

OpenSSF SLSA & NIST SSDF:
**Emerging Software Supply Chain Security
Best Practices**



About Moi



Tony Loehr

Developer Advocate & Evangelist

Twitter: @forkbombETH

- Software Engineer, Intuit Experimentation Platform @ Intuit
- Software Engineer, Mint @ Intuit
- Contributor, SimBioSys Lab @ Emory University



Agenda

1 Introduction & the rise of software supply chain attacks

2 NIST SSDF

3 Google SLSA

4 Comparing SSDF & SLSA

5 Covering gaps

6 Demo

7 Q&A



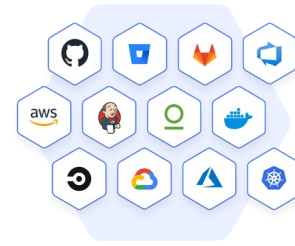
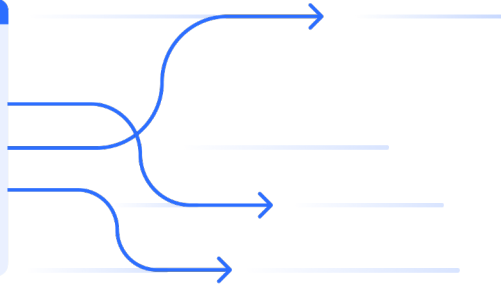
Why do we need new appsec frameworks?



Attackers are Shifting Priorities



Production Apps

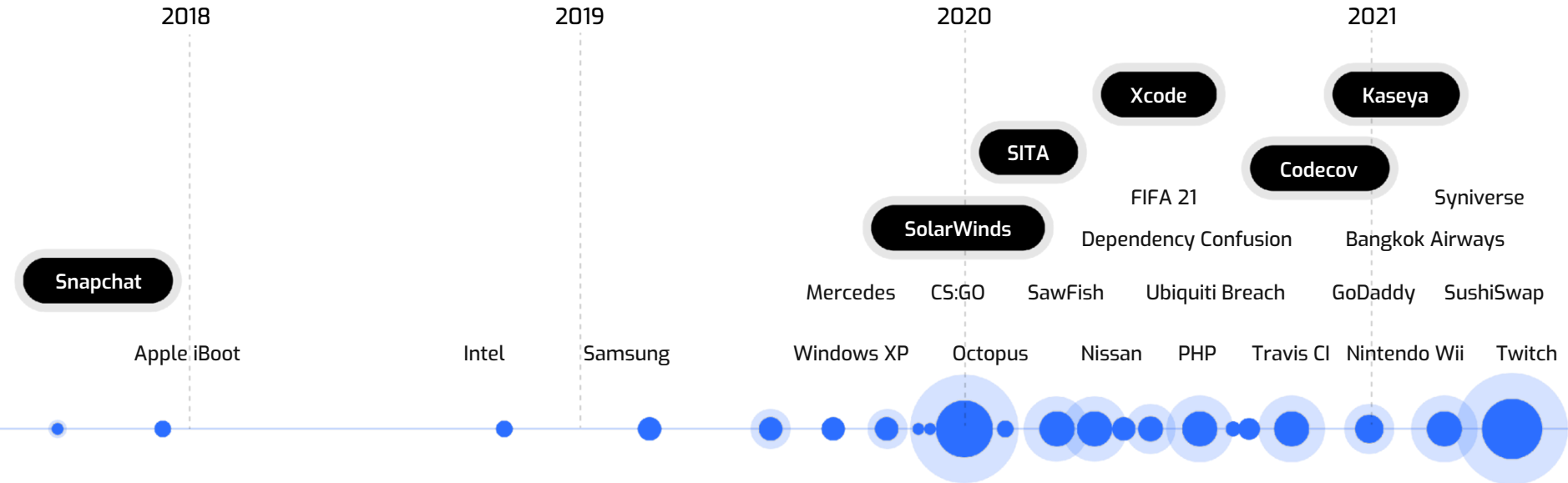


Software Delivery Pipelines



Developers

Software Supply Chain Attacks are on the Rise

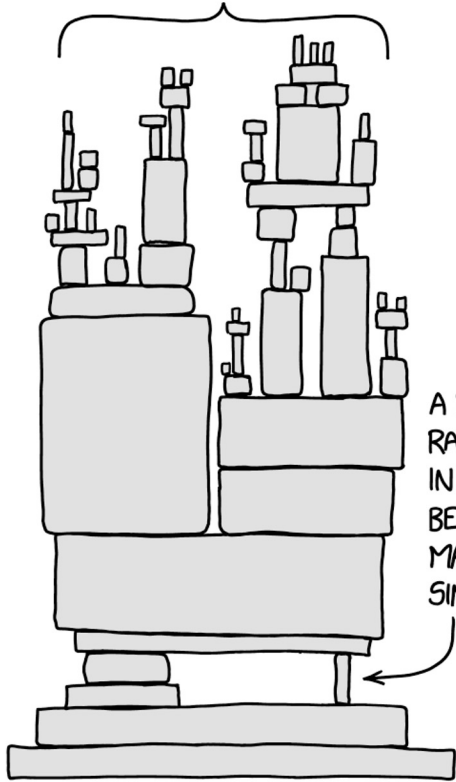


“By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021.”

Gartner



ALL MODERN DIGITAL INFRASTRUCTURE



A PROJECT SOME
RANDOM PERSON
IN NEBRASKA HAS
BEEN THANKLESSLY
MAINTAINING
SINCE 2003




JUNE 1, 2022

TypoSquatting, RepoJacking, and Domain Takeover – The Story of the Recent Attacks

We've had a busy month in terms of software...


 Alex Ilgayev
Security Researcher




APRIL 19, 2022

GitHub OAuth Compromise Affecting Heroku and Travis-CI Users

On April 15, GitHub Security announced that it experienced a software supply chain attack on many of its private repositories due...


 Tony Loehr
Developer Advocate




DECEMBER 16, 2021

Two Ways to Address the Log4J Vulnerability

Researchers have released patches for the log4j vulnerability, allowing some organizations to breathe a sigh of...


 Tony Loehr
Developer Advocate




MAY 22, 2022

Security Advisory: CrateDepression

CrateDepression is a software supply chain attack designed to target GitLab CI Pipelines by impersonating legitimate Rust...


 Tony Loehr
Developer Advocate



JULY 11, 2022

Security Advisory: IconBurst Attack

The IconBurst attack is a software supply chain attack designed to grab data from apps and websites. This attack campaign seeks...

 Tony Loehr
Developer Advocate



NIST SSDF

Secure Software Development Framework



NIST SSDF Heritage

- Directly inspired by OWASP SAMM
- Lists Gartner, the White House, and the Department of Defense as general resources

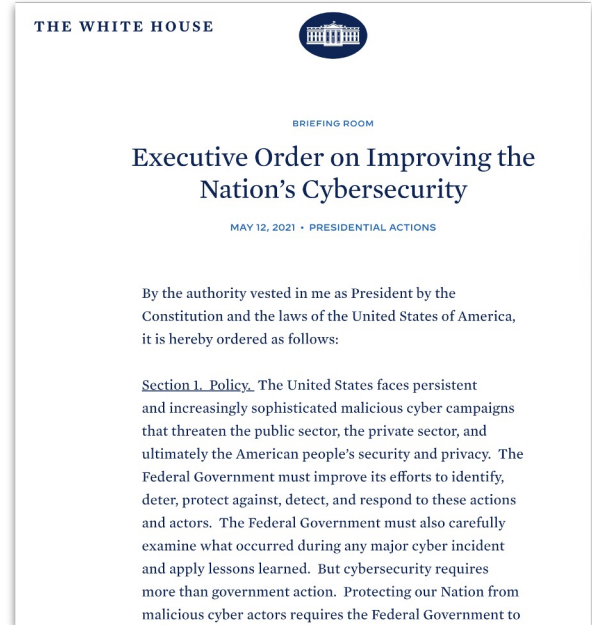


The screenshot shows the NIST Information Technology Laboratory Computer Security Resource Center website. The header includes the NIST logo and the text "Information Technology Laboratory" and "COMPUTER SECURITY RESOURCE CENTER". A green button labeled "PROJECTS" is visible. The main content area features the title "Secure Software Development Framework SSDF" and a paragraph of text describing the framework as a set of fundamental, sound, and secure software development practices based on established secure software development practice documents from organizations such as BSA, OWASP, and SAFECode. It notes that few software development life cycle (SDLC) models explicitly address software security in detail, so practices like those in the SSDF need to be added to and integrated with each SDLC implementation. A second paragraph states that following the SSDF practices should help software producers reduce the number of vulnerabilities in released software, reduce the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent recurrences. It also mentions that the SSDF provides a common language for describing secure software development practices, allowing software producers and acquirers to use it to foster their communications for procurement processes and other management activities.

Presidential Executive Order

“Improving the Nation’s Cybersecurity (14028)” issued on May 12, 2021

- Coordinates across multiple federal agencies
- Software supply chain security and integrity are major focus
- Section 4 of the EO directs NIST to:
 - + Solicit input from the private sector, academia, government agencies, and others
 - + Identify existing or develop new standards, tools, best practices, and other guidelines to enhance software supply chain security

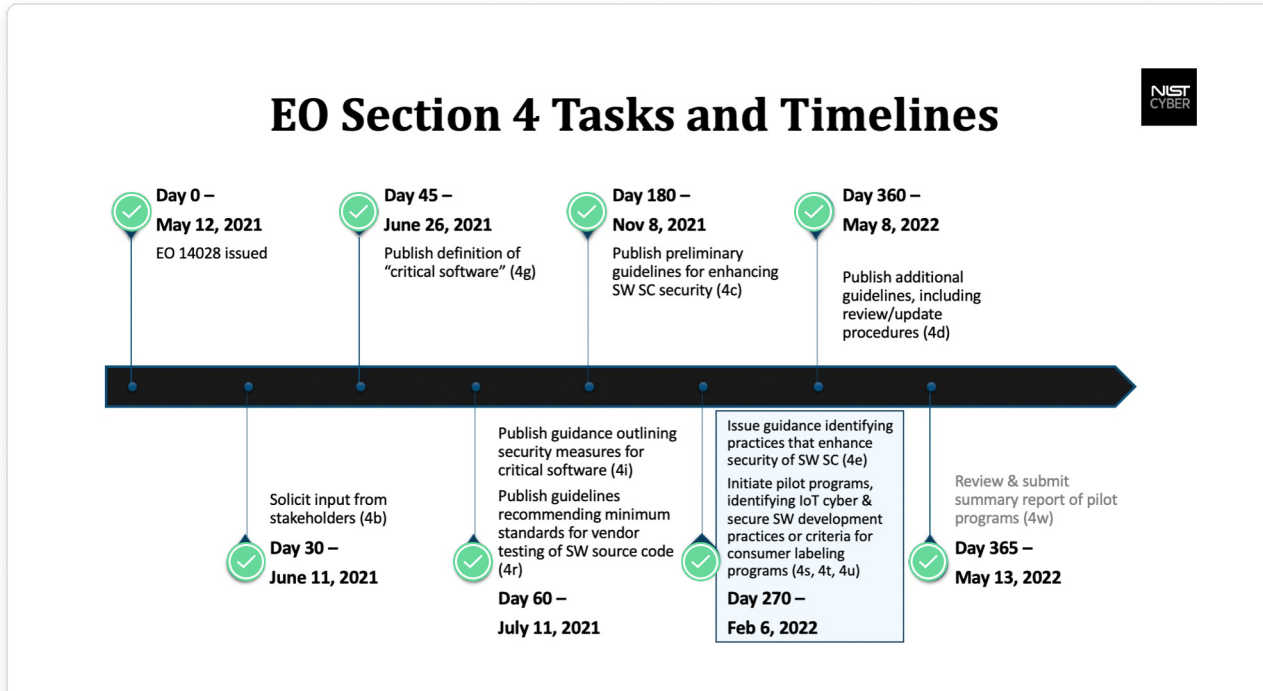


Software supply chain security

Sec. 4. *Enhancing Software Supply Chain Security.* (a) The security of software used by the Federal Government is vital to the Federal Government's ability to perform its critical functions. The development of commercial software often lacks transparency, sufficient focus on the ability of the software to resist attack, and adequate controls to prevent tampering by malicious actors. There is a pressing need to implement more rigorous and predictable mechanisms for ensuring that products function securely, and as intended. The security and integrity of “critical software”—software that performs functions critical to trust (such as affording or requiring elevated system privileges or direct access to networking and computing resources)—is a particular concern. Accordingly, the Federal Government must take action to rapidly improve the security and integrity of the software supply chain, with a priority on addressing critical software.

Presidential Executive Order

NIST



NIST Tasks

Critical Software

**First, NIST is to define “critical software”
by June 26, 2021**

Will consult with:

- National Security Agency (NSA)
- Office of Management and Budget (OMB)
- Cybersecurity & Infrastructure Security Agency (CISA)
- Director of National Intelligence (DNI)

**Second, NIST is to publish security
measure guidance by July 11, 2021**

Will consult with:

- Office of Management and Budget (OMB)
- Cybersecurity & Infrastructure Security Agency (CISA)

These guidelines will outline security measures for critical software



NIST Tasks

Software Supply Chain Security

The Executive Order (EO) on Improving the Nation's Cybersecurity (14028) directs

NIST released documents to enhance software supply chain security:

Secure Software Development Framework (SSDF)

- First draft
Version 1.1 (September 30, 2021)
- Released in response to Section 4e.

NIST SP 800-161

- Second draft
Revision 1 (October 28, 2021)
- Released in response to Section 4c.

Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations (C-SCRM)

- Second draft



Cybersecurity Labeling for Consumers

Section 4 of the order directs NIST to consider consumer product labeling

- NIST shall educate the public on the cybersecurity capabilities
 - + **Internet-of-Things (IoT) devices**
 - + **Software development practices**
- NIST also is to consider ways to incentivize manufacturers and developers to participate in these programs

NIST will identify key elements of labeling program

- Define minimum requirements and desirable attributes
- Will specify desired outcomes
- Allows providers and customers to choose their best solutions

NIST plans to produce a final version of these criteria by February 6, 2022.

- IoT cybersecurity criteria for a consumer labeling program and
- Secure software development practices or criteria for a consumer software labeling program

NIST Tasks

Cybersecurity Labeling for Consumers

Encourage innovation in manufacturers' consumer-oriented IoT and software security efforts, leaving room for changes in technologies and the security landscape.

Be practical and not be burdensome to manufacturers and distributors.

Factor in **usability as a key consideration**.

Build on national and international **experience**.

Allow for diversity of approaches and solutions across industries, verticals, and use cases – **so long as they are deemed useful and effective for consumers**.



NIST SSDF

Critical Software Security Measures

Objective 1: Protect

Protect EO-critical software and EO-critical software platforms from unauthorized access and usage.

Objective 2: Confidentiality

Protect the confidentiality, integrity, and availability of data used by EO-critical software and EO-critical software platforms.

Objective 3: Identify (SBOM)

Identify and maintain EO-critical software platforms and the software deployed to those platforms to protect the EO-critical software from exploitation.

Objective 4: Rapid Response

Quickly detect, respond to, and recover from threats and incidents involving EO-critical software and EO-critical software platforms.

Objective 5: Training

Strengthen the understanding and performance of humans' actions that foster the security of EO-critical software and EO-critical software platforms.



Risk Severity Schema

Level	Type	Description
01	Agency Low or Moderate Risk	Adversarial or non-adversarial risk is assessed, which falls within the agency's risk tolerance thresholds. Assessed risk impact does not extend outside of the agency.
02	Agency High Risk	The adversarial or non-adversarial-related risk is associated with a critical supplier, critical system, or critical system component, and is assessed to have a high risk, per agency-established risk level assessment. Assessed risk impact does not extend outside of the agency.
03	Significant Risk	Adversarial-related significant risk assessed, with potential or known multi-agency/ mission(s) or Government-wide impact.
04	National Security Interest Risk	The adversarial-related significant risk with the potential to impact National Security Interest.
05	Urgent National Security Interest Risk	The adversarial-related significant risk with imminent or present impact to National Security Interest.

NIST SSDF

Recommended Minimum Standard for Vendor or Developer Verification of Code

Threat modeling helps identify key or potentially overlooked testing targets

- As testing is automated, it can be repeated often

Static analysis

- Use a code scanner to look for top bugs
- Review for hard-coded secrets

Dynamic analysis

- Run with built-in checks and protections
- Create “black box” test cases
- Create code-based structural test cases
- Use test cases created to catch previous bugs
- Run a fuzzer.
- Run a web app scanner (when relevant)

Check included software (SBOM),
Fix critical bugs that are uncovered



NIST SSDF

Key Practices

Prepare the Organization

- Perform secure software development at the organization level
- In some cases, this is required for each individual project

Protect the Software

- Protect all components of the software from tampering and unauthorized access

Produce Well-Secured Software

- Produce well-secured software that has minimal security vulnerabilities in its releases

Respond to Vulnerabilities

- Identify vulnerabilities in software releases
- Respond appropriately to address those vulnerabilities
- Prevent similar vulnerabilities from occurring in the future



Google SLSA



Google's SLSA framework

Introducing SLSA, an End-to-End Framework for Supply Chain Integrity

June 16, 2021

Our proposed solution is **Supply chain Levels for Software Artifacts** (SLSA, pronounced "salsa"), an end-to-end framework for ensuring the integrity of software artifacts throughout the software supply chain. It is inspired by Google's internal "Binary Authorization for Borg" which has been in use for the past 8+ years and is mandatory for all of Google's production workloads.

The goal of SLSA is to improve the state of the industry, particularly open source, to defend against the most pressing integrity threats. With SLSA, consumers can make informed choices about the security posture of the software they consume.



Google's SLSA Levels

Level	Description	Example
01	Documentation of the build process	Unsigned provenance
02	Tamper resistance of the build services	Hosted source/build, signed provenance
03	Prevents extra resistance to specific threats	Security controls on hosts, non-falsifiable provenance
04	Highest levels of confidence and trust	Two-party review + hermetic builds



Google's SLSA Levels

Level	Requirements
00	No guarantees. SLSA 0 represents the lack of any SLSA level.



Google's SLSA Levels

01

The build process must be fully scripted/automated and generate provenance. Provenance is metadata about how an artifact was built, including the build process, top-level source, and dependencies. Knowing the provenance allows software consumers to make risk-based security decisions. Provenance at SLSA 1 does not protect against tampering, but it offers a basic level of code source identification and can aid in vulnerability management.



Google's SLSA Levels

02

Requires using version control and a hosted build service that generates authenticated provenance. These additional requirements give the software consumer greater confidence in the origin of the software. At this level, the provenance prevents tampering to the extent that the build service is trusted. SLSA 2 also provides an easy upgrade path to SLSA 3.



Google's SLSA Levels

03

The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively. We envision an accreditation process whereby auditors certify that platforms meet the requirements, which consumers can then rely on. SLSA 3 provides much stronger protections against tampering than earlier levels by preventing specific classes of threats, such as cross-build contamination.



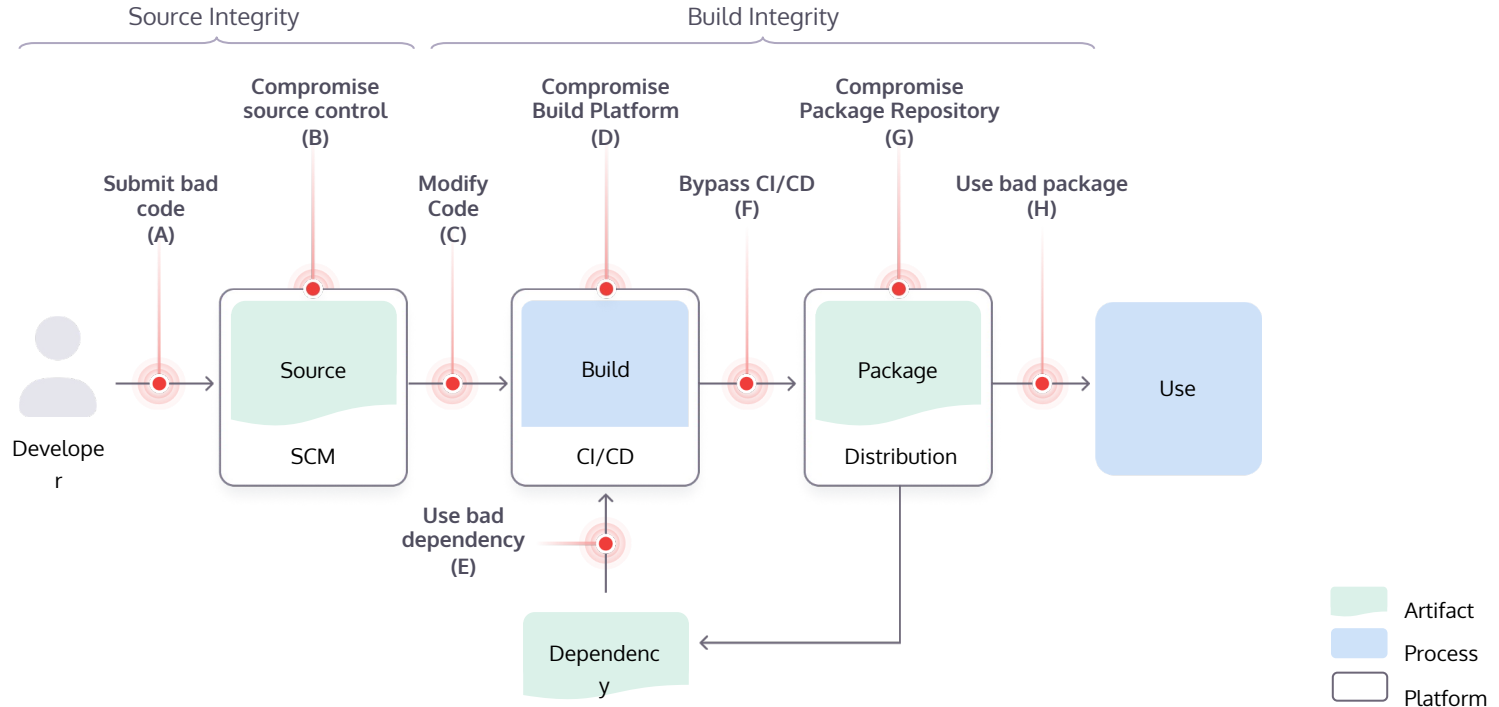
Google's SLSA Levels

04

Requires two-person review of all changes and a hermetic, reproducible build process. Two-person review is an industry best practice for catching mistakes and deterring bad behavior. Hermetic builds guarantee that the provenance's list of dependencies is complete. Reproducible builds, though not strictly required, provide many auditability and reliability benefits. Overall, SLSA 4 gives the consumer a high degree of confidence that the software has not been tampered with.



Google's SLSA framework



The 5 Categories of SLSA Requirements

01 / Source requirements

02 / Build process requirements

03 / Provenance generation requirements

04 / Provenance content requirements

05 / Common requirements



Google's SLSA framework

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source - Version Controlled		✓	✓	✓
Source - Verified History			✓	✓
Source - Retained Indefinitely			18 mo.	✓
Source - Two-Person Reviewed				✓
Build - Scripted Build	✓	✓	✓	✓
Build - Build Service		✓	✓	✓
Build - Ephemeral Environment			✓	✓
Build - Isolated			✓	✓
Build - Parameterless				✓
Build - Hermetic				✓
Build - Reproducible				○
Provenance - Available	✓	✓	✓	✓
Provenance - Authenticated		✓	✓	✓
Provenance - Service Generated		✓	✓	✓
Provenance - Non-Falsifiable			✓	✓
Provenance - Dependencies Complete				✓
Common - Security				✓
Common - Access				✓
Common - Superusers				✓



Google's SLSA framework

Source

Requirement	Description	SLSA
Version controlled	Every change to the source is tracked.	2
Verified history	Every change in the revision's history has at least one strongly authenticated actor identity (author, uploader, reviewer, etc.) and timestamp.	3
Retained indefinitely	The revision and its change history are preserved indefinitely and cannot be deleted.	4
Two-person reviewed	Every change in the revision's history was agreed to by two trusted persons prior to submission, and both of these trusted persons were strongly authenticated.	4



Google's SLSA framework

Build

Requirement	Description	SLSA
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.	1
Build service	All build steps ran using some build service, not on a developer's workstation.	2
Ephemeral environment	The build service ensured that the build steps ran in an ephemeral environment, such as a container or VM, provisioned solely for this build, and not reused from a prior build.	3
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.	3
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.	4
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.	4
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.	0



Google's SLSA framework

Provenance

Requirement	Description	SLSA
Available	The provenance is available to the consumer in a format that the consumer accepts. The format SHOULD be in-toto SLSA Provenance, but another format MAY be used if both producer and consumer agree and it meets all the other requirements.	1
Authenticated	The provenance's authenticity and integrity can be verified by the consumer. This SHOULD be through a digital signature from a private key accessible only to the service generating the provenance.	2
Service generated	The data in the provenance MUST be obtained from the build service (either because the generator is the build service or because the provenance generator reads the data directly from the build service). Regular users of the service MUST NOT be able to inject or alter the contents, except as noted below.	2
Non-falsifiable	Provenance cannot be falsified by the build service's users.	3
Dependencies complete	Provenance records all build dependencies that were available while running the build steps. This includes the initial state of the machine, VM, or container of the build worker.	4



Google's SLSA framework

Provenance Content

Requirement	Description	SLSA
Identifies artifact	The provenance MUST identify the output artifact via at least one cryptographic hash.	1
Identifies builder	The provenance identifies the entity that performed the build and generated the provenance.	1
Identifies source	The provenance identifies the source containing the top-level build script, via an immutable reference. Example: git URL + branch/tag/ref + commit ID.	1
Identifies entry point	The provenance identifies the "entry point" or command that was used to invoke the build script.	1
Includes all build parameters	The provenance includes all build parameters under a user's control. See Parameterless for details.	3
Includes all transitive dependencies	The provenance includes all transitive dependencies listed in Dependencies Complete.	4
Includes reproducible info	The provenance includes a boolean indicating whether build is intended to be reproducible and, if so, all information necessary to reproduce the build.	4
Includes metadata	The provenance includes metadata to aid debugging and investigations. This SHOULD at least include start and end timestamps and a permalink to debug logs.	0



Google's SLSA framework

Provenance Content

SBOM \neq Provenance

```
{
  "_type": "http://in-toto.io/Statement/v0.1",
  //Output file; name is "_" to indicate "not important".
  "subject": [{"name": "_", "digest": {"sha256": "5678..."}},
  "predicateType": "https://slsa.dev/provenance/v0.2",
  "predicate": {
    "buildType": "https://example.com/Makefile",
    "builder": {"id": "mailto:person@example.com"},
    "invocation": {
      "configSource": {
        "uri": "https://example.com/example-1.2.3.tar.gz",
        "digest": {"sha256": "1234..."},
        "entryPoint": "src:foo", //target "foo" in directory "Src"
      },
      "parameters": {"CFLAGS": "-O3"} //extra args to 'make'
    },
    "materials": [{
      "uri": "https://example.com/example-1.2.3.tar.gz",
      "digest": {"sha256": "1234..."}
    }
  ]
}
```



Google's SLSA framework

Common

Requirement	Description	SLSA
Security	The system meets some TBD baseline security standard to prevent compromise. (Patching, vulnerability scanning, user isolation, transport security, secure boot, machine identity, etc. Perhaps NIST 800-53 or a subset thereof.)	4
Access	All physical and remote access must be rare, logged, and gated behind multi-party approval.	4
Superusers	Only a small number of platform admins may override the guarantees listed here. Doing so MUST require approval of a second platform admin.	4



Key Learnings from NIST SSDF and Google SLSA



SSDF focuses on “what”

while

SLSA focuses on “how”



NIST SSDF and Google SLSA

Complementary Frameworks

NIST focuses on

“what”

The **NIST SSDF** is focused on defining minimum requirements for software used within critical infrastructure, particularly federally

Google focuses on

“how”

Google proposes a specific model for scoring the supply chain, focused on improving security within the build phase throughout deployment



NIST SSDF and Google SLSA

Complementary Frameworks

Tiers VS Levels

Tiers and levels both range from 1-4, with higher numbers correlating to increased cybersecurity

Higher tiers within the NIST SSDF represent increasing degrees of rigor and sophistication in cybersecurity risk management practices

Higher levels within SLSA represent greater maturity; each level of SLSA acts as a milestone towards the eventual goal of achieving SLSA 4



Review for Hard-Coded Secrets

NIST: use heuristic tools
to examine the code for:

- hardcoded passwords
- private encryption keys



Review for Hard-Coded Secrets

Consider: where should your organization
look for hard-coded secrets?
to examine the code for:

- Source
- Build
- Registry
- Logs



Uniformly Enforce Security Policy

SLSA and NIST both highlight the importance of establishing baseline security standards to prevent compromise

SLSA: Common requirements

- Access
- Superusers
- Security

NIST 800-53 Security and Privacy Controls for Information Systems and Organizations

- Referenced explicitly by SLSA



Uniformly Enforce Security Policy

Consider:

- > How does your organization enforce security policy?
- > Does you enforce security policies on each tool in the pipeline?
- > Are you tracking the settings? Changes to settings?
- > Does your organization have **visibility** into the components utilized?



Detect and Remediate Misconfigurations

NIST: Configuration management

**Maintain system
configurations and inventories**

- Hardware
- Software
- Firmware
- Documentation

**Enforce security configuration
settings for IT products**



Detect and Remediate Misconfigurations

Consider:

- > Where are you scanning for misconfigurations?
- > How often do you scan?
- > Do you employ automated remediation?



Reduce Code Tampering Risk

SLSA

Enforce source requirements

- Two-person review

Enforce build requirements

- Ensure hermetic builds
- Utilize code signing



Reduce Code Tampering Risk

Consider:

- > Are you adhering to the principle of least privilege?
- > How does your organization enforce access control policies?
- > Do you enforce code signing? 2FA or MFA?
- > Are branch protections enabled?



Gaps:

Even as complementary frameworks, SLSA
and the SSDF don't cover everything that
should be done to mitigate risk



Identify Suspicious Behavior and Code Leaks

- Source code is a software company's intellectual property
- Neither NIST nor Google frameworks address this need
 - + SLSA suggests preventative measures
 - + SSDF suggests to have a contingency plan in place
 - + There is no direct guidance to identify leaks



Defense in Depth

Slow down attackers with each layer of protection

Least privileged access

- Across all DevOps tools

Harden configuration

- Strong version control policies
- Harden security of CI/CD pipelines
- Use trusted component registries
- Change default passwords

Verify integrity at every stage

Centrally manage policies

- Enforce them policies consistently across tools, teams & phases of the SDLC

Audit thoroughly, audit frequently – don't overlook the basics



Anomaly detection

Anomalies in access

- Access grants

Anomalies in configurations

- Monitor configuration changes
 - + In tools
 - + In code

Anomalies in commits

- Deviation from defined workflows
- Code the developer never touched before

Anomalies in Behavior

- Commits outside of the user's normal working hours
- Peer review from non-developer accounts
- Changes in work patterns for employees leaving the company



Learn More:



Q&A

<https://calendly.com/cycode-team/free-repo-audit>



Thank You

Tony Loehr

Developer Advocate

✉ tony@cycode.com

