Developing (and Evaluating) Secure Open Source Software (OSS)

David A. Wheeler
Director, Open Source Supply Chain Security
The Linux Foundation

dwheeler@linuxfoundation.org

Outline

- Open Source Software (OSS) is everywhere!
  - All software under attack & users are not updating to fixed versions of OSS
- How can OSS developers develop & distribute secure OSS (today)?
- How can potential OSS users select secure OSS (often by looking for that)?
- How OSS is developed & distributed - a supply chain (SC) model
  - … including how it’s attacked & some countermeasures
- What’s coming in the future? (SBOMs, etc.)
- Where could I get involved?

What is open source software (OSS)?

- OSS is software licensed to users with these freedoms:
  - to run the program for any purpose,
  - to study and modify the program, and
  - to freely redistribute copies of either the original or modified program (without royalties to original author, etc.)
- Full definition: Open Source Definition (Open Source Initiative)
- Common OSS licenses include MIT, Apache-2.0, BSD-3-Clause, LGPL, GPL
- Antonyms: Closed source, proprietary software
- OSS is a kind of commercial software (licensed to the general public)
- OSS licenses enable worldwide collaborative development of software

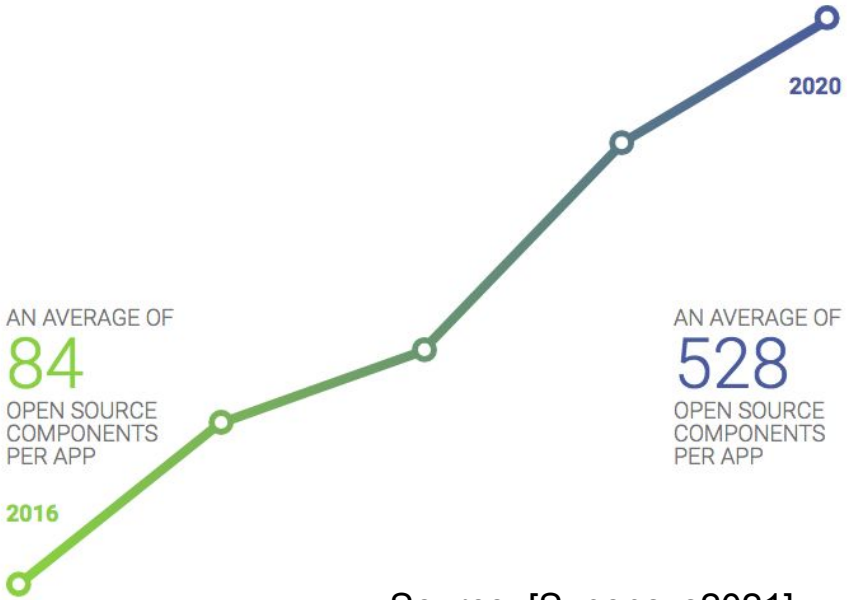
Open Source Source is critically important today

98%

Percent of general codebases and Android apps that contained OSS [Synopsys2021]

70%

Percent of codebase that was OSS on average [Synopsys2020]



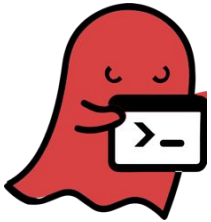
Source: [Synopsys2021]

Source:
[Synopsys2021] "2021 Open Source Security and Risk Analysis Report" by Synopsys https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html
[Synopsys2020] "2020 Open Source Security and Risk Analysis Report" by Synopsys https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2020-ossra-report.pdf

“ We’ve observed **double and triple digit growth** in open source component ecosystems for a decade, and there is **no slowdown in sight.**”

source: 2019 SonaType Report

All software under attack, via vulnerabilities & supply chain



GitHub hacked, millions of projects at risk of being modified or deleted

By Sebastian Anthony on March 5, 2012 at 7:22 am | 21 Comments

POLICY

Cleaning up SolarWinds hack may cost as much as $100 billion

Government agencies, private corporations will spend months and billions of dollars to root out the Russian malicious code

Source: https://www.rollcall.com/2021/01/11/cleaning-up-solarwinds-hack-may-cost-as-much-as-100-billion/

SolarWinds' Orion attacked via a *subverted build environment*

Computer Science > Cryptography and Security

[Submitted on 19 May 2020]

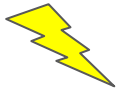
Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks

Marc Ohm, Henrik Plate, Arnold Sykosch, Michael Meier

A software supply chain attack is characterized by the injection of malicious code into a software package in order to compromise dependent systems further down the chain. Recent years saw a number of supply chain attacks that leverage the increasing use of open source during software development, which is facilitated by dependency managers that automatically resolve, download and install hundreds of open source packages throughout the software life cycle. This paper presents a dataset of 174 malicious software packages that were used in real-world attacks on open source software supply chains, and which were distributed via the popular package repositories npm, PyPI, and RubyGems. Those packages, dating from November 2015 to November 2019, were manually collected and analyzed. The paper also presents two general attack trees to provide a structured overview about techniques to inject malicious code into the dependency tree of downstream users, and to execute such code at different times and under different conditions. This work is meant to facilitate the future development of preventive and detective safeguards by open source and research communities.

Source: https://arxiv.org/abs/2005.09535

Reviewed 174 OSS SC attacks 2015-2019,
61% malicious packages use typosquatting



OSS Developer? Do this (now)

*If you're **evaluating** OSS, you should be looking for these!*

1. Learn how to develop & acquire secure software
   - *Secure SW Development Fundamentals* free course*, https://openssf.org/edx-courses/
   - Make software "secure by default" (easy to use securely, hardening)
2. Help projects earn CII Best Practices badge* https://bestpractices.coreinfrastructure.org/
3. Use many tools to find vulnerabilities via CI pipeline (build & verification environment)
   - Quality scanners (linters), security code scanners (static analysis), secret scanning, software composition analysis (SCA), web application scanners, fuzzers, …
   - One guide: https://github.com/ossf/wg-security-tooling/blob/main/guide.md
4. Monitor for known vulnerabilities in what you depend on
5. Enable rapid dependency update via using package managers & automated tests
   - Package managers may be system, language-level, and/or containers
   - Tests should include negative tests, be thorough enough to "ship if it passes"
6. Evaluate before selecting dependencies to use (typosquatting? malicious? secure?)
7. Make it *easy* for your users to update (e.g., stable APIs)
8. *Continuously improve* - attacks get better, so defenders also need to

Vulnerabilities are risks; manage the risks

* More information in next slides

Course: Secure Software Development Fundamentals

- Set of three *free* courses available on edX, *not* a huge time commitment
  - Requirements, Design, and Reuse (LFD104x), ~2-4 hours
  - Implementation (LFD105x), 4-6 hours
  - Verification and More Specialized Topics (LFD106x), ~3-5 hours

  1½ days over time
- Courses teach fundamentals of developing secure software (OSS or not)
  - Apply design principles (e.g., least privilege) & examine designs (threat modeling)
  - Use acceptlists (not denylists) to constrain untrusted inputs (greatly limiting attacks)
  - Know most common kinds of vulnerabilities (top 10, top 25) & how to prevent each one
  - Use hardening methods so bugs less likely to become vulnerabilities
  - Add many vulnerability detection tools (of different kinds) to CI pipeline (detect problems early)
  - Material specific to using/developing OSS (normal case)
- Each is a set of small modules, most with quizzes (to stay on track)
- You can pay to try to earn a certificate (evidence you learned the material)
- Text available online under CC-BY license (others can reuse it!)
- A project within the OpenSSF Best Practices Working Group (WG)
- See: https://openssf.org/edx-courses/

Apply security design principles

- (Architectural) Design = how problem divided into components & interactions
- Design principles = rules-of-thumb to help you avoid serious design flaws
- There are many time-tested security design principles, e.g.:
  - Least privilege
  - Complete mediation (aka non-bypassability) - client-side JavaScript & mobile apps failures
  - Simplicity
  - Open design
  - Fail-safe defaults
  - Two-factor authentication
  - Minimize sharing
  - Easy to use
  - Apply acceptlists on untrusted inputs (define what’s acceptable, reject the rest)

Know most common kinds of vulnerabilities & how to avoid

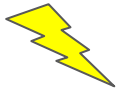
- Over 90% of vulnerabilities fit into a small set of categories
  - Knowing & preventing them reduces vulnerabilities by at least 1 order of magnitude
- Some widely-used carefully-crafted lists - know & use them
  - OWASP top 10 web application security vulnerabilities (for web apps)
  - CWE top 25 list (for anyone) + others “on the cusp”
- Examples of vulnerability categories
  - Injection vulnerabilities (e.g., SQL injection - counter with prepared statements)
  - Cross-site scripting (XSS)
  - Buffer overflows (and memory safety failures more generally. 70% Chrome & MS vulns)

Add vulnerability detection tools to CI pipeline

- Key: detect problems early
- Many different kinds of tools - include multiple kinds
- Practically all tools have false positives & false negatives
  - You still need to *think*
  - Try to have many tools (eventually)
- Greenfield (new start) vs. Brownfield (existing)
  - Greenfield: Add tools right now & make them sensitive (learn of & avoid problems)
  - Brownfield: Add tools slowly & greatly limit what they report, otherwise you'll be overwhelmed. Then increase sensitivity.

CII Best Practices Badge

- Identifies best practices for OSS projects
  - Goal: Increase likelihood of better quality & security. E.g.:
    - "The project sites… MUST support HTTPS using TLS."
    - "The project MUST use at least one automated test suite..."
    - "At least one static code analysis tool MUST be applied…"
    - "The project MUST publish the process for reporting vulnerabilities on the project site."
  - Based on practices of well-run OSS projects
- If OSS project meets best practice criteria, it earns a badge
  - Enables projects & potential users know current status & where it can improve
  - Combination of self-certification, automated checks, spot checks, public accountability
  - Three badge levels: passing, silver, gold
- Participation widespread & continuing to grow
  - >3,700 participating projects, > 500 passing+ projects in 2021-04
  - Current statistics: https://bestpractices.coreinfrastructure.org/en/project_stats
- A project within the OpenSSF Best Practices Working Group (WG)
- For more, see: https://bestpractices.coreinfrastructure.org



OSS User? Things to consider for evaluation (now)

1. Is there evidence that its developers *work to make it secure*?
   - See the previous “for developers” list!
   - E.g., CII Best Practices badge, security tools to detect vulnerabilities, documentation on why it’s secure, evidence of security audits, SAFECode *Principles for Software Assurance Assessment*
2. Is it *easy to use securely*? (e.g., defaults)
3. Is it *maintained*?
   - Recent commits, issues handled, releases, multiple developers (ideally from >1 organization)
   - Multi-organization governance model / governing board
4. Does it have *significant use*? (beware of fads, but no users==no reviewers)
5. What is the software’s *license*? (beware of no-license==not OSS)
   - Software Composition Analysis (SCA) tools, OpenChain*
6. If it is important, what is *your own evaluation* of the software? (OSS makes this possible!)*
   - Citizenship is not trustworthiness. Review the code/project you’re using instead*
7. Did you acquire (download) it securely?*

Source: “Secure Software Development Fundamentals” course, https://openssf.org/edx-courses/ & https://github.com/ossf/secure-sw-dev-fundamentals

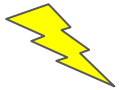
What is your own evaluation of the software?

- If the software is important to you, not examining it (OSS or not) is a risk
- Don’t be afraid; even a brief review of its code can give insight
  - Again, is there evidence that the developers were trying to develop secure software?
    - Rigorous validation of untrusted input, use of prepared statements, etc.
  - Evidence of insecure or woefully incomplete software (e.g., forest of TODO statements)?
    - Nothing made by humans is perfect, but is it adequate for purpose?
  - What are the “top” problems reported by tools that look for vulnerabilities?
    - All such tools have false positives, so you must actually look
    - Consider working with project to fix actual problems
  - Is there evidence it’s malicious?
    - Most malicious packages perform malicious actions during install, check those
    - Most aim at data exfiltration (check for extraction)
    - About half use obfuscation (look for obfuscation & encoded values that get executed)*
- What’s the likelihood that packages were generated from putative source code?
  - E.g., is the package (including container images) from originating project and/or trusted org?
- Many organizations can do an in-depth analysis for a fee

* See “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks”

Things to consider when downloading (now)

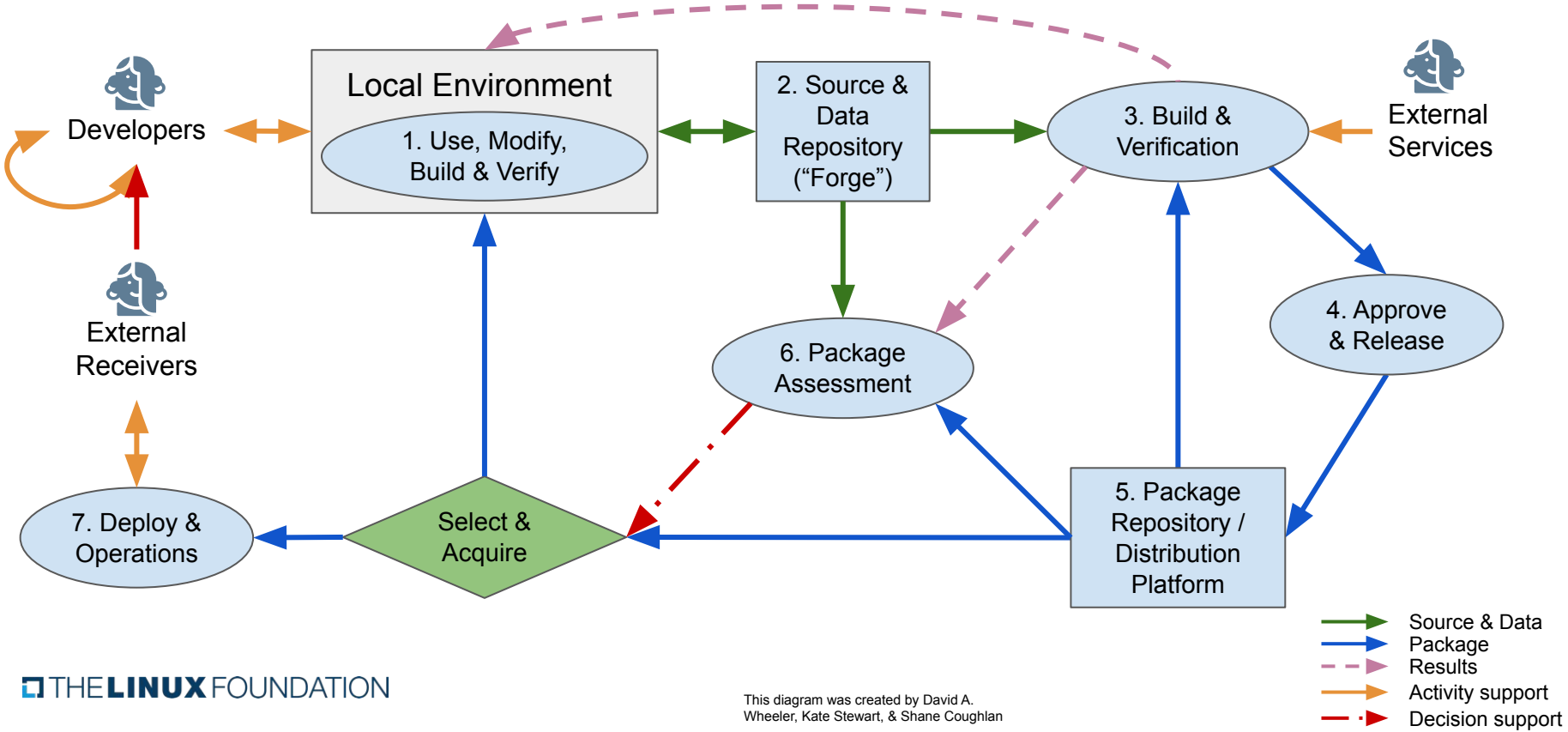
1. Make sure you have exactly the correct name before adding (counter typosquatting, most common OSS SC attack)
   a. Check for - vs _, 1 vs l, 0 vs O, Unicode
   b. Check popularity (download counts, followers, search engine) & date created
2. Download/install trustworthy way
   a. Use main site or “normal” redistribution site (e.g. package manager repository)
   b. Use https not http
   c. Download & delay (in case attack is revealed soon) - try to avoid running immediately
   d. Try to avoid using pipe-to-shell (e.g., curl … | sh) - cannot review, detectable by attackers
   e. If important & practical, try to verify that package is digitally signed by its expected creators



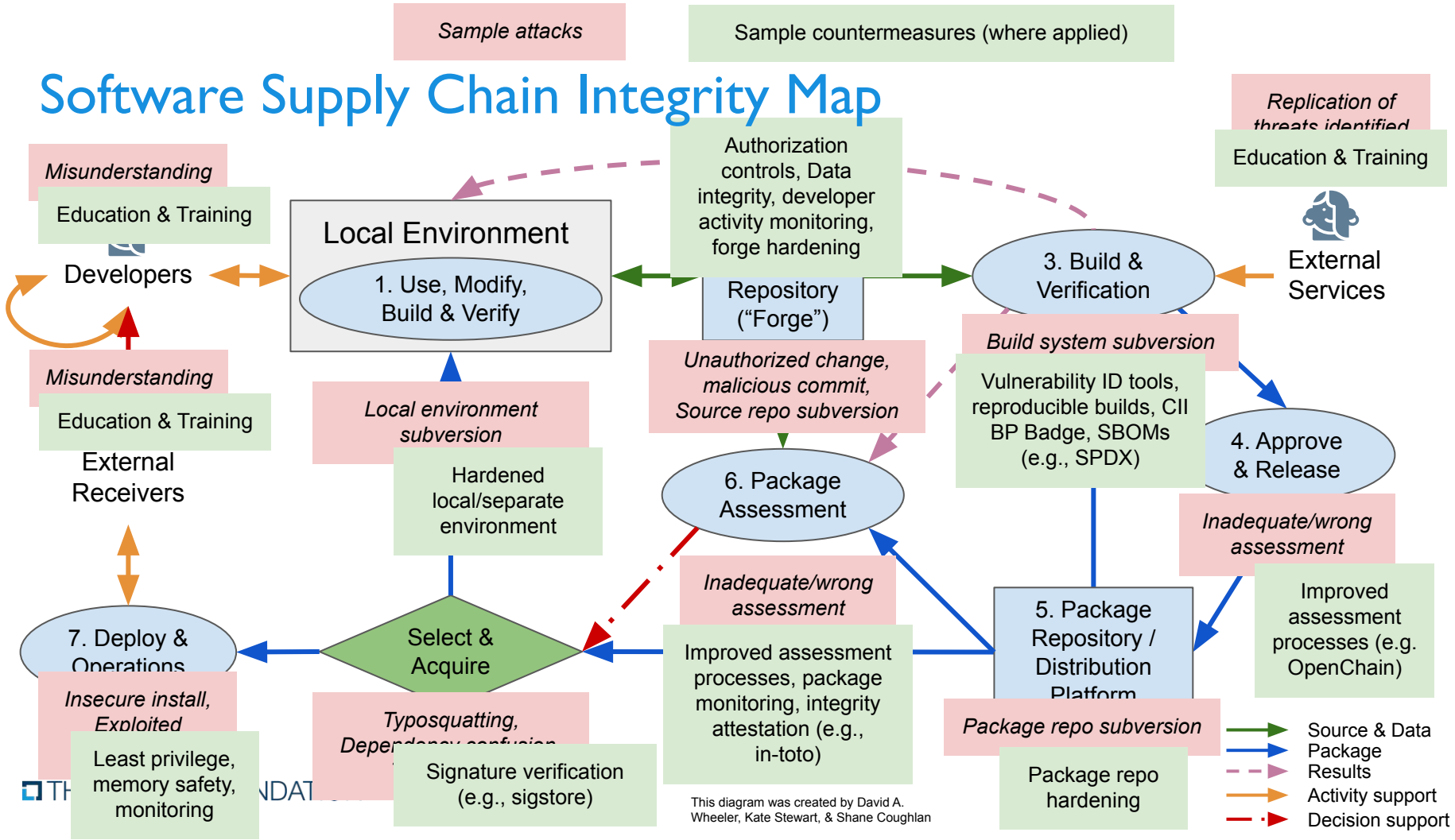
Things to consider when operating any software (now)

1. Protect, detect, respond
    - Protection is good, but you must *also* detect & respond to attacks in operations
    - Here “protect” includes “identify”, and “respond” includes “recover”
2. Constantly monitor for vulnerabilities in all dependencies (OSS or not)
    - SCA tools enable this
    - GitHub, GitLab, Linux Foundation’s LFX Tools, many others
3. If vulnerability found in dependency, examine quickly
    - If you *know* it can’t be exploited in your environment, fine!
    - Otherwise, rapidly update, test (automated tests), ship to production
    - You must be *faster* than the attackers

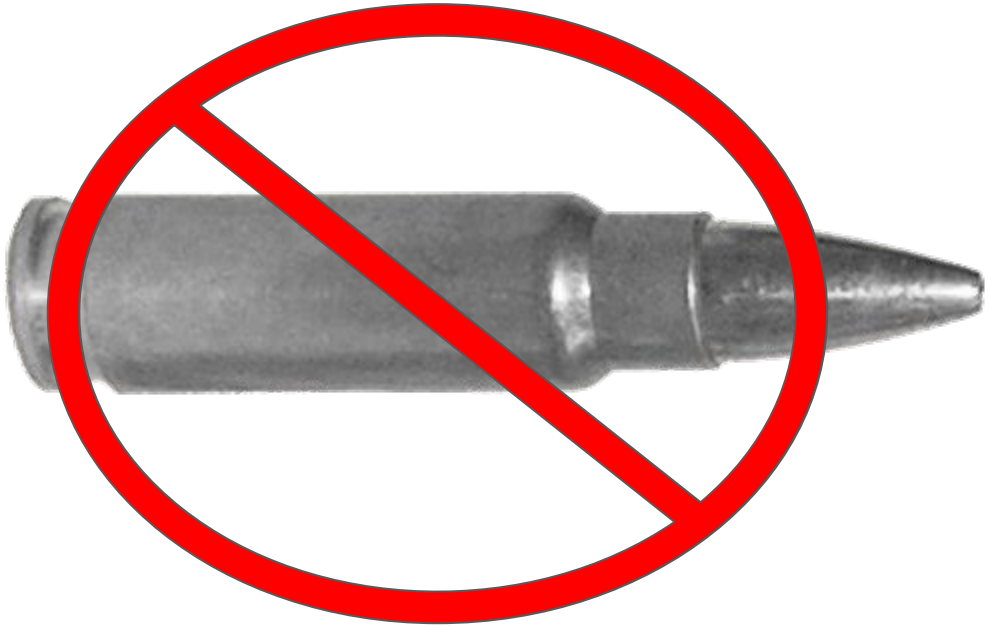
Software Supply Chain Integrity Map



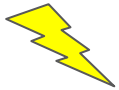
This diagram was created by David A. Wheeler, Kate Stewart, & Shane Coughlan


Sample attacks
Sample countermeasures (where applied)
Software Supply Chain Integrity Map
Misunderstanding
Education & Training
Developers
Misunderstanding
Education & Training
External Receivers
Local Environment
1. Use, Modify, Build & Verify
Local environment subversion
Hardened local/separate environment
Authorization controls, Data integrity, developer activity monitoring, forge hardening
Repository ("Forge")
Unauthorized change, malicious commit, Source repo subversion
6. Package Assessment
Inadequate/wrong assessment
Improved assessment processes, package monitoring, integrity attestation (e.g., in-toto)
Select & Acquire
Typosquatting, Dependency confusion
Signature verification (e.g., sigstore)
7. Deploy & Operations
Insecure install, Exploited
Least privilege, memory safety, monitoring
3. Build & Verification
Build system subversion
Vulnerability ID tools, reproducible builds, CII BP Badge, SBOMs (e.g., SPDX)
4. Approve & Release
Inadequate/wrong assessment
Improved assessment processes (e.g. OpenChain)
5. Package Repository / Distribution Platform
Package repo subversion
Package repo hardening
Replication of threats identified
Education & Training
External Services
Source & Data
Package
Results
Activity support
Decision support
This diagram was created by David A. Wheeler, Kate Stewart, & Shane Coughlan

There is no silver bullet!



Source: A silver bullet, Sir Magnus Fluffbrains, 2019, https://commons.wikimedia.org/wiki/File:Silver_bullet.png

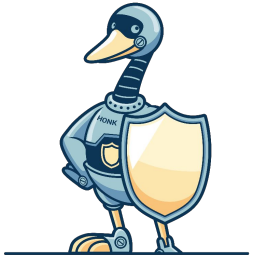


What's coming in the future? Best guesses...

1. More help when evaluating OSS
   - OpenSSF metrics.openssf.org* - provide data dashboard to ease evaluation of OSS
   - CHAOSS working to define more metrics
2. Wider use/adoption/requiring software bill of materials (SBOMs)
   - Package managers already track this within single ecosystems
   - SPDX 2.2 exists today, passed ISO ballot as ISO/IEC Draft International Standard (PRF) 5962
   - SPDX 3.0 in development (select what to exchange & more on identifying software)
   - DBOM in early stages, easier to exchange bill of materials
   - Look for more information about software before selecting
3. Package managers & repositories improved countermeasures
4. Verified reproducible builds*
5. Cryptographic signature verification (sigstore*, git signing agility)
6. Integrity attestation (e.g., in-toto, Alvarium - facilitate measurable trust and confidence)
7. Increased use of memory-safe / safe languages*
8. Formal methods in rare specialized tasks

… please work with others to help make things better!

Interested in improving OSS security? Get involved!



OpenSSF

OPEN SOURCE SECURITY FOUNDATION

sigstore

in-toto

OPENCHAIN

TODO

SPDX

OpenChain
Reference
Tooling

CHAOSS

Automating Compliance Tooling

… and **many** more!!

Developing & deploying secure software is a journey of learning & improving, not a singular event.

Some resources we discussed:

- Secure SW Development Fundamentals free course, https://openssf.org/edx-courses/
- https://bestpractices.coreinfrastructure.org/
- https://github.com/ossf/wg-security-tooling/blob/main/guide.md

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.

Backup slides

Is OSS or proprietary software always more secure?

- Neither. The reality is that neither OSS nor proprietary always more secure
  - If you care, evaluate
- A design principle gives OSS a *potential* security advantage
  - Saltzer & Schroeder [1974/1975] defined secure design principles still valid today
  - Open design principle: “the protection mechanism must not depend on attacker ignorance”
  - OSS better fulfills this principle
  - “Many eyes” theory does work
    - Academics, science & engineering already based on peer review
    - Security experts widely perceive OSS advantage
- No software is perfect, so vulnerabilities may be found in well-run projects
  - Continuous careful review is *more* likely to detect vulnerabilities over time

Common problem: Known-vulnerable reused software

- Problem: Failing to update reused known-vulnerable software (open & closed)
  - There's normally more OSS invisibly in use, so failure to update OSS is especially bad
- 84% of codebases had at least 1 OSS component with a known vulnerability, with an average of 158 vulnerabilities/codebase [Synopsys2021]
- Average known vulnerability was 2.2 years old [Synopsys2021]
- Android apps: 63% contain old OSS versions with known vulnerabilities; 44% of apps have high-risk vulnerabilities [Peril]
- 29% of codebases in 2019 AND 2020 included a vulnerable version of lodash fixed in July 2019 [Synopsys2021]
- 85% of codebases had OSS dependencies >4 years out of date [Synopsys2021]

Many fail to update their reused software (~OSS) when their vulnerabilities are fixed

Source: [Synopsys2021] "2021 OPEN SOURCE SECURITY AND RISK ANALYSIS REPORT" by Synopsys https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html ; [Peril] "Peril in a Pandemic: State of Mobile Application Security" by Synopsys;

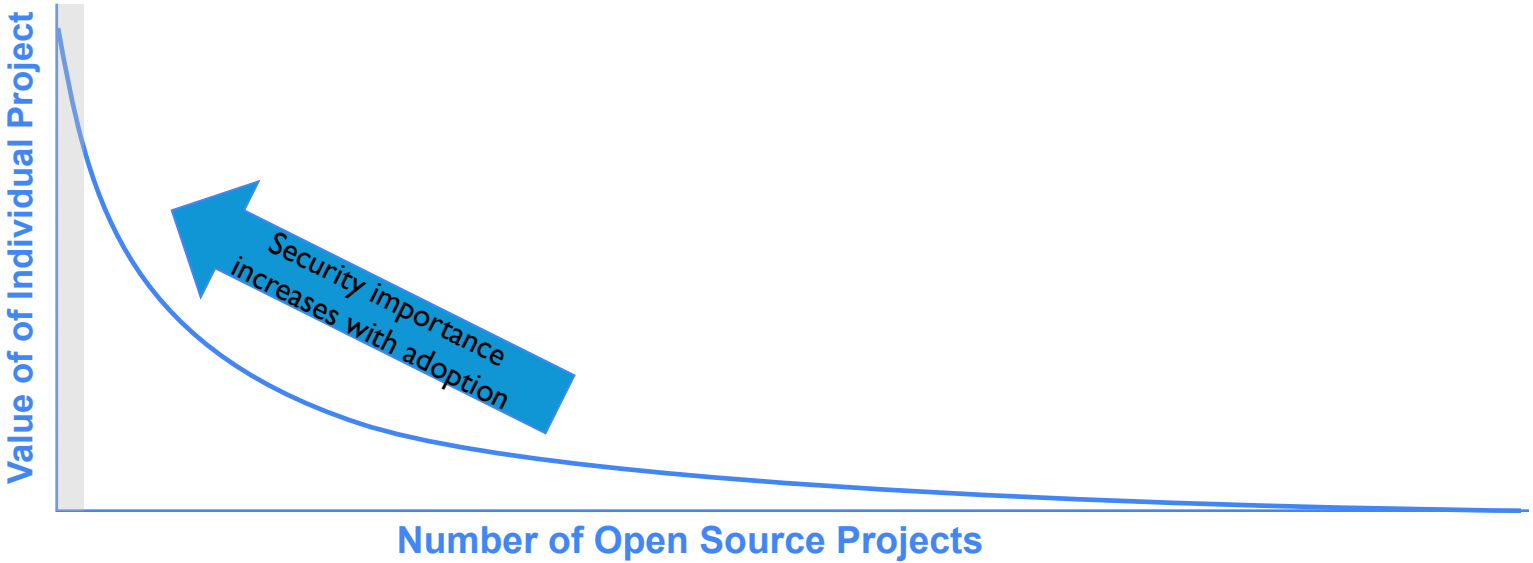
Can people insert malicious code in widely-used OSS?

- *Anyone* can insert malicious code into *any* software, proprietary or OSS
  - Just use a hex editor. Legal niceties are not protection
- Trick is to get result into user supply chain
- In OSS, requires:
  - subverting/misleading the trusted developers or trusted repository/distribution
  - *and* no one noticing the public malsource later
- Distributed source aids detection
- Large community-based OSS projects tend to have many reviewers from many countries
  - Makes undetected subversion more difficult

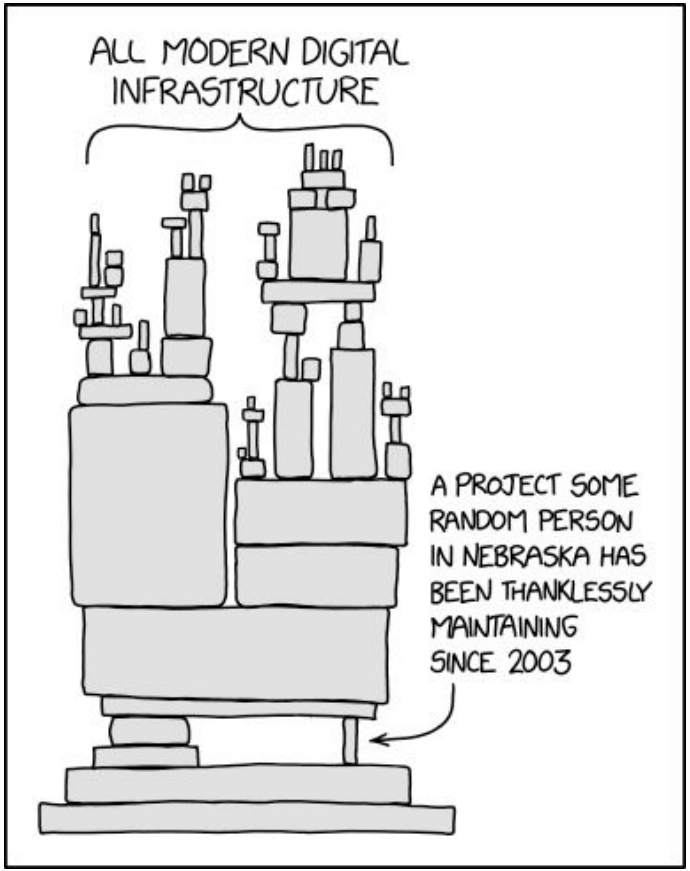
Malicious code & OSS

- OSS repositories demo great resilience vs. attacks
  - Linux kernel (2003); hid via “=” instead of “==”
    - if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
    - retval = -EINVAL;
    - Attack failed (CM, developer review, conventions)
  - SourceForge/Apache (2001), Debian (2003); Haskell (2015)
  - Countered & restored via external copy comparisons
- Linux kernel devs rejected all intentionally-vulnerable code from U of MN (2021)
- Malicious code can be made to look unintentional
  - Techniques to counter unintentional still vulnerabilities apply
- Attacker could try to bypass tools... but for OSS won't know what tools will be used!
- Borland InterBase/Firebird Back Door
  - user: politically, password: correct
  - Hidden for 7 years in proprietary product
  - Found after release as OSS in 5 months
  - Unclear if malicious, but has its form, & shows *improved* detection when switched to OSS

All open source projects are not equal: a very small subset of the millions of OSS projects are of importance to our collective security



Most projects & organizations are **not** able to **accurately summarize** the software that is running on their **systems**.



Source: https://xkcd.com/2347/ This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License.

"In any modern system, 80-90% of software is written and maintained by ***external parties.***

We need to create a **culture of secure coding** and back it up with **incentives, resources, and training**."

- Jim Zemlin, The Linux Foundation

Sample risks & countermeasures

- The following slides walk through the model
  - Show SOME risks & countermeasures
  - In some cases, developers would need to DO something, but evaluators can see what's done
  - For now, don't need to focus on the specific details

Key point: Attacks *can* be countered!

Developers & External Receivers

- Risk: Misunderstandings
  - Countermeasure: Education & training
  - Countermeasure: Processes to ensure best practices are consistently followed
  - Countermeasure: Tools, multi-party review to detect problems that slip in

1. Use, Modify, Build & Verify (in Local Environment)

- Risk: Local environment subversion
  - Countermeasure: Hardened local/separate environment
  - Countermeasure: Regular audits of the environment integrity

2. Source & Data Repository (“Forge”)

- Risk: Unauthorized change, malicious commit
  - Countermeasure: Authorization controls (e.g., 2FA)
  - Countermeasure: Data integrity analysis
  - Countermeasure: Developer activity monitoring
- Risk: Source repo subversion
  - Countermeasure: Forge hardening
  - Countermeasure: Forge audits

3. Build & Verification

- Risk: Build system subversion
  - Countermeasures: Build system hardening
  - Countermeasures: Software Bill of Materials (e..g., SPDX)
  - Countermeasures: Best practices (CII BP)
  - Countermeasures: Verified reproducible builds
  - Countermeasures: Vulnerability identification automation

4. Approve & Release

- Risk: Inadequate/wrong assessment
  - Countermeasure: Improved assessment processes (OpenChain ISO 5230)
  - Countermeasure: Automatically generation analysis SBOMs to automate policy checklists

5. Package Repository / Distribution Platform

- Risk: Package repo subversion
  - Countermeasure: Package repo / distribution platform hardening
  - Countermeasure: Code signing
  - Countermeasure: Verified reproducible builds

6. Package Assessment

- Risk: Inadequate/wrong assessment
  - Countermeasures: Software Bill of Materials (e.g., SPDX)
  - Countermeasures: Best practices (CII BP)
  - Countermeasures: Package monitoring
  - Countermeasures: Verified reproducible builds
  - Countermeasures: Integrity attestation (in-toto)

Select & Acquire

- Risk: Typosquatting (wrong package name)
  - Countermeasures: Software Bill of Materials
  - Countermeasures: Vulnerability identification automation
- Risk: Dependency confusion (wrong repository)
  - Countermeasures: Software Bill of Materials
- Risk: Selecting malicious packages
  - Countermeasures: Package monitoring
  - Countermeasures: Signature verification (sigstore)
- Risk: Selecting highly vulnerable packages
  - Countermeasures: Vulnerability identification automation

7. Deploy & Operations

- Risk: Insecure install
- Risk: Exploited vulnerabilities

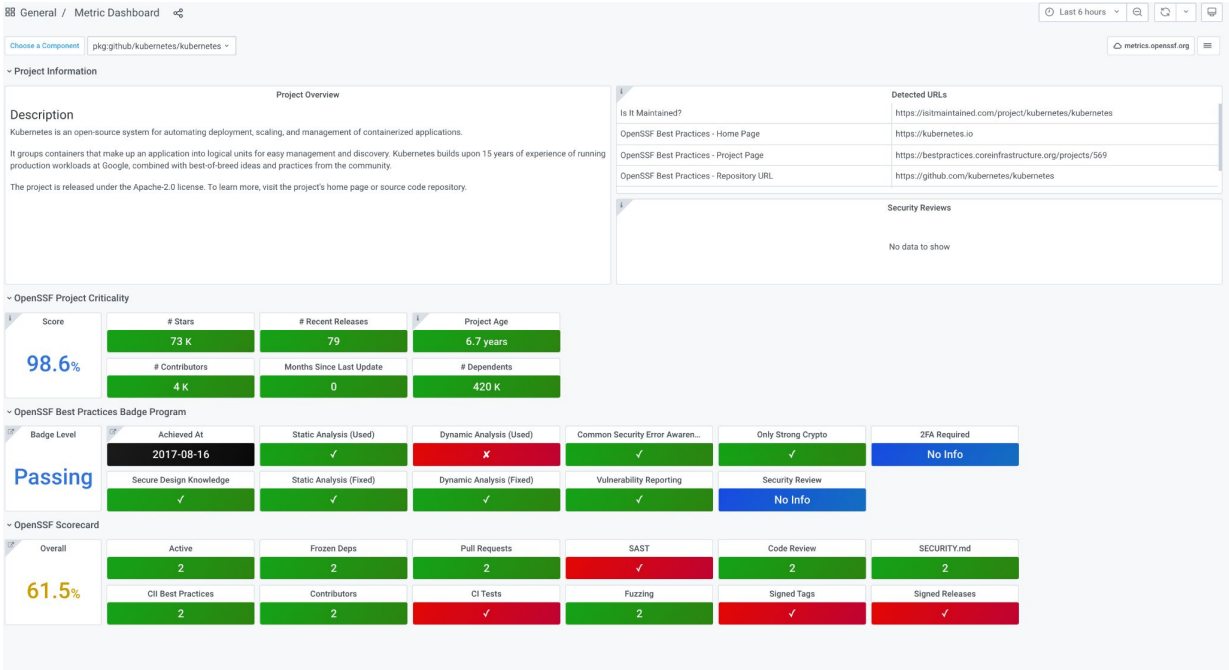
Countermeasures:

- Least privilege, memory safety, monitoring & response

Let’s talk about some of those solutions...

Don’t try to do everything;
Focus on what’s important first

metrics.openssf.org: Information to evaluate OSS



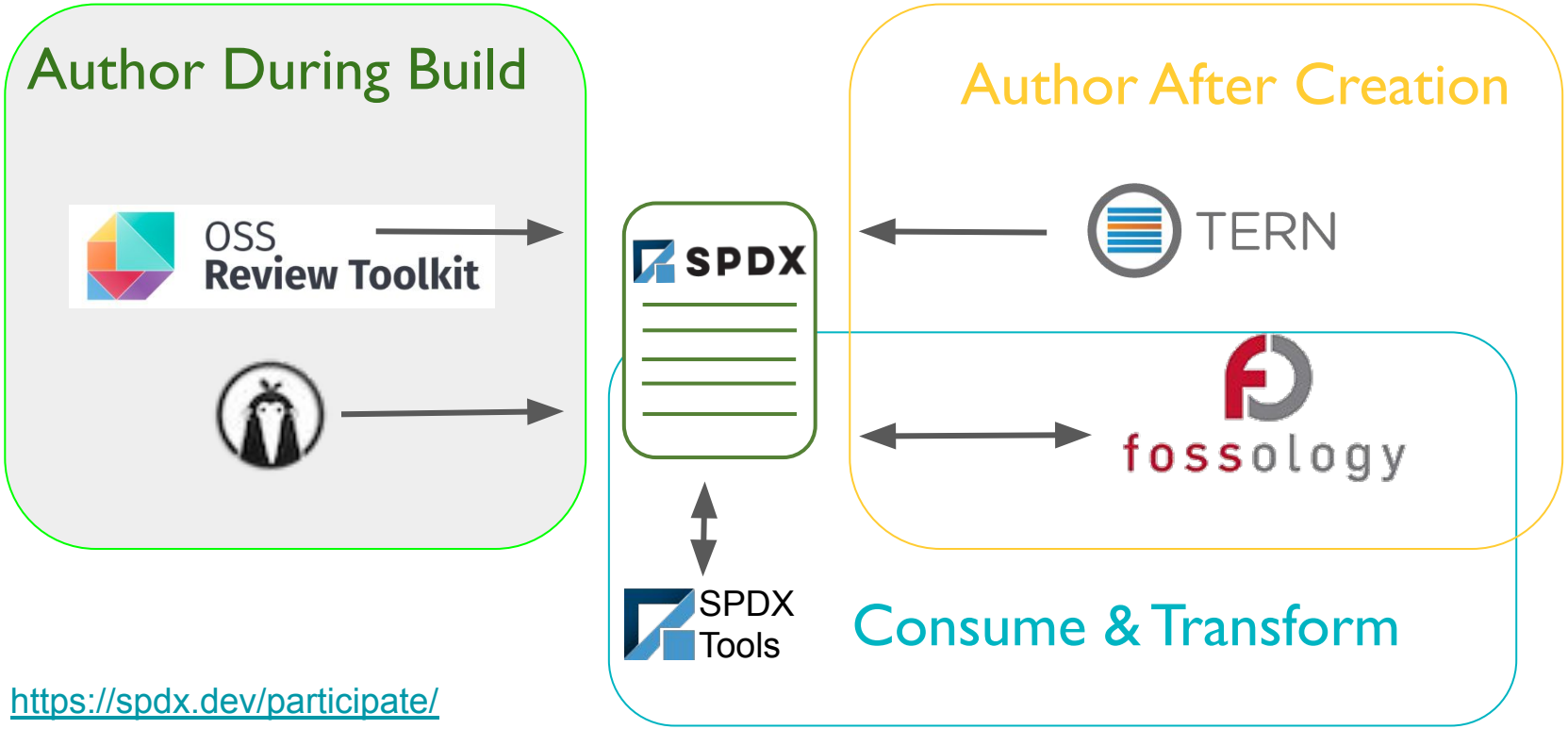
OpenSSF “metrics dashboard” in development to help ease evaluation

Verified reproducible builds

- SolarWinds' Orion (proprietary software) suffered a *build system* attack
  - The signed package received by customers was *not* generated from their source code
  - Most countermeasures fail against build system attacks, reproducible builds counter
- "Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code."
  - "By promising identical results are always generated from a given source, this allows multiple third parties to come to a consensus on a "correct" result, highlighting any deviations as suspect and worthy of scrutiny."
  - Goal: have independently-verified reproducible builds; attackers must break them all
- Significant progress (e.g., Debian bullseye/i386 94.0% reproducible 2021-04)
- Typically requires build changes (forced dates, forced sorts, etc.)
  - Possible for closed & open source software
  - Closed source has added doable challenges (keeping source secret, independently buildable)

Sources: https://reproducible-builds.org/, https://www.linuxfoundation.org/en/blog/preventing-supply-chain-attacks-like-solarwinds/

SPDX is being used as a common exchange standard for SBOM information



https://spdx.dev/participate/

sigstore: software signing service

- Tools currently exist to cryptographically sign OSS packages
  - No widely-practical mechanism to determine if public keys used are correct
  - No easy way to detect malicious signing
  - Key revocation typically impractical in practice
- sigstore (LF project) in development to be a free-to-use non-profit software signing service
  - Users generate ephemeral short-lived key pairs using the sigstore client tooling
  - sigstore PKI service provides a signing certificate generated upon a successful OpenID connect grant
  - All certificates are recorded in certificate transparency log
  - Software signing materials are sent to a signature transparency log
  - Guarantees that claimed user controlled their identity service providers' account at time of signing
  - Once the signing operation is complete, the keys can be discarded, removing any need for further key management or need to revoke or rotate.
- Using OpenID connect identities enables use of existing security controls such as 2FA, OTP and hardware token generators
- Transparency logs are public and open; anyone can monitor transparency logs for issues

Source: https://sigstore.dev/what_is_sigstore/

Increased use of memory-safe / safe languages

- Memory-safe/safe languages eliminate many security issues
  - Prevent many unintentional & mock-unintentional vulnerabilities
  - Most programming languages are memory-safe; use them when you reasonably can
  - C/C++/assembly are performant but not memory-safe/safe
  - Problem: Most languages cannot provide performance sometimes needed, C/C++ can
- Ongoing efforts to enable memory-safe languages in more situations
  - Esp. Rust; “safe” portion provides memory-safety & safe concurrency, yet performant
- Rustification: Transitioning (parts of) software to Rust, e.g.:
  - curl: optional http backend in Rust
  - Mozilla Firefox: increasing proportion in Rust
  - Linux kernel: ongoing effort to add support for drivers in Rust
- Many challenges (maturity issues), but promising
  - Rust currently has only 1 implementation, lacks support for some architectures (gcc ongoing)
  - Some capabilities need “Rust nightly builds” (but those extensions are becoming stable)

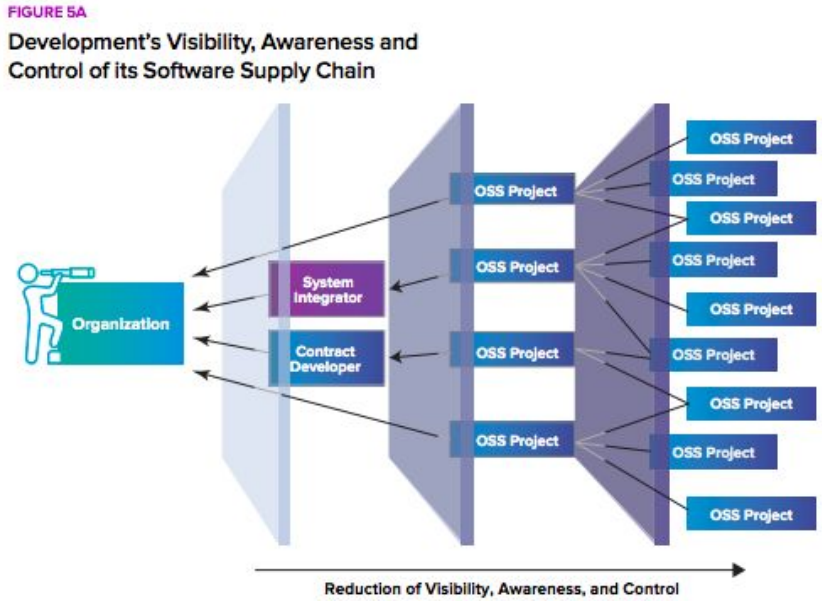
Open Source Security Foundation (OpenSSF)

- “Collaborating to secure the open source ecosystem”
- Established August 3, 2020: https://openssf.org/
  - As of Feb 2021: 36 member organizations, 263 individual participants
- Currently has 6 working groups:
  - Vulnerability Disclosures
  - Security Tooling
  - Security Best Practices
  - Identifying Security Threats to Open Source Projects
  - Securing Critical Projects
  - Digital Identity Attestation
- To improve *general* OSS security, get involved in the OpenSSF
- To improve security of specific OSS project/foundation/ecosystem...
  - Get involved with them!

Developer Citizenship: The wrong worry

- Commercial software (open & closed) is developed in a *global* economy
- It is *possible* to estimate probable OSS developer location & citizenship
  - Timezones & typical commit times. Also, presented name & email address are often available
  - Generally can't know these for closed source; sometimes only company location "known"
- But focusing on OSS location/citizenship is generally a *waste of time*
  - "Legal location of company" is ***irrelevant***, that's just a flag of convenience ($90 for Delaware LLC)
  - "Location of developer" is NOT the same as citizenship
  - "Citizen of [certain countries]" is NOT trustworthiness (Aldrich Ames, Robert Hanssen)
  - Powerful nation-states can bribe citizen or forge citizenship credentials if they care
  - Someone clearly developing code in an adversary state is probably not attacking
- Malicious state actor could attempt insertion, but OSS is less risky
  - With OSS you can review the code you'll use; closed source often requires blind faith
- With OSS, focus on evaluating *code*, not citizenship
  - This is how larger OSS projects counter potentially malicious developers
  - Esp. since top risks are unintentional vulnerabilities, old known-vulnerable code, typosquatting

Open Source Software Impact



Source: [2020 State of the Software Supply Chain]

User(s) of OSS Project
(Organizations, other projects, etc.)

Contributor(s) to an OSS Project

maintainer(s) of OSS Project

OPENCHAIN

OpenChain 2.1 (ISO/IEC 5230:2020) is the **International Standard** for open source license compliance.

It is simple, effective and suitable for **companies of all sizes in all markets**.

This standard is **openly developed** by a **vibrant user community** and **freely available** to all.

It is supported by free online **self-certification**, **reference material** and **service provider partners**.

www.openchainproject.org