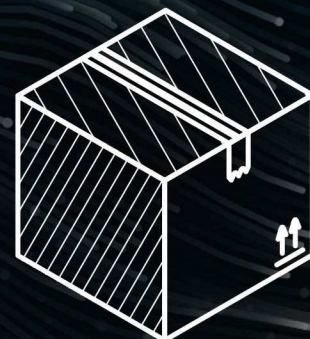# SECURING THE SOFTWARE SUPPLY CHAIN

RECOMMENDED PRACTICES GUIDE FOR
SUPPLIERS

Enduring Security Framework

September 2022

Executive Summary

Cyberattacks are conducted via cyberspace and targets an enterprise's use of cyberspace for the purpose of disrupting, disabling, destroying, or maliciously controlling a computing environment or infrastructure; or destroying the integrity of the data or stealing controlled information.[1]

Recent cyberattacks such as those executed against SolarWinds and its customers, and exploits that take advantage of vulnerabilities such as the Log4j, highlight weaknesses within software supply chains, an issue which spans both commercial and open source software and impacts both private and Government enterprises. Accordingly, there is an increased need for software supply chain security awareness and cognizance regarding the potential for software supply chains to be weaponized by nation state adversaries using similar tactics, techniques, and procedures (TTPs).

In response, the White House released an Executive Order on Improving the Nation's Cybersecurity (EO 14028). EO 14028 establishes new requirements to secure the federal government's software supply chain. These requirements involve systematic reviews, process improvements, and security standards for both software suppliers and developers, in addition to customers who acquire software for the Federal Government.

Similarly, the Enduring Security Framework[2] (ESF) Software Supply Chain Working Panel has established this guidance to serve as a compendium of suggested practices for developers, suppliers, and customer stakeholders to help ensure a more secure software supply chain. This guidance is organized into a three part series: Part 1 of the series focuses on software developers; Part 2 focuses on software suppliers; and Part 3 focuses on software customers.

Customers (acquiring organizations) may use this guidance as a basis of describing, assessing, and measuring security practices relative to the software lifecycle. Additionally, suggested practices listed herein may be applied across the acquisition, deployment, and operational phases of a software supply chain.

The software supplier (vendor) is responsible for liaising between the customer and software developer. Accordingly, vendor responsibilities include ensuring the integrity and security of software via contractual agreements, software releases and updates, notifications, and mitigations of vulnerabilities. This guidance contains recommended best practices and standards to aid suppliers in these tasks.

This document will provide guidance in line with industry best practices and principles which software developers are strongly encouraged to reference. These principles include security requirements planning, designing software architecture from a security perspective, adding security features and maintaining the security of software and the underlying infrastructure (e.g., environments, source code review, testing).

---

[1] Committee on National Security Systems (CNSS)

[2] The ESF is a cross-sector working group that operates under the auspices of Critical Infrastructure Partnership Advisory Council (CIPAC) to address threats and risks to the security and stability of U.S. national security systems. It is comprised of experts from the U.S. government as well as representatives from the Information Technology, Communications, and the Defense Industrial Base sectors. The ESF is charged with bringing together representatives from private and public sectors to work on intelligence-driven, shared cybersecurity challenges.

DISCLAIMER

DISCLAIMER OF ENDORSEMENT

This document was written for general informational purposes only. It is intended to apply to a variety of factual circumstances and industry stakeholder, and the information provided herein is advisory in nature. The guidance in this document is provided "as is." Once published, the information within may not constitute the most up-to-date guidance or technical information. Accordingly, the document does not, and is not intended to, constitute compliance or legal advice. Readers should confer with their respective advisors and subject matter experts to obtain advice based on their individual circumstances. In no event shall the United States Government be liable for any damages arising in any way out of the use of or reliance on this guidance.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government, and this guidance shall not be used for advertising or product endorsement purposes. All trademarks are the property of their respective owners.

PURPOSE

NSA, ODNI, and CISA developed this document in furtherance of their respective cybersecurity missions, including their responsibilities to develop and issue cybersecurity recommendations and mitigations. This information may be shared broadly to reach all appropriate stakeholders.

CONTACT

Client Requirements / Inquiries: Enduring Security Framework nsaesf@cyber.nsa.gov,

Media Inquiries / Press Desk

- NSA Media Relations, 443-634-0721, MediaRelations@nsa.gov
- CISA Media Relations, 703-235-2010, CISAMedia@cisa.dhs.gov
- ODNI Media Relations, dni-media@dni.gov

Table of Contents

1   Introduction

Unmitigated vulnerabilities in the software supply chain pose a significant risk to organizations. This series presents actionable recommendations for a software supply chain's development, production and distribution, and management processes to increase the resiliency of these processes against compromise.

All organizations have a responsibility to establish software supply chain security practices to mitigate risks, but the organization's role in the software supply chain lifecycle determines the shape and scope of this responsibility.

Because the considerations for securing the software supply chain vary based on the role an organization plays in the software supply chain, this series presents recommendations geared toward these important roles, namely, developers, suppliers, and customers (or the organization acquiring a software product).

This guidance is organized into a three part series and will be released coinciding with the software supply chain lifecycle. This is Part 2 of the series which focuses on the software supplier. Part 1 of the series focused on software developers and Part 3 of the series will focus on the software customer. This series will help foster communication between these three different roles and among cybersecurity professionals that may facilitate increased resiliency and security in the software supply chain process.

In this series, terms such as risk, threat, exploit, and vulnerability are based on descriptions defined in the Committee on National Security Systems Glossary (CNSSI 4009).[3]

1.1   Background

Historically, software supply chain compromises largely targeted commonly known vulnerabilities organizations that were left unpatched. While threat actors still use this tactic to compromise unpatched systems, a new, less conspicuous method of compromise also threatens software supply chains and undermines trust in the patching systems themselves that are critical to guarding against legacy attacks. Rather than waiting for public vulnerability disclosures, threat actors proactively inject malicious code into products that are then legitimately distributed downstream through the global software supply chain. Over the last few years, these next-gen software supply chain compromises have significantly increased for both open source and commercial software products.

Technology consumers generally manage software downloads and broader, more traditional software supply chain activities separately. Considering both the upstream and downstream phases of software as a component of supply chain risk management may help to identify problems and provide a better way forward in terms of integrating activities to achieve systemic security. However, there are also some differences to account for in the case of software products. A traditional software supply chain cycle is from point of origin to point of consumption and generally enables a customer to return a malfunctioning product and confine any impact. In contrast, if a

---

[3] CNSSI-4009.pdf

software package is injected with malicious code which proliferates to multiple consumers; the scale may be more difficult to confine and may cause an exponentially greater impact.

Common methods of compromise used against software supply chains include exploitation of software design flaws, incorporation of vulnerable third-party components into a software product, infiltration of the supplier's network with malicious code prior to the final software product being delivered, and injection of malicious software that is then deployed by the customer.

Stakeholders must seek to mitigate security concerns specific to their area of responsibility. However, other concerns may require a mitigation approach that dictates a dependency on another stakeholder or a shared responsibility by multiple stakeholders. Dependencies that are inadequately communicated or addressed may lead to vulnerabilities and the potential for compromise.

Areas where these types of vulnerabilities may exist include:

- Undocumented features or high risk functionality,
- Unknown and/or revisions to contractual, functionality or security assumptions between evaluation and deployment,
- Supplier's change of ownership and/or of geo-location, and
- Poor supplier enterprise or development hygiene.

1.2   Document overview

This document contains the following additional sections and appendices:

Section 2 provides best practices and standards recommended for suppliers to help ensure the integrity and security of software from production through delivery.

Section 3 is a collection of appendices supplementing the preceding sections:

Appendix A: Crosswalk Between the NIST SP800-218; *Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework* (SSDF) [4] and Use Cases described herein.

Appendix B: Dependencies

Appendix C: Supply-Chain Levels for Software Artifacts (SLSA)[5]

Appendix D: Recommended Artifacts and Checklist

Appendix E: Informative References

Appendix F: Acronyms.

---

[4] NIST SP 800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities
[5] GitHub - slsa-framework/slsa: Supply-chain Levels for Software Artifacts

Each section contains examples of threat scenarios and recommended mitigations. Threat scenarios explain how processes that compose a given phase of the software development lifecycle (SDLC) relate to common vulnerabilities that could be exploited. The recommended mitigations present controls and mitigations that could reduce the impact of the threats.

2   Supplier

A supplier acts as a liaison, or intermediary, between the developer and customer, and, as such, retains primary responsibilities over the following:

1. Maintaining the integrity of securely delivered software.
2. Validating software packages and updates.
3. Maintaining awareness of known vulnerabilities.
4. Accept Customer reports of issues or newly discovered vulnerabilities and notifying Developers for remediation.

The objective of a secure software development and delivery system is to help safeguard software code, provenance, and integrity, thereby creating resilience to compromise of the software supply chain or preventing it entirely.

Threat scenarios described in this section align with NIST SP800-218; *Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework* (SSDF)[6]. The intent of the scenarios in this section is to provide high-level guidance on the communication mechanisms that should exist between developers, suppliers, and customers. While this section focuses on software delivered via a traditional, on-premises method, the section discusses Software-as-a-Service (SaaS) delivery, where applicable.

2.1   Prepare the Organization

2.1.1   Define Criteria for Software Security Checks

Policies should be established and implemented that specify checks required to securely deliver software. These policies should be accessible and known by everyone who has a role in the Software Development Lifecycle (SDLC) and should include notifying customers of vulnerabilities, mechanisms to be used for mitigations, and End-of-Life (EOL) support.

Threat scenarios: On-Premises

The following are example scenarios that could create exploitable vulnerabilities:

1. Policies do not exist or are not clearly defined.
2. The team is unaware of or does not implement policy requirements throughout the SDLC.
3. Policy requirements are not inclusive (e.g., no defined mechanism to notify customers of vulnerabilities) or do not provide customer support through the product's end-of-life.

Recommended mitigations

The following mitigations can help reduce threats and risks associated with the deployment process:

1. Verify that the software shipped was the same as that received.
2. Create fully signed images and binary code.

---

[6] NIST SP 800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities

3. Ensure a secure hash to validate the binary.
4. Ensure the communication channel is secure.
5. Establish and implement secure processes for updates/patches.
6. Verify revision level of executable:
   a) Version control should be implemented,
   b) Changes should be timestamped,
   c) A Version Control System (VCS) should be implemented to allow reverting if needed.
7. Verify organizational checks are conducted by leveraging internationally recognized standards, such as NIST SSDF, which helps ensure software functionality and security requirements have been met prior to software release and throughout software lifecycle.

2.2   Protect software

2.2.1   Protect all Forms of Code from Unauthorized Access

All forms of code should be protected from unauthorized access and tampering, and the principle of least privilege should be applied throughout the SDLC. This is key in ensuring that code delivered to customers includes all required security features and that those features are working as designed. This includes code with no additional, possibly hidden, functionality that would reduce security such as backdoors or hard-coded passwords.

Threat scenarios: On-premises

The following situations present threats to the integrity of software code:

- Persons with unauthorized access and the ability to clearly view the unencrypted, un-obfuscated design or source code,
- Unauthorized modification or deletion of code or packages used in the build and delivery processes,
- Unauthorized modification or deletion of documentation, code, or packages prior to delivery to customer,
- Existence of backdoors or hard-coded passwords to enable future unauthorized access.

Recommended mitigations

The following mitigations can help reduce threats and risks associated with the process:

- Enforce role-based access control (RBAC) with segmentation of duties and least privilege,
- Identify and vet trusted architects, developers, testers, and documentation personnel with the requisite skills necessary to complete the project,
- Assign individuals to one or more design, coding, and quality assurance (QA) tasks based on their capabilities and interests and register the privileges for their corresponding project roles in secure access control systems,
- Verify each assigned individual's corporate-issued development system conforms to all company security standards, including full-disk encryption, minimum password complexity, patch management; antivirus and intrusion detection systems, AI-based endpoint protection platforms and endpoint detection and response, data loss prevention (DLP), etc.,

- Ensure the code repository, build, and test environments have at least the same security protections as other critical network capabilities such as network segmentation, firewalling, monitoring, automated encryption, and remote backups,

- Ensure only corporate-issued or approved development systems can access the development, build, and test environments using multi-factor authentication (MFA) or continuous authentication based on behavior analytics[7] and only through office networks with physical security or through secure virtual private networks (VPNs). Ensure that failed access attempts are detected, reported, and investigated. This is particularly important for mobile or remotely working employees,

- Conduct reviews of third-party software (e.g., using binary software composition analysis) and assure the security of those included modules,

- Deliver digitally signed code and associated supporting files using a code-signing system that protects sensitive signing keys and that uses hardware protection such as a Federal Information Processing Standards (FIPS) 140-2/-3 Hardware Security Module (HSM). This requires at least two individuals to activate the signing keys and approve the software release package (i.e., code, supporting files, and metadata),

- Establish a strong security culture in the development and operations support teams,

- Review personnel, tasks, systems, and policies to ensure they continue to be appropriate, necessary, and complete. Conduct reviews both on a schedule and as triggered by events.

Threat scenarios: SaaS

The following are examples of cloud-native software development scenarios that could be exploited such as:

- Software development process that promises faster time to market, better scalability and management, and lower costs, all while maintaining the same levels of development security and integrity,

- Adoption of a new approach which requires changes to on-premises development and distribution processes as well as security and security management regimes,

- Key changes that include adoption of containerization and micro services architectures.

Recommended mitigations

The following mitigations can help reduce threats and risks associated with the development process:

- Use of strong authentication, authorization, code scans, vulnerability analyses, and digital signing of applications,

- These practices should be broadened, as needed, to address any additional authentication challenges associated with endpoint and operational cloud security.

---

[7] Zero Trust Architecture (nist.gov)

2.2.2   Mechanism for Verifying Integrity of Software Releases

Suppliers should provide a mechanism for verifying software release integrity by digitally signing the code throughout the software lifecycle. Digitally signed code enables recipients to positively verify and trust the provenance and integrity of the code.

Threat scenarios: On-premises

A lack of verifiable provenance and integrity would make it easier for the following scenarios to occur:

- An adversary with access to a software support activity could substitute malicious software for a legitimate component,
- An adversary with access to a software support activity could insert malicious code into a legitimate component,
- Insertion of malicious code could occur on either the supplier's side prior to delivery, or on the recipient's side prior to use.

Recommended mitigations

The following mitigations could help reduce threats and risks associated with threats:

1. Digitally sign code so recipients can verify and trust the provenance and integrity of the code:
   a) Because code signing only provides protection benefits after code is signed, and to mitigate risk of code tampering and malicious code injection prior to signing, it is recommended that organizations consider implementing a verifiable build process to serve as a checkpoint prior to signing code,
   b) The verifiable build system should consist of a logically isolated secondary mirror of the production build environment that independently compiles and packages code concurrently with production build environment compilation and packaging routines,
   c) The hashed results of two parallel operations can be compared, and, if verified as a match, the production build artifacts can be signed,
   d) Final-revision binary code should be signed using a code-signing certificate issued by a trusted certificate authority and a mechanism provided to validate those signatures.
2. Establish and implement procedures for key management that includes:
   a) Private keys used for code-signing operations should be stored securely on an HSM to mitigate the risk of theft of code-signing keys,
   b) Systems that access key materials and processes stored on an HSM or code-signing infrastructure should be protected with appropriate infrastructure security controls,
   c) Code-signing operations should be protected with appropriate infrastructure security controls to mitigate the risk of code-signing key misuse and abuse.
3. Include signature validation in the list of security checks and ensure it is appropriately executed:

a) Document the signature validation mechanism or process in the official security configuration guide or equivalent documentation,

b) If a third-party supplier signature validation mechanism is not available for architectures that allow third-party suppliers, a custom programmatic method should be developed and provided,

c) Suppliers of third-party software should establish checks to validate the authenticity and integrity of software packages when they are received and prior to incorporation into any internal build process,

d) The signature validation mechanism should include certificate revocation list checks and trust chain verification,

e) In the signature, include a timestamp issued by a timestamping authority.

2.2.3   Archive and Protect each Software Release

Organizations should have an archival strategy that allows the organization to specify major and minor releases. Some organizations are bound by specific standards, such as the Payment Card Industry Data Security Standard, Health Insurance Portability and Accountability Act (HIPAA), and Sarbanes-Oxley Act. These standards impose frequency and retention requirements on the supplier. However, each software supplier determines when software should be archived, how long it is kept before it can be overwritten or destroyed, and where it should be stored. Archived software releases can be used for disaster recovery, but are also useful for forensic investigations during cyber-incidents as well.

Threat scenarios: On-premises

Missing or inadequately established enterprise-wide policies that provide guidance for determining which files and metadata should be stored together increases the likelihood of the following threat scenarios:

- Using inappropriate types of storage mediums that are insecure, not easily accessible or available,
- Infrequent review of archives or reviews that are not in accordance with the organization's retention policy.

Recommended mitigations

By moving released software to an archived state, which is typically a lower-cost storage area, the organization can save money and allocate faster storage for more critical software projects. This process can also speed up productivity by reducing the time it takes for employees to open files under development and to access the associated metadata. The following mitigations can help reduce associated risks and threats:

1. Establish repositories with appropriate access control.
2. Ensure code and compiled executables are maintained in a persistent archive.
3. Archives should have limited access and store software in read-only mode so that it cannot be altered. Creating archives in read-only mode serves to retain its integrity should it be needed in an emergency, investigation, or data breach. This approach also assures customers that the software is available in the event of a compromise.

4. The media used to store archives is determined by the organization, and the decision usually hinges on its convenience, reliability, and availability. Organizations have traditionally used network storage and other media such as tape devices:

   a) Tape media is standard for some organizations that need a low-cost way to store large amounts of data in a small space. However, retrieval and restore for this media can become a problem,

   b) Attached network drives are also common, but this media is much more expensive. Network storage requires the real estate to host it and expensive hardware to secure and maintain it. However, unlike most tape systems, network drives offer archive data that is readily available should the organization or investigators need to access it,

   c) Cloud storage has the advantages of availability and low costs, but the speed is dependent on the organization's bandwidth and network speed. Many organizations have moved to cloud storage for its convenience and savings. However, it is still the responsibility of the organization to keep the data secure,

   d) Based on the organization, other storage types are listed below:

      • Block storage services, which expose software-defined block devices that can be presented to virtual hosts running in the cloud,

      • Object storage services, which can be mapped to hosts, applications, or even other cloud services, and allow addressing discrete, unstructured data elements by ID or metadata,

      • Scalable shared file systems, which allows a scalable set of hosts to access the same file system at a high speed,

5. The process of archiving data is often automated using software. The features and capabilities offered by archiving software depend on the supplier, but most have standard features across every platform:

   a) A system administrator configures the time, location, and frequency for software to be archived. An archiving policy is created to determine the rules behind moving data. Using archive policies, an administrator ensures that data moved to the storage location follows the proper regulatory standards and requirements,

   b) In conjunction with other rules about archiving, a retention policy is also necessary. A retention policy determines the amount of time an archive stays available before the data can be overwritten or destroyed. A typical retention policy for backups is about 30 days, but archived data might be retained longer before it is destroyed. Some organizations keep archived data for years before media is rotated or archives are deleted. For the most sensitive data, archives may never be overwritten or destroyed. Archiving and compliance standards may have a retention policy requirement, so organizations should ensure that their configuration does not violate any regulatory standards.

2.3   Produce Well-Secured Software

2.3.1   Design Software to meet Security Requirements

Producing well-secured software and mitigating security risks are key objectives in delivering software that permits a customer to access only information and resources for which they are authorized and prevents access to unauthorized information and resources. To achieve this, security requirements must be accurate and complete, the code must satisfy these requirements, and any exploitable vulnerabilities should be addressed.

Threat scenarios: On premises

The following are example scenarios that could be exploited:

- Developers do not employ secure development practices throughout the SDLC,
- Developers or associated security team are inadequately trained in conducting threat or risk assessments,
- Design or operational requirements are not fully understood or implemented.

Recommended mitigations

The following mitigations can help reduce threats and risks associated with the process:

- Demonstrate secure development lifecycle practices and processes, to include design specifications that define the intended environment,
- Perform threat modeling to achieve security objectives,
- Obtain as much information as possible about the software sources and operational environment,
- Use skilled software architects and security professionals to conduct a threat and risk assessment for the purpose of understanding how different deployment scenarios, incidents, and operational errors may impact the software's ability to satisfy its requirements,
- Document design and operational assumptions, so the impact of requirements and other changes on security may be more easily assessed. Revisit design assumptions and decisions throughout the development cycle.

Threat scenarios: SaaS

Software developers utilize continuous delivery or continuous deployment methodologies, which integrate security, for their SaaS offerings. The following are example scenarios that could be exploited:

- An adversary, or unwittingly employee, may introduce vulnerable code into a repository that triggers an automated build and deployment of compiled code,
- Customers of a SaaS offering may become susceptible to exploitation of vulnerable code already residing on their systems,
- Variations in security requirements between different modules within a single SaaS solution may result in insufficient validation of all components,

- A threat actor or malicious insider may compromise a module provided by a third-party supplier that is not validated by the Supplier or third-party supplier.

Recommended mitigations: SaaS

The following are example mitigations of threats:

- Establish, enforce, and verify common security requirements between SaaS suppliers and all associated sub-contractors and third-party suppliers,
- Define common tools for performing security evaluations, of software products, which must be used by all parties responsible for code development,
- Execute any code not fully vetted from a security perspective within a minimally privileged execution environment until the code can be fully evaluated.

2.3.2   Verify Third-Party Supplier Software Complies with Security Requirements

Third-party supplied software is often available in organizations - inside the organization or included in other products. Suppliers must ensure both natively developed software and any components obtained by third-party suppliers complies with security requirements.

Threat scenarios: On-Premises

The following are example scenarios that could be exploited:

- Contractual agreements between a supplier, third-party suppliers, and a customer may "cascade;" third-party software is a potential source of additional vulnerabilities,
- The supplier excludes common factors (e.g., geolocation, supplier ownership/control and past performance) when making a risk decision to use a software component supplied by a third party,
- Unsuccessfully expressing security requirements in terms of software engineering quality (e.g., cyclomatic complexity), minimum cryptographic key length, identifying approved cryptographic algorithms, identifying approved libraries, or compiler options.

Recommended mitigations

The following are example mitigations to threats:

1. Verify whether third-party software complies with security requirements; this reduces the risks associated with using acquired software modules and services.
2. Assess whether the third-party supplied software meets the applicable security requirements:
   a) Define security requirements (e.g., minimum key length, use of FIPS 140-2 crypto graphic algorithms),
   b) If applicable, communicate the security requirements to the third-party supplier via contractual agreement. The contractual agreement should also include vulnerability disclosure/mitigation requirements, or incident reporting,
   c) Define tests that indicate compliance with security requirements (e.g., source code analyses, analyses of source code deltas, or binary code analyses),
   d) Define a process to re-evaluate the security requirements and tests,

   e) Identify the third-party software impacted by modified or additional security requirements and tests,

   f) Define a process mitigating third-party software which satisfies applicable security requirements.

3. Ensure contractual agreement addresses potential third-party software concerns:

   a) Contractual agreements should be in place, if possible. The agreement should detail security requirements and require such things as timely disclosure of vulnerabilities and SDLC practices,

   b) Only make copies of third-party software available if it meets security requirements [NIST SSDF PO.2], and the specific security requirements met should be indicated. Copies should only be obtained from a read-only location maintained by the organization.

4. Vet third-party software as a potential source of additional vulnerabilities:

   a) Maintain a read-only location with approved third-party software which identifies the security requirements conformed to [NIST SSDF PS.1],

   b) Deny usage of third-party software that was not obtained from the read-only location,

   c) Only permit usage of approved third-party software when its security requirements conform to the developed software's security requirements:

   - Disapprove third-party software if (new) tests indicate (new) security requirements are violated,

   - Document the actions taken when it is determined that third-party software does not meet (new) security requirements.

   d) Establish, and update as necessary, a corporate policy that enforces usage of only the most suitable, or most recent, version of third-party software:

   - Example use cases in a corporate policy may outline requirements for updating the third-party software when a new version is available, where the new version patches a critical vulnerability, every X number of minor releases of the third-party software, and every major release of the third-party software.

2.3.3   Configure the Compilation and Build Processes

Whether the build process is 'full' or 'incremental,' dictates whether it is performed using the 'never seen' or 'last build state.' Full builds check for dependencies, compile all the source files, build all the parts in order, and assemble it into a build artifact.

In an instance of an incremental build, the build checks and compares the source and other files that depend on the target for modifications since the last build. If a modification exists, the target will rebuild. Otherwise, the file from the last build will be reused.

Threat scenarios: On-Premises and SaaS

The following are example scenarios that could be exploited:

- An adversary with access to software processes and tools within the production build environment may insert malicious software into components,
- Inadequately configuring the compilation and build process may compromise the executable's security,
- Improperly safeguarding the integrity of the production build process and the binary artifacts which it generates.

Recommended mitigations

The processes that are used to compile and build software components must be properly configured to be secure by default in order to insure the integrity of all binary production code artifacts. The following controls should be implemented to harden software compilation and build processes:

1. Organizations should establish and maintain a trusted toolchain for all tools involved in the compilation and building of software; these tools should be configured to a known secure baseline state, recorded, and tracked via an inventory maintained in a Configuration Management Database, continuously monitored for emergent security vulnerabilities, and receive appropriate security updates in accordance with defined remediation timelines.

2. All build processes should be automated and the resulting scripts and metadata should be stored securely in a version control system with access limited to the individuals responsible for building the components.

3. Non-interactive service accounts should be used to invoke automated build processes to ensure build outputs are service-generated and non-falsifiable.

4. The authentication credentials of service accounts which execute automated compilation and packaging processes should be verified using an approved method to validate the processes as trusted.

5. Automated build processes should execute in an ephemeral environment which is logically isolated and free from external influence.

6. Automated build processes should generate and maintain a build manifest identifying builder, sources, entry point, and parameters to ensure build reproducibility.

7. Secure compiler settings should be enabled to help prevent or limit the effectiveness of some types of security issues, most notably buffer overflows (both stack and heap-based). Examples of secure compiler settings may include, but are not limited to, the following:

   a) Enable stripping of symbols from binary output,

   b) Enable data execution prevention,

   c) Enable safe structured exception handling,

   d) Runtime checks for security,

   e) Enable address space layout randomization,

   f) Emit an error if an array index can be determined at compile time to be out of bounds,

   g) Emit a warning upon the detection of a suspicious use of an address pointer.

8. Organizations should consider implementing a verifiable build process consisting of a logically isolated secondary mirror of the production build environment which independently compiles and builds software components concurrently with the production build environment. The hashed results of these two parallel operations can then be compared to verify the integrity of the build process.

2.3.4 Review or Analyze Human-Readable Code

Software should include operational and security functionality, and both should be considered when making an information security assessment. This is especially important for the procurement, management, and deployment processes. Organizations utilizing software packages with strong security features may reduce their information security risk exposure and reduce the likelihood of further issues.

The underlying software quality and security of software technology should be a consideration for calculating total cost of ownership, risk mitigation and control, operating costs, and calculating overall business value. It is important to help ensure the confidence level of a software package's security, functionality, and value. Quality measures should include constraints for human-readable code to be reviewed to help identify vulnerabilities and verify compliance with all requirements.

Threat scenarios: On-Premises

The following are example scenarios that could be exploited:

- Inability for code to be easily reviewed, or analyzed, to help identify vulnerabilities and verify compliance,
- Security functionality that is inadequate or inappropriate to meet procurement, management, or deployment requirements.

Recommended mitigations

The following are example mitigations to threats:

1. Establish a security assurance program that requires the following:
   a) A security risk evaluation has been performedm
   b) Security requirements have been established for the software and data being developed and/or maintainedm
   c) Security requirements have been established for the development and maintenance processm
   d) Each software review and audit includes evaluation of security requirementsm
   e) The configuration management and corrective action processes provide security for the existing software and the change evaluation processes prevent security violationsm
   f) Physical security for the software and the data is adequatem
   g) Security assurance includes activities for the requirements, design, implementation, testing, release, and maintenance phases of an SDLC and CI/CDm
   h) Code Coverage which measures unit test coverage.

2. Review and/or analyze human-readable code to ensure it is in accordance with an established security assurance program, identify vulnerabilities and verify compliance with established requirements:

    a) Product definition and marketing requirements documents,

    b) Product charter, vision, and scope documents,

    c) Requirements and user stories,

    d) Design documents,

    e) Product specifications,

    f) Verification plans,

    g) Test and validation plans and test cases.

2.3.5　Test Executable Code

Executable code should be tested to identify vulnerabilities and verify compliance with security requirements. A supplier must seek to deliver software that permits customer access to information and resources for which they are authorized, while preventing access to information for which they are not authorized. To achieve this, suppliers and developers should work in concert to help ensure all known vulnerabilities are mitigated.

Threat scenarios: On-Premises

The following are example scenarios that could be exploited:

- Foregoing the implementation or use of the privilege concept,
- Lack of, or inadequate testing and scanning of vulnerabilities,
- Misunderstanding of, lack of, or inappropriate implementation of security requirements.

Recommended mitigations

The following are example mitigations to threats:

1. Obtain a clear and complete set of security requirements. Have the requirements reviewed and verified by a team of security subject matter experts to ensure requirements are explicit, appropriate, and comprehensive.
2. Understand the governance and compliance requirements for sales made to the US Government (e.g., FedRAMP, OMB policies, accessibility, etc.).
3. Threat models should be developed for all critical software components, as well as for all systems in the build pipeline (e.g., all systems in which source code resides, or which can be accessed, repositories, build systems, etc.):

    a) Each threat model should be reviewed and approved by at least two independent engineers on the team,

    b) All code and systems should be reviewed on an ongoing basis against associated threat model(s) and changes should be made as needed to ensure that code and systems do not have structural vulnerabilities,

    c) Threat models should be updated for major releases or at least annually,

      d)   Threat models should be made available to other internal engineering teams, such as those picking up or operating any associated software components or systems.

4. Develop test plans to cover each requirement. Code coverage should be a key element of every component's test plan and should be integrated into each build and tracked as part of implementing the test plan. Baseline levels of code coverage should ideally be achieved on all code that is checked in, and before new code is committed.

5. Staff the security test team based on employee experience and qualifications.

6. Provide sufficient testing resources (e.g., hardware and software) and time to execute the test plans.

7. Security testing should be part of every software component's QA plan and should include the following (see NIST SSDF):

      a)   Static (linting) and dynamic source code analysis should be performed on all code prior to each release using a standard set of company-approved tools. Results of testing should be documented, and all discovered vulnerabilities should be analyzed and mitigated,

      b)   Fuzzing should be performed on all software components to assure that they exhibit expected behavior on all inputs. Results should be documented, and any anomalies or vulnerabilities should be mitigated,

      c)   Penetration testing should be done every 6-24 months depending on potential risk (e.g., cloud products should be pen tested more frequently). All results should be documented, and issues should be mitigated,

      d)   Results of all security testing should be documented, including any Common Vulnerability Scoring System (CVSS) scores and security defects should be mitigated and verified.

8. Repeat steps 1-7, if necessary, after changes are made.

9. Verify and document that security requirements have been satisfied.

2.3.6   Configure the Software to have Secure Settings by Default

Customers are sometimes anxious to quickly install software and may occasionally use it in an operational environment without completing proper configurations or before configuration options are fully understood. To prevent these actions from leading to compromise, software should require minimal access upon installation, allowing customers to explicitly enable additional accesses when required.

Threat scenarios

The following are example scenarios that could be exploited:

- Software being installed and used prior to complete or proper configuration,
- Software installations that allow fully operational access by default,
- Software that does not limit access to and/or execution of administrative processes,
- Neither administrative nor general user actions are logged.

Recommended mitigations

The following are example mitigations to threats:

1. Require all administrative/configuration changes to occur over local or strongly-encrypted remote connections.
2. Locally log all administrative actions (including attempted and successful authentications and configuration changes) by default and keep the log for at least 30 days.
3. Provide only one default administrative account and do not provide any default user accounts or access.
4. Require customers to change the default administrative authentication credential immediately after first login; recommend instituting MFA if possible.
5. Require all administrative account credentials to be sufficiently complex (passwords) or of sufficient cryptographic strength (certificates). Access attempts by any account should involve MFA if possible, through office networks with physical security and/or secure VPNs, and/or continuous authentication based on behavior analytics (see NIST SP 800-207).
6. Disable services and functionalities that do not require authentication until explicitly enabled.
7. Password recovery procedures must require console or physical access. Password reset procedures should revert to the initial state. Refer to NIST Guidelines for more information on password characteristics (i.e., length, complexity, time, etc.).

2.4   Respond to Vulnerabilities

2.4.1   Identify, Analyze, and Remediate Vulnerabilities on a Continuous Basis

Suppliers should make every effort to ensure that publicly known or easily identified vulnerabilities are not in any software provided to customers. Testing for, understanding, and removing vulnerabilities in software is required to help prevent supplying code that can be readily compromised before delivering software to customers.

Moreover, customers who procure software have a strong interest in obtaining transparent information regarding the provenance and security of the software being purchased. The controls below are designed to allow transparency of an organization's secure software development process. Failure to perform these steps and maintain evidence of adherence may diminish confidence that software is of sufficient quality to operate in customers' networks.

Threat scenarios: On-Premises

The following are example scenarios that could be exploited:

•   Software contains known and/or non-disclosed vulnerabilities,
•   Testing or attempts to remove known vulnerabilities are incomplete or inadequate,
•   Provenance of software or components is unknown.

Recommended mitigations: On-Premises

The following are example mitigations to threats:

1. Create a vulnerability assessment team consisting of architects, developers, testers, cryptologists, and human factor engineers whose goal is to identify exploitable weaknesses in software.

2. For the software capability define a process that uses known environment analysis, monitor vulnerabilities associated with the software capability, and unknown environment fuzz testing of individual units within the combined system.

3. For the software components define a process that uses known environment analysis, uses source or binary composition analysis tools to monitor vulnerabilities associated with the identified software components, and unknown environment fuzz testing of individual units within the combined system.

4. Invest in static and dynamic evaluation tools that are state of the art. Keep them current and implement them according to supplier documentation.

5. Create a central company-wide Product Security Incident Response (PSIRT) team. Public-facing PSIRT information (e.g., on a web page) should be easily accessible for external researchers to report vulnerabilities in the organization's products. The PSIRT team should work with external researchers to acknowledge and gather information on any reported vulnerabilities, as well as to ensure that any reported vulnerability is fixed. Organizations should practice responsible disclosure on all vulnerabilities.

6. All known security issues and/or vulnerabilities should be tracked as product defects in the organization's defect tracking tool. Items tracked should include CVSS scores, specific impacts on the component, and any other relevant supporting data. Vulnerability information should only be stored in access-controlled pages in a bug tracking system and based on the potential sensitivity.

7. Provide sufficient human and compute resources, software testing, and time to test based on the multiple factors and complexity that could constitute a software component or package. Factors may include load, branches, race conditions, corner cases, etc..

8. Review and either eliminate or document any weaknesses found.

9. Refer to the SBOM (or a similar mechanism) related to third-party software and open-source components associated with the software. Establish and follow corporate guidance on the upgrade of embedded components as issues are announced.

10. When a software component is modified, repeat the recommended process herein for that unit and the system.

Threat scenarios: SaaS

The following are example scenarios that could be exploited:

- The deployment and implementation of SaaS applications at the expense of security,

- Organizations which have been provided with the capability to enhance, improve, and optimize their overall workflow,

- Speedy and fast adoption and acquisition of SaaS tools and products (especially to satisfy rapid post-COVID work from home requirements) may have inherent risk(s) and may ultimately impact the overall security posture of an organization.

Recommended mitigations: SaaS

The following are example mitigations to threats:

1. Implement a stringent security policy towards SaaS application security.
2. Design a mechanism to monitor and scan third-party applications which are directly connected to the cloud environment.
3. Develop a comprehensive and reliable backup solution.
4. Implement identity and access control mechanisms.
5. Develop mature security assessments (this may possibly include utilizing Cloud Access Security Brokerage capabilities) so that any security gaps between the cloud service customer and cloud service provider may be bridged.
6. Implement industry standard encryption algorithms.

3   Appendices

3.1   Appendix A: Crosswalk Between Scenarios and SSDF

The section reference numbers in the below crosswalk may look similar for each role (Developer, Supplier and Customer) however they are from the respective parts of the Series. (PO – Prepare Organization; PW - Produce Well-Secured Software; PS – Protect Software; and RV – Respond to Vulnerabilities)

SSDF #	Developer	Supplier	Customer
PO.1	2.2.3 Secure Development Practices	2.1.1 Define criteria for software security checks	
PS.1	2.2.1.1 Source Control Check-in Process 2.2.1.4 Code Reviews 2.2.6 External Development Extensions 2.3.2 Selections and Integration 24.1 Build Chain Exploits 2.5.3 Secure the Distribution System	2.2.1 Protect all forms of code from unauthorized access 2.2.2 Provide a mechanism for verifying software release integrity (PS.1, PW.9)	
PS.3	2.2.1.1 Source Control Check-in Process 2.2.1.2 Automatic and Manual Dynamic and Static Security / Vulnerability Scanning 2.3.2 Selections and Integration 2.3.3 Obtain Components from a Known and Trusted Supplier 2.4.1 Build Chain Exploits	2.2.3 Archive and protect each software release	
PW.1	2.3.2 Selections and Integration	2.3.1 Design software to meet security requirements	
PW.3	2.2.3 Secure Development Practices 2.3.2 Selections and Integration	2.3.2 Verify third-party software complies with security requirements	2.1 Procurement/Acquisition (1) Requirements Definition / Recommended Controls (viii)(viii)

	2.3.3 Obtain Components from a Known and Trusted Supplier 2.3.4 Component Maintenance 2.3.5 Software Bill of Material (SBOM)		2.2 Deployment (6) (2) Testing – Functionality (c) Recommended Controls (ii) Verify contents in SBOM 2.2 Deployment (6) Deploy (3) Contracting / Recommended Controls (v) (viii) (ix)(x)
PW.6	2.2.3.2 Use of Unsecure Development Build Configurations 2.4.1 Build Chain Exploits	2.3.3 Configure the compilation and build processes	
PW.7	2.2.1.4 Code Reviews 2.2 Open source Management Practices 2.2.6 External Development Extensions 23.2 Selections and Integration 2.3.3 Obtain Components from a Known and Trusted Supplier	2.3.4 Review and/or analyze human-readable code	
PW.8	2.2.1.3 Nightly Builds with Regression Test Automation 2.3.2 Selections and Integration 2.4.1 Build Chain Exploits	2.3.5 Test executable code	
PW.9	2.2.3.2 Use of Unsecure Development Build Configurations 2.4.1 Build Chain Exploits	2.2.2 Provide a mechanism for verifying software release integrity (PS.1, PW.9) 2.3.6 Configure the software to have secure settings by default	
RV.1	2.3.4 Component Maintenance 2.4.1 Build Chain Exploits	2.4.1 Identify, analyze, and remediate vulnerabilities on a continuous basis	

3.2 Appendix B: Dependencies

3.2.1 Developer Group Dependencies

Green - Dependencies/artifacts recommended to be provided by the supplier for benefit of the developer.

Dark Green - Dependencies/artifacts recommended to be provided by the third-Party suppliers for benefit of the developer.

Pink - Dependencies/artifacts recommended to be provided by the customer for benefit of the supplier/developer.

#	Dependency
1	Provide issues from customers
2	Provide given hashes as required
3	SDLC policies and procedures
4	Secure architecture, high-level design
5	Qualified team assembly with code/security training
6	Independent QA individual/team
7	Independent security audit individual/team
8	Open-Source Review Board (OSRB) with repository
9	Product release management/resources
10	SBOM
11	Development location and information
12	Third-party SBOM
13	Third-party License
14	Release notes (detailing vulnerabilities fixed)
15	Vulnerability notifications
16	Publish updates and patches to the customer to address new vulnerabilities or weaknesses found within the product
17	Requirements and criteria for success
18	Implied industry security requirements
19	Provide issues from operational environment, take updates and patches
20	Vulnerability notifications and reporting from the users

3.3   Appendix C: Supply-Chain Levels for Software Artifacts (SLSA)

Supply-Chain Levels for Software Artifacts (SLSA) is a security framework from source to service, giving anyone working with software a common language for increasing levels of software security. The framework is currently in Alpha stage and constantly being improved by supplier-neutral community. Google has been using an internal version of SLSA since 2013 and requires it for all their production workloads. http://slsa.dev

Requirement	Description	L1	L2	L3	L4
Scripted build	All build steps were fully defined in some sort of "build script." The only manual command, if any, was to invoke the build script. Examples: • Build script is Makefile, invoked via make all. • Build script is. github / workflows / build.yaml, invoked by GitHub Actions.	✓	✓	✓	✓
Build service	All build steps ran using a build service, not on a developer's workstation. Examples: GitHub Actions, Google Cloud Build, Travis CI.		✓	✓	✓
Ephemeral environment	The build service ensured that the build steps ran in an ephemeral environment, such as a container or virtual machine (VM), provisioned solely for this build, and not reused from a prior build.			✓	✓
Isolated	The build service ensured that the build steps ran in an isolated environment free of influence from other build instances, whether prior or concurrent. • It MUST NOT be possible for a build to access any secrets of the build service, such as the provenance signing key. • It MUST NOT be possible for two builds that overlap in time to influence one another. • It MUST NOT be possible for one build to persist or influence the build environment of a subsequent build. • Build caches, if used, MUST be purely content-addressable to prevent tampering.			✓	✓
Parameterless	The build output cannot be affected by user parameters other than the build entry point and the top-level source location. In other words, the build is fully defined through the build script and nothing else. Examples: • GitHub Actions workflow dispatch inputs MUST be empty.				✓

	• Google Cloud Build user-defined substitutions MUST be empty. (Default substitutions, whose values are defined by the server, are acceptable.)				

Hermetic	All transitive build steps, sources, and dependencies were fully declared up front with immutable references, and the build steps ran with no network access.
	
	The developer-defined build script:
	• MUST declare all dependencies, including sources and other build steps, using immutable references in a format that the build service understands.
	
	The build service:
	• MUST fetch all artifacts in a trusted control plane.
	• MUST NOT allow mutable references.
	• MUST verify the integrity of each artifact.
	   o If the immutable reference includes a cryptographic hash, the service MUST verify the hash and reject the fetch if the verification fails.
	   o Otherwise, the service MUST fetch the artifact over a channel that ensures transport integrity, such as TLS or code signing.
	• MUST prevent network access while running the build steps.
	   o This requirement is "best effort." It SHOULD deter a reasonable team from having a non-hermetic build, but it need not stop a determined adversary. For example, using a container to prevent network access is sufficient.
					✓

Reproducible	Re-running the build steps with identical input artifacts results in bit-for-bit identical output. Builds that cannot meet this MUST provide a justification why the build cannot be made reproducible. "○" means that this requirement is "best effort." The developer-provided build script SHOULD declare whether the build is intended to be reproducible or a justification why not. The build service MAY blindly propagate this intent without verifying reproducibility. A customer MAY reject the build if it does not reproduce.				○

3.4   Appendix D: Artifacts and Checklist

In principle, any artifacts created during the lifecycle of the software development process are owned by and private to a developing organization. These organizations can determine what artifacts are made available with potential and current users of a product with or without a Non-Disclosure Agreement (NDA). Availability of information must take into consideration regulatory and legal requirements, the customer requirements for the information and the risk involved by exposing information leading to the exploitation of the product. Exceptions may include open-source development organizations, which are more inclined to make all development information available, to include source code.

When defining the availability of an artifact, the general terms used in this section will be the following:

1. Publicly disclosed

2. Externally available
    a) under a Non-Disclosure Agreement (NDA)
    b) government agency mandated requirement

3. Private / company confidential

The availability of an artifact varies between companies and agencies and is only described here as a reference for what might be possible when using artifacts to validate the software supply chain process. Some artifacts, such as a high-level architecture document may be intentionally generated to allow any perspective consumers an introductory artifact detailing the overall strategies used in the design, development, and operation of a product. These publicly disclosed documents may describe common industry nomenclature, such as Federal Information Process Standards (FIPS) compliance, cryptography standards used, development processes adhered to or certifications processes passed. NDA and government mandated availability require contractual agreements providing access to artifacts that would not normally be exposed by the organization that produced the product. While private/company confidential artifacts are generally low-level and detailed work products that may contain sensitive secrets and knowhow and if exposed, provide potential insight into product's competitive implementation and threat vectors that may not be addressed in the product, therefor posing a threat if exposed outside of the producers environment. Private/company confidential artifacts are generally maintained by the "Suppliers" and "Developers" of the product to facilitate the auditing and validation of adherence to the Secure Software Development Lifecycle (Secure SDLC) and Security practices set forth by the product owner, company, or organization. For more information on the Secure SDLC process, refer to Section **2.1 "Secure Product Criteria and Management,"** subsection "Recommended Mitigations," Item 8 of the Part 1 Developer of the series.

Most of the artifacts collected during the development lifecycle are not meant to be shared outside the developing organization yet may be preserved in persistent storage as evidence to verify the integrity of the policies and processes used during the development of a product. A developer should securely retain artifacts of software development for a certain duration according to its secure software development policies and processes. As a by-product of the process used to implement and mitigate the attack surface and threat model of the software as well as the software build pipeline during the development process, the following artifacts may be created, and collected:

Artifact Examples	Description/Purpose
High-level Secure Development Lifecycle Process document	Attestation to secure development practices which can cover: • Secure software architecture/design process • Attack surface investigation and threat modeling process • Secure software development/programming training • Software security testing process • Source control check-in process • Trusted repository for modules and processes • Continuous integration and delivery (CI/CD) processes • Defect/vulnerability reporting and customer update process • Code reviews process for security and continuous software security improvement • Continuous verification of third-party binaries • Open-source management practices • Hardening the build environment • Secure relationship with a third-party supplier • Process to secure the signing server • Final package validation process
Product Readiness checklist	Attestation to product release and secure shipping criteria and product readiness for shipment which can cover: • No pending known critical bugs and vulnerabilities (e.g. bug track report) • Cryptographically signed components • Proper software licensing
Product Support/Response Plan	Attestation to vulnerability discloser and response process (e.g. handling of policy violation and anomalies)
Software Bill of Material (SBOM)	• Attestation to the integrity of the producer • Attestation to the security and authenticity of components included in the product • Attestation to the third-party software components • Attestation to the integrity of software licenses
Architecture/Design Documents	• Attestation to secure architecture/design practices • Mitigation of attack surface vulnerabilities • Attestation to mapping between secure requirements to software architecture and components
Developer Training Certificates/Training completion Statistics/data	• Attestation to secure development practices • Attestation to secure coding practices
Threat Model Results Document	• Attestation to secure design practices • Attestation to secure third-party component integration practices

High-level Software Security Test Plan and Results	High-level, system and unit level test plan and results (A set of tests should be commensurate with the requirements and risk profile of the product or service.) - Coverage details - SAST - Static Application Security Testing - DAST - Dynamic Application Security Testing - SCA - Software Composition Analysis - Fuzzing/Dynamic - Penetration - Red team testing - Black box testing - QA security feature analysis
Automatic and Manual Dynamic and Static Security/Vulnerability (Security Scanning Results) Reports	The reports can cover: - Security Scanning Results for Static, Dynamic, Software Composition Analysis and Fuzzing - Security Scanning Results for Penetration or Red-Teaming - Attestation to secure development/build/test practices - Mitigation against known software weakness classes in the Common Weakness Enumeration (CWE) - Mitigation against publicly known vulnerabilities and Common Vulnerabilities and Exposures (CVEs)
Open-Source Review Process Document and Allowed List	Attestation to secure open-source review process and management
Build Log	- Attestation to the integrity of securely built products - Attestation to no known critical errors/warnings - Attestation to use of tool-chain defenses (stack checking, ASLR, etc.)
Secure Development Build Configurations Listing	Attestation to secure build environment
Third-Party Software Tool-Chains List	Attestation to secure build environment

The artifacts described in the table above may be used for attestation of the integrity of an organizations' secure development process that was used to produce a given product. Organization can then provide a high-level checklist, illustrated below, which may utilize artifacts created during the development process that attest to the adherence, at some level, of the recommended practice during the development process. The developer may add a brief description regarding how the organization supports a check list item in addition to Yes/No/Not Applicable (NA)/Incomplete (Inc) response, e.g. alternative practices to support it and reasons for non-applicability.

The document references in the following table are focused on the Supplier Section of the Guidance release.

Measurable Outcome/ Description	Practice Observed Yes, No, NA, Inc	SSDF Tasks	Artifact Examples	Document References
Secure Product Criteria & Management				
Do you define policies that specify risk-based software architecture and design requirements?		**PO.1.2**	Architecture/Design Documents	
Do you require team members to regularly participate in secure software architecture, design, development, and testing training and monitor their training completion?		**PO.2.2** **RV.3.4**	Training Completion Data/Statistics Developer Training Certificates	
Have development team members attended training programs specific to their roles, development tools and languages to update their skills?		**PO.2.2**	Training Completion Data/Statistics Developer Training Certificates	
At a minimum, for all critical software components and external services that your team operates and own, have you completed the attack surface survey and threat models for all such services?		**PW.1.1** **PW.2.1**	Threat Model Results Documents	2.3.5 Test Executable Code
Do you have up to date threat models for all critical components your team ships that have been reviewed by a person trained in software security and do you make this document available to other teams that pick up your component?		**PW.1.1, PW.2.1**	Threat Model Results Document	2.3.5 Test Executable Code
Has your team held a black-box investigation for security?			Black box test results	
Do you have and use security tools and methodology (e.g. recommended by NISTIR 8397) for static, dynamic and Software Composition Analysis		**PO.3.1**	SAST, DAST, SCA test results	2.3.5 Test Executable Code

and ensure that all high severity issues are addressed?				
Do you perform input fuzzing as part of a regular process for your component or product's inputs?		**PW.8.2**	Fuzzing/Dynamic test results	2.3.5 Test Executable Code
Do you have security testing as part of your overall QA plan to enhance the testing of specific features of your product?			Product test results	2.3.5 Test Executable Code
Have your product or components been identified as needing penetration testing? If so, are all issues found recorded in a bug tracker, with high priority defects set to prevent shipment of the product?		**PW.8.2**	Penetration Test Results	2.3.5 Test Executable Code
Have your product or components been identified as needing red-team testing? If so, are all issues found recorded in a bug tracker, with high priority defects set to prevent shipment of the product?			Red-Team Test Results	
Have your product or components been identified as needing testing for security gaps by an external party? If so, has your code or systems been tested for security gaps by an external party (e.g. JFAC Software Assurance providers, pen testing, threat model reviews, vulnerability scan tools and red-teams)?			Third-party Test Results	2.3.5 Test Executable Code
Does your release include an SBOM and confirmation that no unacceptable security vulnerabilities are pending, binaries are digitally signed and meet cryptographic standards?			SBOM Product Bug Tracking Report	

Are all public cloud resources continuously monitored by a tool that analyzes and alerts for policy violations and anomalies?			Product Support / Response Plan	2.4.1 Identify, analyze, and remediate vulnerabilities on a continuous basis
Are the alerts being actively monitored?			Product Support / Response Plan	
Is there a process in place to resolve policy violations within a specific amount of time?			Product Support / Response Plan	2.4.1 Identify, Analyze, and Remediate Vulnerabilities on a Continuous Basis
Develop Secure Code				
Are all your security issues tracked with a bug tracker and scored, for example using CVSSv3 scores to help determine fix prioritization and release scheduling?		**RV.2.1**	Secure Software Development Lifecycle Process document Bug Tracker Report	2.2.3 Archive and Protect Each Software Release 2.3.5 Test Executable Code
Do you use access-controlled applications to store sensitive vulnerability information for all issues affecting production code that is more restrictive than plain bug tracker defects?		**PO.5.1**	Secure Software Development Lifecycle Process document	2.2 Protect Software 2.4.1 Identify, Analyze, and Remediate Vulnerabilities on a Continuous Basis
Does your team have a process to reduce a class of vulnerabilities based on previously identified vulnerabilities or attacks?		**PW.7.2**	Secure Software Development Lifecycle Process document	2.3.4 Review and/or Analyze Human-Readable Code
Do you perform nightly builds with automated regression and security test to quickly detect problems with recent builds?			Secure Software Development Lifecycle Process document	
Are code check-ins gated by code collaborators and source control to prevent anyone from accidentally or intentionally submitting un-reviewed code changes?		**PW.7.2**	Secure Software Development Lifecycle Process document	
Does the team require code reviews for all code and build scripts / configuration changes?		**PW.7**	Secure Software Development Lifecycle Process document	

Does the team measure and analyze the quality of the code review process?			Secure Software Development Lifecycle Process document	
Do you ensure only required modules are included in the product and "unused" modules and code out of scope of the requirements and design document are uninstalled or removed, mitigating "living-off-the-land" attacks and decreasing the attack surface?			Secure Software Development Lifecycle Process document Requirements Document	
Do you map all your security requirements to the software component of the product and track their completion/adherence?			Secure Software Development Lifecycle Process document Security Requirements Document	
Are unmodified third-party libraries retrieved from a common location such as a secured persistent storage or shared repository location out of band of the development process and not individually built by your team?			Secure Software Development Lifecycle Process document	
Do you monitor new vulnerabilities applicable to your software e.g. using registered vulnerability notification services?		RV.1.1	Secure Software Development Lifecycle Process document	
Do you have and adhere to responsible disclosure requirements for all externally identified vulnerabilities?			Secure Software Development Lifecycle Process document	
Are all your builds continuously built and tested?			Secure Software Development Lifecycle Process document	2.3.3 Configure the Compilation and Build Process
Does a check-in immediately trigger a build?			Secure Software Development Lifecycle Process document	2.3.3 Configure the Compilation and Build Process

Does a completed build automatically go through some acceptance testing?			Secure Software Development Lifecycle Process document	
If the testing passes, is the build automatically deployed so others can consume it?		**PO.3.1**	Secure Software Development Lifecycle Process document	
Verify Third-Party Components				
Do you track all third-party components you use directly and all internal components in a secure and persistent repository?		**PS.1.1** **PW.4.1**	Secure Software Development Lifecycle Process document OSRB Approved List Product/Component Scan Results	
Do you have the requirement for an Open-Source Review Board to approve third-party libraries included in a product and audit approved third-party libraries for version adherence and vulnerabilities?		**PW.4.1** **PW.4.4**	Secure Software Development Lifecycle Process document OSRB Approved List	
Do you remove or mitigate critical known vulnerabilities or end of life issues of third-party components before each release?		**PW.4.5**	Secure Software Development Lifecycle Process document OSRB Approved List	2.2.2 Provide a Mechanism for Verifying Software Release Integrity 2.3.2 Verify Third-Party Software Complies with Security Requirements
When considering the selection of a third-party component, do use a known and trusted supplier that has a proven record for secure coding practices and quality delivery of their components?		**PO.1.3**	Secure Software Development Lifecycle Process document OSRB Approved List	
Within a developer environment, do you monitor and approve of all IDEs and third-party development/debugging extension to ensure their adoption does not weaken the			Secure Software Development Lifecycle Process document	

security posture of the local development environment?				
Do you have a trusted repository to support ongoing software composition analysis and security testing for all external and downloaded modules?			Secure Software Development Lifecycle Process document	
Harden the Build Environment				
Have you completed attack surface investigation and threat modeling for your build environment?			Threat/Risks Model Results Documents	
Do you ensure that only in very rare cases, the build process accesses the open Internet and these cases are documented and approved within the security plan?		**PO.5.1**	Secure Software Development Lifecycle Process document	
Do you limit and secure access to your development environment to essential administrators?			Secure Software Development Lifecycle Process document	
Do you monitor the build chain for unauthorized access and modifications?			Secure Software Development Lifecycle Process document	
Do you document approval and audit logs of build chain modifications?			Secure Software Development Lifecycle Process document	
Do you enforce build-chain configuration defensive techniques required to narrow the attack vectors of the components and products being developed?			Secure Software Development Lifecycle Process document Build Logs	
Do you ensure the integrity of the individual development environment, caring to harden the development systems within the build pipeline?			Secure Software Development Lifecycle Process document	

Does your build process encrypt data in transit?			Secure Software Development Lifecycle Process document	2.2.1 Protect all Forms of Code from Unauthorized Access 2.3.6 Configure the Software to have Secure Settings by Default
Does each critical server within the build chain owned by the team have a clearly defined owner responsible for patch maintenance?		**PO.5.1**	Secure Software Development Lifecycle Process document	2.2.1 Protect all Forms of Code from Unauthorized Access
Do you have a requirement that server patch levels are checked periodically?			Secure Software Development Lifecycle Process document	
Is unnecessary outbound internet connectivity blocked?		**PO.5.1**	Secure Software Development Lifecycle Process document	
Is unnecessary inbound internet connectivity blocked?		**PO.5.1**	Secure Software Development Lifecycle Process document	
Is the integrity of the builds verified to ensure no malicious changes have occurred during the build and packaging process, for example, are two or more builds performed in different protected environments and the results compared to ensure the integrity of the build process?			Secure Software Development Lifecycle Process document	
Do you use the toolchain to automatically gather information that informs security decision-making?		**P0.4.2**	Secure Software Development Lifecycle Process document	2.3.3 Configure the Compilation and Build Process
Does the tool chain automatically scan for vulnerabilities and stop the build process and report errors when detected, if so configured?		**PS.1.1** **PW.7.2**	Secure Software Development Lifecycle Process document	2.4.1 Identify, Analyze, and Remediate Vulnerabilities on a Continuous Basis
Do you store access credentials (e.g. hashes for				

passwords) and secrets in a secure (e.g. encrypted) location such as a secure vault?				
Secure Code Delivery				
Do you perform binary composition analysis of the final package?			Secure Software Development Lifecycle Process document	
Do you have a Software Bill of Materials (SBOM) that satisfies the contracts?		**PS.3.2** **PW.4.1**		
Do you digitally sign all required binaries you ship?		**PS.1.1** **PS.2.1**	Secure Software Development Lifecycle Process document	
Do you ensure that globally-trusted certificates are not directly accessible and use a dedicated, protected signing server when signing is required?			Secure Software Development Lifecycle Process document	2.2.2 Provide a Mechanism for Verifying Software Release Integrity
Are you using organization approved Configuration Management tools to sign your shipping binaries?			Secure Software Development Lifecycle Process document	2.2.2 Provide a Mechanism for Verifying Software Release Integrity
Do you comply with the use of cryptography recommended by organization's security policy?		**PS.1.1**	Secure Software Development Lifecycle Process document	

3.5 Appendix E: Informative References

Abbreviation	Document Name
ACM	Communications of the ACM 17, "The Protection of Information in Computer Systems". Available at (http://web.mit.edu/Saltzer/www/publications/protection/index.html)
BSA	BSA (2019) Framework for Secure Software. Available at (https://www.bsa.org/reports/bsa-framework-for-secure-software)
BSIMM10	Migues S, Steven J, Ware M (2019) Building Security in Maturity Model (BSIMM) Version 10. Available at (https://www.bsimm.com/download/)
CISA	Cybersecurity & Infrastructure Security Agency. Available at (https://www.cisa.gov/defining-insider-threats)
CISCO_SDLC	Cisco. 2021. Cisco Secure Development Lifecycle. Available at (https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/cisco-secure-development-lifecycle.pdf)
EO14028	EOP. 2021. "Improving the Nation's Cybersecurity", Executive Order 14028, 86 FR 26633, Document number 2021- 10460. Available at (https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/)
FIPS140	National Institute of Standards and Technology. 2019. "Security Requirements for Cryptographic Modules." Available at (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf).
IDASOAR	Hong Fong EK, Wheeler D, Henninger A (2016) State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016. (Institute for Defense Analyses [IDA], Alexandria, VA), IDA Paper P-8005. Available at (https://www.ida.org/research-and-publications/publications/all/s/st/stateoftheartresources-soar-for-software-vulnerability-detection-test-and-evaluation-2016)
INTEL	Intel. Software Supply Chain Threats; A White Paper
ISO27034	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, 2011. Available at (https://www.iso.org/standard/44378.html)
MITRE_CAPEC	MITRE. 2021. Common Attack Pattern Enumeration and Classification. Available at (https://capec.mitre.org/data/definitions/437.html)
MITRE_CVE	MITRE. 2021. "Common Vulnerability and Exposure, CVE." 2021. Available at (https://cve.mitre.org/index.html).
MSSDL	Microsoft (2019) Security Development Lifecycle. Available at https://www.microsoft.com/en-us/sdl
NASASTD8739	National Aeronautics and Space Administration. 2021. "SOFTWARE ASSURANCE AND SOFTWARE SAFETY STANDARD, NASA-STD-8739.8A." Available at

	(https://standards.nasa.gov/sites/default/files/standards/NASA/PUBLISHED/A 1/nasa-std-8739.8a.pdf).
NICCS	National Initiative for Cybersecurity Careers and Studies, National Initiative for Cybersecurity Education. 2021 Workforce Framework for Cybersecurity (NICE Framework). Available at (https://niccs.cisa.gov/workforce-development/cyber-security-workforce-framework)
NISTCSF	National Institute of Standards and Technology. 2018. "Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1." Available at (https://doi.org/10.6028/NIST.CSWP.04162018)
NISTMSDV	National Institute of Standards and Technology. 2018. "Guidelines on Minimum Standards for Developer Verification of Software". available at (https://www.nist.gov/system/files/documents/2021/07/13/Developer%20Verification%20of%20Software.pdf)
NTIASBOM	National Telecommunications and Information Administration. 2021. "The Minimum Elements for a Software Bill of Materials (SBOM)." Available at (https://www.ntia.doc.gov/report/2021/minimum-elements-software-bill-materials-sbom)
NVD	National Vulnerability Database. Available at (https://www.nist.gov/programs-projects/national-vulnerability-database-nvd)
OWASP_ASVS	Open Web Application Security Project (2019) OWASP Application Security Verification Standard 4.0. Available at https://github.com/OWASP/ASVS
OWASP_SAMM	Open Web Application Security Project (2017) Software Assurance Maturity Model Version 1.5. Available at (https://www.owasp.org/index.php/OWASP_SAMM_Project)
OWASP_SCVS	OWASP. 2021. "OWASP Software Component Verification Standard." Retrieved Sep. 25, 2021 (https://owasp.org/www-project-software-component-verification-standard/).
OWASP_TEST	Open Web Application Security Project (2014) OWASP Testing Guide 4.0. Available at https://www.owasp.org/images/1/19/OTGv4.pdf
PCI_SSLRAP	Payment Card Industry (PCI) Security Standards Council (2019) Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures Version 1.0. Available at (https://www.pcisecuritystandards.org/document_library?category=sware_sec #results)
SC_AGILE	Software Assurance Forum for Excellence in Code (2012) Practical Security Stories and Security Tasks for Agile Development Environments. Available at (http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf)
SC_FPSSD	Software Assurance Forum for Excellence in Code (2018) Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program, Third Edition. Available at (https://safecode.org/wpcontent/ uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Dev elopment_March_2018.pdf)

SC_SIC	Software Assurance Forum for Excellence in Code (2010) Software Integrity Controls: An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain. Available at (http://www.safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf)
SC_TPC	Software Assurance Forum for Excellence in Code (2017) Managing Security Risks Inherent in the Use of Third-Party Components. Available at (https://www.safecode.org/wpcontent/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf)
SC_TTM	Software Assurance Forum for Excellence in Code (2017) Tactical Threat Modeling. Available at (https://www.safecode.org/wpcontent/uploads/2017/05/SAFECode_TM_Whitepaper.pdf)
SLSA	The Linux Foundation. 2021. "Improving artifact integrity across the supply chain – SLSA." Available at (https://slsa.dev/)
SP80050	National Institute of Standards and Technology. 2021. "PRE-DRAFT Call for Comments: Building a Cybersecurity and Privacy Awareness and Training Program, SPS 800-50 Rev 1." Available at (https://csrc.nist.gov/publications/detail/sp/800-50/rev-1/draft)." Retrieved Sep. 25, 2021.
SP80052	National Institute of Standards and Technology. 2020. "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, SP 800-52 Rev. 2." Available at (https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final).
SP80053	National Institute of Standards and Technology. 2020. "Security and Privacy Controls for
SP80057	National Institute of Standards and Technology. 2020. "Recommendation for Key Management: Part 1 – General, SP 800-57 Part 1 Rev. 5." Available at (https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final)
SP800160	National Institute of Standards and Technology. 2018. "Systems Security Engineering." Available at (https://doi.org/10.6028/NIST.SP.800-160v1)
SP800161	"National Institute of Standards and Technology. 2021. ""Supply Chain Risk Management Practices for Federal Information Systems and Organizations."" Available at (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161.pdf)"
SP800172	"Enhanced Security Requirements for Protecting Controlled Unclassified Information: A Supplement to NIST Special Publication 800-171. Available at (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-172.pdf)
SP800175B	National Institute of Standards and Technology. 2020. "Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms. SP 800-175B Rev. 1." Available at (https://csrc.nist.gov/publications/detail/sp/800-175b/rev-1/final).

SP800181	National Institute of Standards and Technology, National Initiative for Cybersecurity Education. 2020. "Workforce Framework for Cybersecurity (NICE Framework)." Available at (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-181r1.pdf)
SP800193	National Institute of Standards and Technology. 2018. "Platform Firmware Resiliency Guidelines, SP-800-193." Available at (https://csrc.nist.gov/publications/detail/sp/800-193/final).
SP800207	National Institute of Standards and Technology. 2020. "Zero-Trust Architecture, SP-800-207." https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
SSDF	National Institute of Standards and Technology. 2020. "Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)." Available at (https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04232020.pdf)
SWEBOK3	IEEE Computer Society. 2014. Guide to the Software Engineering Body of Knowledge. Available at (https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3)
SYNOPSYS	Synopsys. 2021. "Synopsys Information Security Requirements for Vendors." Available at https://www.synopsys.com/company/legal/info-security.html
ZDNET	IBM, ZDNET. 2021. "Managing a Software as a Vendor Relationship: Best Practices". Available at (https://www.zdnet.com/article/managing-a-software-as-a-service-vendor-relationship-best-practices/)

3.6   Appendix F: Acronyms

Acronym	Meaning
ASLR	Address Space Layout Randomization
CI/CD	Continuous Integration/Continuous Delivery
CNSSI	Committee on National Security Systems Instruction
CVSS	Common Vulnerability Scoring System
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DLP	Data Loss Prevention
EO	Executive Order
EOL	End of Life
FedRAMP	Federal Risk and Authorization Management Program
FIPS	Federal Information Process Standards
HIPAA	Health Insurance Portability and Accountability Act
HSM	Hardware Security Module
HTTP(S)	Hypertext Transfer Protocol (Secure)
MFA	Multi Factor Authentication
NDA	Non-Disclosure Agreement
NIST	National Institute of Standards and Technology
NTIA	National Telecommunications and Information Administration
OSRB	Open-Source Review Board
OWASP	Open Web Application Security Project
PO	Prepare Organization
PS	Protect Software
PSIRT	Product Security Incident Response Team
PW	Produce Well-Secured Software
QA	Quality Assurance
RACI	Responsible, Accountable, Consulted, and Informed
RBAC	Role-Based Access Control
RM	Risk Management
RV	Respond to Vulnerabilities
SaaS	Software-as-a-Service
SAST	Static Application Security Testing
SBOM	Software Bill of Material
SCA	Software Composition Analysis

SCM	Supply Chain Management
SCM	Source Code Management
SCRM	Supply Chain Risk Management
SCVS	Software Component Verification Standard
SDLC	Software Development Lifecycle
SLSA	Supply-chain Levels for Software Artifacts
SSDF	Secure Software Development Framework
TLS	Transport Layer Security
VCS	Version Control System
VM	Virtual Machine
VPN	Virtual Private Network