence of this opportunity, the organization should obtain a certification from the vendor that such authentication mechanisms were applied in the vendor's ordinary course of business while obtaining a digital signature or data for checksum verification. Customers can employ ongoing integrity

[15] NIST, "Security and Privacy Controls for Information Systems and Organizations," NIST SP 800-53 rev 5, SA-11, SR-6, https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final.

[16] For more information, see MITRE, "About CVE," https://cve.mitre.org/about/.

[17] NIST SP 800-161, SI-7.

management—either independently or in coordination with a manufacturer or distributor—by applying commercially available software/firmware tamper seals, which allow ongoing, automated integrity checking. An organization could make integrity management part of a broader comply-to-connect/comply-to-remain policy.[18]

Actions to Mitigate Deployed Malicious or Vulnerable Software

Despite C-SCRM actions, some malicious content and vulnerabilities may still find their way into an organization's enterprise environment. Therefore, an organization should take other steps to mitigate vulnerable software components.

Central to its efforts, an organization should develop and implement a vulnerability management program, which enables the organization to scan for, identify, triage, and mitigate discovered vulnerabilities. An organization's vulnerability management program should include processes and tools for provisioning and applying software patches, as necessary.[19]

An organization can reduce its software attack surface through configuration management, which includes:[20]

- Placing configurations under change control;
- Conducting security impact analyses;
- Implementing manufacturer-provided guidelines to harden software, operating systems, and firmware; and
- Maintaining an information system component inventory.[21]

Alongside configuration management, an organization should identify its critical data and baseline how that data flows between processes or systems. Defenders can deploy analytics, including those based on machine learning/artificial intelligence, to identify subsequent anomalies in data flows, which may be early indicators of a threat actor's exploitation of a vulnerability.

Every organization should monitor configuration settings to focus on maintaining the integrity of hardware, software, and firmware.[22]

For example, monitoring can identify unauthorized changes to Trusted Platform Module (TPM) configurations, such as those that establish which TPM features are enabled. Similarly, an organization

---

[18] NIST SP 800-53 rev 5, SI-7, SR-9, SR-10; National Security Agency, "Comply-to-Connect," https://apps.nsa.gov/iaarchive/library/ias/adversary-mitigations/comply-to-connect.cfm; Defense Information Systems Agency, "Comply to Connect Fact Sheet," https://www.disa.mil/-/media/Files/DISA/Fact-Sheets/Comply_to_Connect_Fact_Sheet_050720.ashx.

[19] Joint Task Force, "Security and Privacy Controls for Information Systems and Organizations", NIST SP 800-53 rev 5 (September 2020), RA-5, https://doi.org/10.6028/NIST.SP.800-53r5; Murugiah Souppaya and Karen Scarfone, "Guide to Enterprise Patch Management Technologies," NIST SP 800-40 rev 3 (July 2013), http://dx.doi.org/10.6028/NIST.SP.800-40r3; and Murugiah Souppaya, et al., "Improving Enterprise Patching for General IT Systems: Utilizing Existing Tools and Performing Processes in Better Ways," NIST SP1800-31a (Preliminary Draft, September 2020), https://www.nccoe.nist.gov/sites/default/files/library/sp1800/patching-nist-sp1800-31a-preliminary-draft.pdf.

[20] NIST SP 800-161, CM-1 through CM-11.

[21] NIST, "National Checklist Program Repository." https://checklists.nist.gov.

[22] NIST SP 800-53 rev 5, CM-2, CM-3, CM-6, CM-8, CM-14, SA-10, SI-2, SI-7, SR-9, SR-10.

should monitor configurations that establish a vendor- or user-defined hardened state and whether unauthorized changes to those configurations occur.

Additionally, limiting external and internal connections to only those on an approved list for each software deployment can help mitigate risk. Using expected software behavior—such as expected vendor URLs or IP ranges and ports with which a software package will periodically communicate—security engineering can implement information controls (e.g., firewalls, intrusion detection/prevention) to prevent and detect unexpected behaviors.[23] However, limiting connections based on static set of URLs or IP ranges may not be feasible or effective because many vendors have highly dynamic environments and use cloud service providers to host vendor resources. For many reasons, organizations should consider applying an identity- and object-based approach to baseline normal behavior. Organizations should also consider using machine learning or artificial intelligence to identify anomalies and deny abnormal information flows.

Mitigation
Some Simple Steps

1. Implement a documented vulnerability management program
   - Using instructions from the vendor, configure software to automatically check for and install patches
   - Register software licenses with the vendor, including contact information, so that vulnerabilities and mitigation strategies can be communicated
   - Follow vendor instructions to harden software, operating systems, and firmware
2. If vendor specifies URLs or IP ranges and ports to and from which software should communicate, consider establishing firewall rules to ensure such communications do not occur outside of those parameters
3. Where feasible, apply basic network segmentation to isolate different parts of the enterprise (e.g., maintain a separate network for guest users, separate the networks used by different functional areas of the organization, etc.)
4. Monitor endpoints and/or servers for unexplained deviations from your software inventory; remove or isolate unauthorized software

Using deliberate network segmentation, organizations can mitigate the effects of software vulnerabilities and associated exploits, as well as aid incident response and recovery. Segmentation helps confine a vulnerability or attack to portion of a customer's enterprise. Organizations can also achieve such mitigation by implementing endpoint-based micro-segmentation with host-based firewalls or agents. Micro-segmentation can be part of a "zero trust" architecture or implemented on its own.[24]

Furthermore, organizations can use heterogeneity techniques (e.g., using two or more vendors to cover different network segments) to increase resilience and decrease the overall enterprise risk from vulnerabilities of a single product or service (see following section).

Actions to Increase Resilience to a Successful Exploit

If a threat actor successfully exploits vulnerable software, organizations can use resilience measures to limit the impact to mission or business operations, personnel, and systems. An important action is to ensure the organization's contingency planning accounts for software.[25]

[23] NIST SP 800-161, AC-4, CA-3, and SC-7.

[24] Scott Rose, et al., "Zero Trust Architecture," NIST SP 800-207 (August 2020), https://doi.org/10.6028/NIST.SP.800-207.

[25] NIST SP 800-161, CP-1 and CP-2.

Where feasible, this planning includes pre-identifying and establishing alternative suppliers for software capabilities. It also includes establishing failover processes to follow when software capabilities and the processes they support become unavailable.

Establishing failover processes requires a strong understanding of how each piece of software is used within an organization—the mission or business it supports, the associated processes of which it is a part, and the anticipated mission or business impact if those processes are interrupted. Having this understanding enables the organization to assess the criticality of its various mission or business processes and associated software dependencies. Within that context, the organization can then identify failover options.

Understanding the software's criticality also enables an organization to make risk-based decisions regarding the extent to which resources should be spent on resilience measures. As an organization determines actions to increase resilience, it should consider developing a playbook for software supply chain compromises.[26]

Resilience
Some Simple Steps

1. Pre-identify and establish alternative suppliers for the critical software you use
    - Have plans in place to switch to a new supplier, if feasible, when critical software becomes unavailable or presents an increased risk
2. Use your understanding of how software supports critical business or mission functions to identify failover processes and workarounds in the event functionality with specific software becomes unavailable
    - Prepare written failover processes for critical software
    - Periodically conduct table-top exercises or walk-throughs to ensure your organization understands the steps in its failover processes
    - Where possible, coordinate failover processes with vendors and other external stakeholders

Recommendations for Software Vendors

CISA encourages software vendors to implement and follow a software development life cycle (SDLC) in their ordinary course of business. Vendor contracts for customers in some sectors increasingly contemplate—and even include—SDLC requirements. Vendors may also apply a maturity model to assess and communicate the quality and capability of their SDLC processes. However, NIST observes that "[f]ew [SDLC] models explicitly address software security in detail, so secure software development practices usually need to be added to each SDLC model."[27] As such, NIST published a white paper that suggests a subset of high-level practices that should be particularly helpful for integrating a secure software development framework (SSDF) into a vendor's SDLC. Much like the NIST Cybersecurity Framework, NIST's SSDF white paper offers a set of practices, subdivided into tasks, and mapped to industry and NIST standards. SDLC and SSDF processes provide a means for the vendor community to meet their customers' requirements for specific security practices.

Before a vendor can prevent, mitigate, or increase resilience related to its software, it must prepare for secure software development, which includes:[28]

---

[26] Michael Bartock, et al., "Guide for Cybersecurity Event Recovery", NIST SP 800-184 (December 2016), https://doi.org/10.6028/NIST.SP.800-184.

[27] Dodson, et al., "Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)," ii (April 23, 2020), https://doi.org/10.6028/NIST.CSWP.04232020.

[28] Ibid., 6.

- Defining software development security requirements,
- Establishing SSDF roles and responsibilities within the SDLC,
- Automating developer and security toolchains, and
- Establishing software security criteria and processes to collect the data necessary for security checks.

Actions to Prevent Supplying Malicious or Vulnerable Software

Vendors should implement an SSDF in the context of a secure development infrastructure. Vendors should build that infrastructure with a view towards securing the entirety of the SDLC and can follow a risk-based approach to select appropriate security controls for the anticipated development activities.[29] Beyond the SSDF, vendors may want to approach the software development environment in a manner similar to how the Federal Government approaches high-value assets (HVAs). If interested, vendors could:

1. Establish an organization-wide HVA governance program.
2. Identify and prioritize HVA information systems.
3. Consider the interconnectivity and dependencies of information systems when determining which systems are HVAs.
4. Develop a methodology for prioritizing HVAs based on criticality and mission importance.
5. Develop an assessment approach based on HVA prioritization.
6. Ensure timely remediation of identified vulnerabilities.[30]

Following these actions, CISA encourages vendors to follow a systems security engineering approach[31] to build security into their development infrastructure. Vendors should build security with an understanding of the interdependencies within the infrastructure and dependencies on connections with external systems.

Vendors can use the SSDF to prevent malicious software content or vulnerabilities from entering the cyber supply chain. NIST suggests practices to assist in protecting software and producing well-secured software. These include:

- Defining criteria for software security checks to help ensure that the software resulting from the SDLC meets the vendor's expectations when it checks the software's security during development.
- Supplying software that satisfies security requirements and mitigates security risks through design decisions.

- Protecting code from unauthorized access and tampering.
- Verifying that third-party software, such as libraries and other packages incorporated into vendor code, complies with security requirements.
- Reusing existing, well-secured software to reduce the risk of introducing vulnerabilities, when possible, instead of recreating functionality.
- Following secure coding practices to produce source code.

---

[29] Joint Task Force, "Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy," NIST SP 800-37 Rev. 2 (December 2018), https://doi.org/10.6028/NIST.SP.800-37r2.

[30] CISA, "CISA Insights: Secure High Value Assets (HVAs)," https://www.cisa.gov/sites/default/files/publications/CISAInsights-Cyber-SecureHighValueAssets_S508C.pdf.

[31] Ross, et al.

- Performing in-house and third-party code review, analysis, and testing.
- Using properly configured compilation and build processes to improve the security of executable code.
- Configuring software so that it is secure by default at the time of installation, such as:
    - Avoiding the use of hardcoded passwords.
    - Provisioning an operating system with its firewall enabled.
    - Enabling only minimally required services "out of the box."
- Providing a mechanism for verifying software release integrity (in particular, the protection of the code signing certificate) to help customers ensure that the software they acquire has not been subjected to tampering.[32]

For additional details on how to approach these practices, CISA recommends that vendors review the associated NIST publication, "Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)."

In addition, other NIST publications suggest that vendors consider implementing, where applicable:

- **Formal methods**—software development and analysis approaches based on mathematics and logic, including type checking, correctness proofs, model-based development, and correct-by-construction.
- **System-level security**—recent advances in hardware and software raise the possibility of security-enforcing and intrusion-tolerant systems that are both performance and cost effective.
- **Additive software analysis**—a comprehensive methodology for addressing impediments to using multiple advanced software checking tools in concert for synergy.
- **Domain-specific software development frameworks**—these promote the use (and reuse) of well-tested, well-analyzed code, and thus reduce the incidence of exploitable vulnerabilities.
- **Moving target defenses and automatic software diversity**—a collection of techniques to automatically vary software's detailed structures and properties such that an attacker has much greater difficulty finding and exploiting any weakness.[33] For example, a vendor might implement exploit mitigation technology to limit the impact of unidentified vulnerabilities on in-house, bespoke code, open source, and software acquired from third-party vendors. The nature of exploit mitigation should align to the inherent nature of risk in the code.

Actions to Mitigate Post-Deployment Malicious or Vulnerable Content

Although it is important to prevent the presence of malicious or vulnerable content within software, not all vulnerabilities can be eliminated. However, vendors should make post-deployment mitigations available.

Vendors should follow a set of practices to support such mitigations. These include:

- Archiving and protecting each release of software so that the vendor can

[32] Dodson., 10–18.

[33] Paul E. Black, et al., "Dramatically Reducing Software Vulnerabilities: Report to the White House Office of Science and Technology Policy," NISTIR 8151 (November 2016), https://doi.org/10.6028/NIST.IR.8151.

analyze, identify, and develop mechanisms to eliminate vulnerabilities discovered post-release.
- Maintaining processes, and even a formal program, to identify and confirm suspected vulnerabilities in software, whether identified by the vendor, its customers, or third-party researchers.
- Establishing an assessment, prioritization, and remediation approach that enables vulnerabilities to be remediated quickly.[34]

Vendors should develop software to enable future patching to eliminate undesirable content. Additionally, in support of its software customers, a vendor should develop and deliver a software component inventory (e.g., software bill of materials) to customers with each release of software.[35] Primary beneficiaries may include product developers, who incorporate third-party software into their deliverables, as well as sophisticated customers. Coupled with tools that can check the software component inventory against known vulnerabilities, including a software component inventory can support customers before they deploy new or updated software within their environments.

Finally, beyond the vulnerability identification and remediation practices previously highlighted, a vendor should implement a disclosure practice for discovered vulnerabilities. This practice should include identifying and making vulnerability mitigations available to customers as quickly as possible (and ideally prior to or simultaneous to a disclosure), submitting vulnerabilities into the CVE community and, if appropriate, becoming a CVE Numbering Authority.[36]

Actions to Increase Resilience in the Software Development Process

A vendor can use lessons learned and feedback loops to increase resilience.[37] Specifically, analyzing discovered vulnerabilities and identifying their root causes allows a vendor to pinpoint opportunities for improvement in its SDLC, including its SSDF. While this will not prevent or mitigate vulnerabilities from previously distributed software, it implements a continuous improvement process, which enables the vendor to produce increasingly secure code.

Considerations to Implementing a Secure Software Development Framework

The software supply chain is both exploitable by malicious actors and susceptible to the unintended introduction of vulnerabilities. However, software customers and vendors can manage these risks through both independent and collaborative efforts. Although they can use technical security controls to prevent or mitigate software supply chain risk, customers and vendors should recognize that robust planning and communication are fundamental to risk management in this area. Vendors should incorporate security features in their software design plans. Customers can assist by communicating security requirements to their vendors.

---

[34] Ibid., 10, 19–-20.

[35] National Telecommunications and Information Administration Multistakeholder Process on Software Component Transparency Framing Working Group, “Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM),” (November 12, 2019), https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf.

[36] For more information, see About CVE.

[37] Dodson, et al., 20–21.

Similarly, customers should make risk-informed decisions regarding software procurement and deployment. Customers can build such considerations into their acquisition processes, and they can select software based on vendor SDLC practices. Customers should request—and vendors should disclose—those practices. In turn, vendors should make available technical details regarding their software. If a customer understands the expected behaviors of software—such as its external communication periodicity, the ports and protocols it uses for external communications, and the IP range or domains with which it is expected to communicate—the customer can implement controls to allow only such connections and monitor for deviations.

Even a vendor's well-implemented SDLC and a customer's astute procurement due diligence and contracting provisions will not eliminate all vulnerabilities from entering the software supply chain. Vendors and customers will need to mitigate vulnerabilities as they become known. Mitigation efforts may include patching; as such, vendors should maintain a focus on vulnerability identification and responsible disclosure in already-distributed software. Patch development and distribution should receive similar attention, and vendors should document patching processes so that customers understand how to participate.

A software bill of materials will also assist customers as they address previously unknown vulnerabilities—or guide acquisition decisions if a software bill of materials suggests software contains components known to be vulnerable. Often, a customer needs to understand whether a vulnerable component exists in its enterprise environment.

RESOURCES

For further information, consider the following references contained within NIST's documentation on SSDF and C-SRCM.

SSDF

- NIST: Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)
- BSIMM: Building Security in Maturity Model (BSIMM) Version 11
- BSA: The BSA Framework for Secure Software: A New Approach to Securing the Software Lifecycle, Version 1.1
- Institute for Defense Analyses (IDA): State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation
- International Organization for Standardization/International Electrotechnical Commission (ISO/IEC): Information technology – Security techniques – Application security – Part 1: Overview and concepts, ISO/IEC 27034-1:2011
- Microsoft: Microsoft Security Development Lifecycle
- NIST: Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1
- NIST: SP 800-53 Rev. 5, Security and Privacy Controls for Information Systems and Organizations
- NIST: SP 800-160 Vol. 1, Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems

- Open Web Application Security Project (OWASP): OWASP Application Security Verification Standard 4.0.2
- OWASP: Software Assurance Maturity Model Version 1.5
- Payment Card Industry (PCI) Security Standards Council: Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures Version 1.1
- Software Assurance Forum for Excellence in Code (SAFECode): Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program, Third Edition
- SAFECode: Managing Security Risks Inherent in the Use of Third-Party Components
- SAFECode: Practical Security Stories and Security Tasks for Agile Development Environments
- SAFECode: Software Integrity Controls: An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain
- SAFECode: Tactical Threat Modeling

C-SCRM

- NIST SP 800-161: Supply Chain Risk Management Practices for Federal Information Systems and Organizations 2015
- NIST: Cybersecurity Framework
- NIST: Risk Management Framework
- NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations
- NISTIR 8272: Impact Analysis Tool for Interdependent Cyber Supply Chain Risks
- NISTIR 8151: Dramatically Reducing Software Vulnerabilities
- NISTIR 8179: Criticality Analysis Process Model: Helping Organizations
- NISTIR 8276: Key Practices in Cyber Supply Chain Risk Management: Observations from Industry
- NIST: Federal C-SCRM Forum
- NIST, DoD, DHS, GSA: Software and Supply Chain Assurance (SSCA) Forum
- NCCoE Demonstration Project