A Supply Chain of Weak Links

Open Source Versus Proprietary Software Threat Analysis

A Technical Paper prepared for SCTE by

Brian A. Scriber
Distinguished Technologist and VP of Security Technologies
CableLabs
858 Coal Creek Circle, Louisville, CO 80027
𝕏 @brianscriber
b.scriber@cablelabs.com

Table of Contents

List of Tables

Abstract

Research explores the assumptions, resourcing, and maintenance realities of software in both closed and open ecosystems. This work is an evaluation of the Software Lifecycle using a security lens to highlight advantages and disadvantages of each approach at different development stages. An aggregation of risks and threats is provided to build an overview of the myths and realities of ecosystem transparency, modifiability, and ownership while answering questions about forking, hybridization, and proprietization. With recent supply-chain attacks in the networking industry, and identification of malicious actors within the open-source ecosystems, these macro-threats are evaluated for applicability to each approach: monoculture vulnerability analysis, presumption of security review, motivation for feature additions, and software patching.

Categories and Subject Descriptors

D.2.8 [SOFTWARE ENGINEERING]: Metrics

K.6.5 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS]: Security and Protection

K.4.1 [COMPUTERS AND SOCIETY]: Public Policy Issues

General Terms

Security, Economics

Keywords

Security, Metrics, Open source software, Closed source software, Economics, Policy, Monoculture

1. Introduction

There exists a debate centered on the security of open source versus closed source software; this is predicated upon the false idea that there is one choice that the enterprise or implementation team must make, and that after selecting one of these two models, the implications of that decision can be ignored going forward. This paper will address the metrics that go into these considerations, defining the legal frameworks and contribution models for the different types of software, but it does so not to provide an answer to either one of these being more secure than the other, but with the objective of identifying the threats particularly inherent to adoption of either. Awareness of the threats lends itself to the considered mitigation steps necessary to better buttress a software service to be resistant or resilient to those threats.

2. Defining Open Source and Proprietary Software

Without clarification of definitions, it is possible to get tangled into emotional debates that could distract from the point of this paper which is to identify the inherent threats different across the approaches. The definitions are important, and for other contexts, increasingly so, but for purposes here the classification is based on a threat analysis perspective, not as heavily upon the licensing or legal frameworks.

3. Software Licensing

There are different licensing models that often play into definitions of different software models, for purposes of this paper, one key element for consideration is if the license publishes the source code, allows for modification and whether it allows for commercial hybridization or proprietization, but this paper will not tackle an otherwise contentious issue of whether software licensing nuance makes software open or not. From a perspective to be reviewed later, determining who contributed, or potentially who and how the contribution was reviewed, would be interesting concepts to consider, and only a very few licenses are starting to explore those caveats.

Table 1 Activities Supported by Software License Model

License	Allows for Commercial Hybridization or Proprietization	Requires contributions be attributed to author	Includes explicit notes on contribution review
CC BY	Yes	Yes	No
CC BY-SA	Yes	Yes	No
CC (other)	No	No	No
GPL v3	Yes	No	No
Apache v2.0	Yes	General	No
MIT	Yes	No	No
Apple Public License 2.0	Limited	No	No
BSD 3.0	Yes	No	No
EUPL 1.2	Yes*	No	No
Open SSL 3 (Now Apache 2.0)	Yes	Some[i]	No

Free and and open also get confused when looking at software like Adobe Acrobat Reader, which is closed source, but offered free of charge, or tools like the Oracle MySQL database which is open source, but subject to payment requirements. For purposes of this paper, the actual payment requirements are less important than the development methodology.

4. Defining Free and Open Source Software

Free and Open-Source Software (FOSS) is commonly used term inclusive of both the free software and open-source software models. For the purposes here, we can cover all those open models listed above under Software Licensing as FOSS because the threats from the software development lifecycle will be similar and the resultant code from non-hybridized or proprietized software will be available for review.

This paper will look at any open contribution model software with subsequent open review of the source code as FOSS. Oracle's MySQL, given this definition, would be considered FOSS for purposes of this paper.

5. Defining Proprietary Software

Proprietary software shall be defined here as those solutions that are developed privately (either commercially or with no initial intention of public release of the final source code) and for which all contributors have a commercial agreement (employee, contractor, subcontractor or similar) for the final released solution. The development process and the resultant deliverables are controlled in a non-public manner. Adobe's Acrobat Reader would be considered proprietary for purposes of this paper.

6. Security Assumptions

While comparative security in FOSS vs proprietary code bases has been a topic of research and measurement[ii,iii], new threats have emerged during the decade since the some of the most recent novel work in this space and some salient questions remain in performing a comparison.

Outstanding questions for security considerations include the following: What metrics exist for comparative analysis? What do those metrics focus on? Is reliability/availability a reliable proxy for security? Who's writing the code? Who is reading the code? How frequently is the code reviewed and how efficient are those reviews at finding vulnerabilities before adversaries? What are the economic hurdles to getting access to the code or to modifying the code for malicious purposes? How do embedded software and vulnerabilities therein contribute to the overall security posture?

Analysis

7. Empirical vs. Intuitive

From a methodological perspective, even comparing published vulnerabilities in FOSS and proprietary solutions limits the sampling to only those known/public vulnerabilities in widespread software packages. Comparison between the mean time between vulnerability disclosure and fix has shown some advantages toward FOSS[iv].

While the population in that prior study was extremely limited, the impact of the time between discovery and fix is increasingly important because we see examples[v] of adversaries adopting attack behavior to leverage recent disclosures within hours of CVE publication (e.g., Citrix CVE-2022- 27518, Microsoft Exchange Server CVE-2022-41040, Confluence Data Center CVE-2022-26134). Metrics for security currently tend toward reliability and availability and less toward the frequency or severity of vulnerabilities in a software solution or toward the measurement of how widespread and broadly impacting a vulnerability in a particular software package might be (e.g., Log4Shell CVE-2021-44228). Without at least these three axes, comparisons are going to remain difficult.

Availability of metrics for those three dimensions (count, severity, and impact breadth) are difficult to obtain or even approximate, and the danger of comparing empirical data with intuitive assumptions is a significant risk to the industry.

8. Count vs Severity vs Impact Breadth

Modern software is manifold, and while legacy systems may have had a monolithic approach to all the software running in a solution, the modern software leverages packages and libraries in compiled code, it uses callouts to multiple systems and microservices, the software also relies upon dynamic elements in configuration or deployment-optimization files. While legacy measurements looked at a system independently and heavily weighed vulnerability count and severity of vulnerability, the approach failed to account for the supply-chain implication of the modern technical solution space by including something beyond the severity concern of a compromise to this specific system. That vector beyond count and severity is the impact breadth, which addresses the concern with embedded libraries that are widely used[vi], hardware deficiencies that allow compromise (e.g., Spectre variants, CVE-2017-5753), or even compilers that have hidden vulnerabilities allowing for illicit code insertion or modification[vii].

For a true measure or comparison between systems or solutions, we would need to weight the count, severity, and impact breadth, and find the product of all three.

Threat Comparison

Threats across the supply chain, software lifecycle, and in the related processes abound in both FOSS and proprietary solutions, but over the last ten years the supply chain attacks have changed the risk calculus and bring consideration to different weights for existing and newer risks. Table 2, below, looks at threats where the implications differ between FOSS and proprietary software.

9. Forking and Versioning

In FOSS environments, forked codebases split reviewers and active participants across multiple projects. Fixes in one branch may not be adopted in base software or other branches and take time to migrate if they do. Fixes may not be appropriate for other branches or may introduce new vulnerabilities.

Support for older versions of software is the analogous behavior in the proprietary world. Budgets may prevent upgrades, and dependencies between versions may force organizations to stay on platforms or services that have known vulnerabilities.

10. Patching

Reliability engineering looks at mean time between failures, but security patching can look at mean time between vulnerability discovery and deployed patch. In this, FOSS seems to have a slight advantage over proprietary solutions[viii]. Not having a managed patching mechanism may be hurting FOSS though, perhaps more than speed to fix can make up for: the Log4Shell vulnerability (CVE-2021-44228), had a fix available within a day, but nearly a third of current downloads of that package are of an earlier, vulnerable, version[ix]. The true horror from research on downloaded packages is that 96% of the time that a software package is downloaded, a safer, more recent, version is available[x]. Updating build scripts, links that don't always point to HEAD, and time available for keeping dependencies fresh are all security considerations. Another concern with FOSS is that patching and mitigation actions are public and when the CVE and fix are published, threat actors can take this knowledge, along with sample exploit code and look for similar toolsets that may have the same code profile (even cut/paste identical code) to apply to create new zero-day vulnerabilities[xi]. Examples of this came after the Log4Shell vulnerability (CVE-2021-44228) and leveraging the privilege escalation vulnerability in Linux (CVE-2021-4034).

Proprietary solutions and hybridized solutions often have a maintenance contract related to updates and patching, the solution may or may not be managed, but the update process is frequently run by the solution provider. From an economic perspective software support is often based upon the expected revenue from newer versions of the software that can include patched updates from prior revisions. If the solution isn't widely adopted, if the version is a couple revisions behind the lead for that package, or if the company supporting it cannot invest in support, patching can take a back seat. With proprietary code, the users have no option to adopt the code and support it themselves and are beholden to the software provider.

11. Malign Code Insertion and Insider Threats

FOSS projects open contributor model is often paired with a review prior to adoption by maintainers[xii]. Those maintainers may not be experts in reviewing for security or privacy vulnerabilities allowing for insertion. Intentional contributions enabling nuanced vulnerabilities also occur, the field from which these are drawn include nation-state APTs, hactivism, organized crime and more. These vulnerabilities can be leveraged for data exfiltration, ransomware insertion, machine hijacking, fraud, or denial of service.

The closed list of contributors in corporate or proprietary solutions spaces is not proof against malign code insertion, there are examples of insider threat actors having actually engaged exfiltration or code insertion that include departing employees (e.g., Proofpoint[xiii]), through acquired companies' vulnerabilities (e.g., the Marriott-Starwood acquisition hack in 2014), accidental exposure of proprietary secrets by employees or contractors (e.g. Microsoft posts to GitHub in 2022[xiv]).

12. Continuous Security Review

The claim is made that "given enough eyeballs, all bugs are shallow"[xv] and that this fundamentally improves the quality and security of FOSS. Opposite questions also arise: "Sure the source code is available But is anyone reading it."[xvi] Questions about the reviewers include their number, qualifications, incentives/motivations, experience with the particular code base and cryptography, and their belief or trust in existing code.

Proprietary code differs in the reviews and motivations. Presumably, threat analysis is being performed (consciously or not) within corporations where vulnerabilities are being balanced against revenues or risk. For any system, commensurate processes and personnel would be assigned to write, review, test, and monitor. Those personnel face the same questions as FOSS, but the answers may differ for proprietary issues and for resource prioritization.

13. Software Use

In FOSS, the software is available for download, compilation, and use, often "as is" and without warranty. Adversaries have leveraged the trust in using open-source software during campaigns targeting senior developers, posing as recruiters and asking the developer to use modified versions of open source tools to demonstrate mastery[xvii]. As will be shown in section 17, Software Updates and Trust, checksums are not reliable proof against this type of attack.

Proprietary software is not proof against these types of threats, and with hybridization, the risks associated with composite software are present in a large majority of software packages[xviii]. When the included software is purchased as a binary library from other providers, particularly when that happens without an SBOM, the unknowns are effectively being granted full citizenship in the now proprietary software or internal process with little understanding of what may happen with the process, the data, or the distributed final binaries.

14. Developer Expertise

FOSS volunteers may not be security experts or have access to training or mentorship from those with that expertise. Private employers look for developer experience in open-source projects and encourage that activity[xix]. This drives even the experienced developers to contribute and review open-source projects to gain employment opportunities. With some FOSS implementations (see Table 1) explicitly citing authors, external credit is available and can lead to being hired to positions with higher compensation[xx], but with that credit comes accountability if vulnerabilities are found.

Proprietary projects face the same hurdle as FOSS when developers leave projects, but hand-offs, training, documentation, backup developers with depth on important projects can be retained or recruited within private companies. For security concerns, it may be easier to find compensated help than it will be to find time from those already employed willing to volunteer time to FOSS. When the cybersecurity job market sees 35% annual growth[xxi] with 3.5 million open cybersecurity positions[xxii] open globally, available time from qualified security developers is challenging even for those who can compensate for that time.

15. Privacy Engineering

Privacy engineering is a relatively new discipline[xxiii]. While developers in FOSS are as educated as developers in proprietary solutions, qualifications for privacy work include strong policy support, some legal support, and technical understanding of data architecture and understanding of movement of data through and across systems. While open-source efforts have attracted legal support through groups like the Law Center[xxiv], it may still be early to expect policy support.

Privacy engineering in the proprietary realm is influenced by a few factors, a few of these include corporate public perception, regulation and the associated enforcement efforts, as well as the financial benefits of the data economy that may work against the first two. Looking at liability and risk concerns as part of the motivation in incentives for support of technologies like privacy protections could advantage proprietary software over FOSS, but data economy tradeoffs play into this calculus as well.

16. Liability and Regulatory Risk

The EU Product Liability Directive (PLD) update work in 2023 appears to be struggling with FOSS[xxv]. In one clause it states: “With the aim of not hampering innovation: (i)free and open-source software developed or supplied outside the course of commercial activity, as well as (ii) the source code of software, should be excluded from the definition of products covered under the proposal (Recital13).”[xxvi] In a later clause of the full proposal[xxvii], it states “However where software is supplied in exchange for a price or personal data is used other than exclusively for improving the security, compatibility or interoperability of the software, and is therefore supplied in the course of a commercial activity, the Directive should apply.” The implication being that if FOSS is ever incorporated into a product, it is subject to liability risk, and with respect to the EU Cyber Resilience Act and the PLD, major FOSS tool providers wrote in opposition “it will have a chilling effect on open source software development as a global endeavor, with the net effect of undermining the EU’s own expressed goals for innovation, digital sovereignty, and future prosperity.”[xxviii].

Software liability for proprietary solutions certainly carries with it the legislative and regulatory risk of providing a service, but add to this the relatively recent additions to privacy regulation (GDPR in the EU and state-level privacy regulation in the US, PIPL in China, PIPEDA in Canada, along with several other sectoral or national privacy regulations in place now). Cybersecurity and Infrastructure Security Agency (CISA) in the US has subpoena powers related to cyber incidents and they are advocating for SBOM requirements, particularly for anything deemed critical infrastructure. Violations, delays, or inaccuracies in reporting may have consequential ramifications.

17. Software Updates and Trust

FOSS builds offer checksums on the binary distributions and allow users to compile these themselves as well. Checksums are not always present, but in over 88% of cases users may not notice checksums, understand checksums, know how to go about verifying checksums, or simply don't bother to verify packages. Over 33% of those asked specifically to verify software do so improperly and fail to catch a mismatch (partial pre-image attack)[xxix].

Signed binaries from PKI-backed digital certificates of trusted entities are intended to verify the source and the package contents match those of the trusted entity. While security policies, IDS/IPS, and software update mechanisms work to automate the verification of these packages, that doesn't mean supply chain issues can't occur with those credentials or the verification assumptions. The SolarWinds supply-chain attack inserted malicious code upstream of the signing[xxx], and the Aug 2023 Microsoft Azure attack using compromised Managed Service Account credentials to sign requests for new credentials which exposed user email and potentially virtual hosts[xxxi].

18. Linking Libraries and SBOM

FOSS linking is done publicly. There are tools to help organize these, and even some tools to help you find our who may be including other libraries in their builds. For a cybercriminal, knowing that a vulnerability exists in a particular software package, and perhaps having an exploit at the ready is more valuable when there is a clear list of other software that is now vulnerable because of the binary inclusion, however many levels.

Proprietary software still has the linked vulnerability problem, but the non-public nature of linking requires more trial-and-error work on the part of the cybercriminal looking for a way to apply an exploit. Software Bill of Material (SBOM) efforts of late[xxxii] are pushing for knowledge of all linked libraries by software owners. While that knowledge is valuable and can help with tracking and stopping vulnerabilities, the publication of this list of included libraries could be used as a roadmap for criminal adversaries.

19. Software Monoculture

In agriculture, plant propagation, breeding, and genetic similarities are used to help formulate herbicides and pesticides that can work without impact to target crops, it also helps in terms of standardization of equipment used for planting and harvesting and the efficiency of yield for a given quantity of land can be higher through monoculture farming[xxxiii]. The danger of these similarities, or identical genetic makeup in some cases, is that a threat to a single plant becomes a threat to all plants. One pest[xxxiv], one disease, one change in economics or distribution pricing can have a massive impact to the monoculture. This is entirely analogous to the work taking place in software.

Reliance upon existing public software is economically incentivized, and when underlying libraries are leveraged across multiple included platforms, services, or other software it drives further consolidation. In FOSS, the classic example of this is the reliance upon the OpenSSL library vulnerability (CVE-2014-0160) and the resultant Heartbleed impact which made the vulnerability in that package a concern across the ecosystem. Another more recent example has broader implications from the monoculture perspective, the Log4Shell vulnerability (CVE-2021-44228), which was pervasive with mitigation efforts that continue today.

Proprietary software is not immune to these considerations. When the operating system, browser, cloud offering, microservice, or tooling are all identical, a risk to one can become a global crisis. The other implication to using services outside of software run independently (e.g., an online email management service), is that an organization effectively makes each of the organizations running those services an insider to their data, their systems, and this opens the horizontal attack threat surface. When Microsoft Azure services were attacked in 2023[xxxv], Outlook 365 and Azure subscribers were all vulnerable.

Table 2 Threat Comparisons Where FOSS and Proprietary Software Differ

Threat	FOSS Implications	Proprietary Implications
Forking and Versioning	Update propagation across forked codebases, reviewers and participants split across projects, update applicability to forks, timeframe, new vulnerabilities.	Support for older versions of software: budgets may delay/prevent upgrades, dependencies between versions may force organizations on vulnerable platforms.
Patching	Mean time between vuln. ID and patch vs. 96% of patched software updates are available but ignored. Public patching and mitigation alerts adversaries.	Maintenance contracts, update process is frequently run by the solution provider, feature lag, security patch timeframes, code abandonment risk.
Malign Code Insertion & Insider Threats	Maintainer expertise in developing and reviewing for security, allowing for insertion of intentional malware contributions.	Active employee/contractor data or code exfiltration or code insertion. Departing employees, acquired companies, accidental exposure of proprietary secrets.
Continuous Security Review	Reviewer count, qualifications, incentives/motivations, experience with the project code, cryptography, and their belief or trust in existing code.	Liability pits resource prioritization against revenues and risk: development, review, testing, and monitoring; FOSS questions remain regarding experience and trust.
Software Use	FOSS is often trusted by technical users, it is offered "as is" and without warranty, and checksums are not proof of invulnerability.	Transitive corporate trust of aggregated software or packages (FOSS or proprietary) through business agreements and redistribution.
Developer Expertise	Existing security expertise, access to training or mentorship, pull toward commercial/paid development, contribution credit/accountability.	Developers leave projects, hand-offs, training, documentation, backup developers with depth can be retained or recruited, cybersecurity job market is tight.
Privacy Engineering	Technical understanding of data architecture and data movement across systems, strong policy and legal understanding and research tooling.	Balancing liability/risk of damaging public perception of corporate entities along with regulation and enforcement, against financial benefits of the data economy.
Liability and Regulatory Risk	EU Product Liability Directive update (2023) introduces significant legislative and regulatory risk to existing and future open-source projects and teams.	Global privacy legislation/regulation; security oversight (CISA/DHS Subpoena Powers), cyber incident reporting, SBOM, and critical infrastructure designations
Software Updates and Trust	Checksum presence, awareness, understanding, verification knowledge, consistency of practice, inability to catch partial pre-image attacks.	Signed binaries from PKI-backed certificates of trusted entities, IDS/IPS, and automation aren't proof vs. attacks upstream of signing or credential theft.
Linking Libraries and SBOM	Linking is done publicly, organization tools help cyber criminals map vulnerabilities to exploits to all software packages that include that vulnerability.	Linked vulnerability problem still exists but requires more trial-and-error work on the part of the cybercriminal. Publication of SBOM could remove this advantage.
Software Monoculture	Economic incentives for consolidation lead to wide-striking vulnerabilities such as Log4Shell and Heartbleed/OpenSSL; diversifying increases the attack surface.	Identical operating systems, browsers, and tooling have broad risk. Hyperscaling and microservices are business insiders that can yield wide and immediate compromise.

20. Publication of CVEs

When the fixes to published CVEs are made in proprietary code, the mechanism by which the solution has been secured is not readily available to all observers, however in FOSS, the mitigation is published and can lead to providing a roadmap for other potential vulnerability exploits.

The implications of CVE publication, notification periods, and bug bounty programs are beyond the scope of this paper, but the public response, the measurable time between vulnerability listing and fix, are all information that the adversaries can use to understand which projects may be faster than others to update, and when fixes may be more challenging to adopt (e.g., such as when an API changes or parameter structure is modified). The proprietary response may make the fix visible to adversaries who are leveraging or testing that vulnerability, but for those not observing as actively, it may be difficult to discern which companies are faster than others at closing vulnerabilities.

21. Hybridization and Proprietization of Open Source

With the economic benefits of code reuse and libraries, FOSS currently exists in between 80-90% of all modern application code[xxxvi]. With aggregation, forking, and rates like this, can claims for a distinction between the two hold up? Proprietization may not be a word (yet), but the hybrid nature of modern software, the long supply chains, and the somewhat nebulous practices of different open-contributor projects mean significant threat analysis by cybersecurity professionals.

22. Economic Factors and Software Library Friction

Development costs involve not just the software being custom written for the solution, development costs now include the analysis, retrieval, documentation, testing and integration of other software to integrate into the final solution. If those are FOSS, the time spent to integrate those is still part of this equation. Adding to this are the maintenance costs, including staying up to date on patches, API changes, updates to tests, internal data management, deployment architecture, regulatory compliance checks (e.g., privacy regulations on data movement), security review and decisions on how to handle software at its end-of-life. For each new feature addition to a final software solution, the maintenance and development costs, referred to above, must be undertaken along with ensuring that the update process is managed, ensuring that new vulnerabilities aren't introduced, and population of release notes are made. For providers of library solutions or packages that get used by other software or included in other software, the list of activities (and related costs) increases: any other managed software must also be tested, version differences and mandatory upgrades must be considered, API updates, documentation changes, breaking changes, migration paths, support for prior versions, compatibility, interoperability and deployability must be part of the decision and release process. Do SBOMs need to be updated, are known vulnerabilities closed, are license requirements met, do contributions need to be submitted back to source projects, and are there any new regulatory requirements around reporting that need to be met.
All of this introduces friction into the process, in a proprietary solution necessary steps may be mandated by internal processes, checklists, and requirements from groups or clients that have their own list of requirements to check. In the FOSS environment, volunteers would need to do this job, or limited funds must be used to make sure to stay close to the verification and validation processes described above. As these may be considered overhead by some developers, it's possible that some of these steps could be overlooked.

23. Policy Implications

As governments and policymakers look to regulations to help get a better hold on supply-chain insecurities, several actions are taking place. Governments limiting sources of origin for some technical solutions such as the FCC's actions to limit access to the U.S. market[xxxvii] but foreign actors, malicious actors, and organized crime can all make contributions to most open-source projects so that part of the chain is potentially still vulnerable. The European Union is exploring product liability extensions which could have an impact to FOSS, and we noted earlier in this paper the response to those new potential regulations. The US Government also recognizes the risk in this space as they actively seek feedback on open-source security for purposes of focus and prioritization through their 2023 RFI[xxxviii]. Whether government pressure changes the culture or development models of FOSS remains to be seen, but discussion has turned to considerations like SBOM, component risk, mandated mean time between disclosure and patching, product support lifetime expectations, minimum expectations for security and privacy, and, like what we see in the EU, the reconsideration of product liability in relation to modern concepts, tooling, and services provided by computational infrastructure. Government policy will play a part in the nuances between FOSS software security and that of proprietary or commercial endeavors; for those endeavors that utilize, aggregate, or include FOSS as part of their deliverables, the implications could be dramatic, and this could also have an impact on FOSS contribution.

24. Mitigations

This paper has addressed several threats that have a difference in implications, ramifications or nuance between FOSS and proprietary solutions. This can be overwhelming to see and can feel futile to fight, but there are some improving solutions and tools to help address supply-chain security that are available today and coming soon. Some considerations follow:

- Everyone should follow best practices for DevSecOps[xxxix] including securing the entire software lifecycle (development process and production environment), manage authentication and authorization (use least-privilege principles, encrypt sensitive data, role-based access control, two-factor authentication, access control lists), monitor (logging and monitoring, penetration tests, intrusion detection and prevention systems (IDS/IPS), audit regularly), train anyone engaged or authorized for any part of the process, use secrets management tools like Hardware Security Modules (HSMs), and have a disaster recovery plan (including regular table-top exercises and drills).
- For providers of open source software and tooling to handle versioning (e.g., GitHub, SourceForge, etc.), institute a practice of notification of applicable CVEs, generate and maintain the SBOM for each software package, complicate or prevent downloading of binaries or source versions that have been patched in subsequent versions, remove exemplar code or tools that exist for exclusively malicious purposes, and help to track, attribute, and verify contributors to FOSS.
- For those providing builds or using builds, consider the Open-Source Security Foundation SLSA[xl] version 1.0 levels from 2023(see Table 3, below).
- For those in industries identified by CISA as being part of the Critical Infrastructure (Chemical Sector, Commercial Facilities Sector, Communications Sector, Critical Manufacturing Sector, Dams Sector, Defense Industrial Base Sector, Emergency Services Sector, Energy Sector, Financial Services Sector, Food and Agriculture Sector, Government Facilities Sector, Healthcare and Public Health Sector, Information Technology Sector, Nuclear Reactors, Materials and Waste Sector, Transportation Systems Sector, and the Water and Wastewater Systems), there are several resources for improving supply chain security[xli] including this report from the FCC and CSRIC VIII[xlii] which list the following suggestions for addressing open source security:

- Strict internal requirements should exist to protect the company and its customers.
- Third-party suppliers should be held to the same company standards.
- Policies should apply equally to open-source software as with proprietary software.
- Third-party software should be sourced by a centralized configuration management team.
- Centralized configuration management teams should ensure sources are reputable.
- A gating subprocess should validate that patches are applied, and scans are completed.
- A post-scan analysis should be used to reveal issue severity, priority, and applicability.

- For small and medium-sized businesses who are engaging in software in any manner are addressed by CISA who have highlighted the top six risks where those businesses should focus their attention and growth: Cyber Expertise, Executive Commitment to Cybersecurity, Supply Chain Risk Management, Single Source Suppliers, Supplier Disruption and Visibility into Supplier Cybersecurity Practices[xliii].

Table 3 Open-Source Security Foundation SLSA

SLSA Level	Description	Requirements	Example
0	No guarantees.	SLSA 0 represents the lack of any SLSA level.	Unknown provenance
1	Documentation of the build process	**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.	Unsigned provenance
2	Tamper resistance of the build service	**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.	Hosted source/build, signed provenance
3	Extra resistance to specific threats	**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.	Security controls on host, non-falsifiable provenance
4	Highest levels of confidence and trust	**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.	Two-party review + hermetic builds

The above are not the complete list of options available for mitigation, but these steps are intended to help organizations, FOSS tool providers, and software enterprises contribute to a safer software supply chain.

Outlook

If one assigns a point to each threat and count wins for FOSS versus proprietary solutions, one would be doing it wrong. Each piece of software is going to have a unique history, contribution model, algorithmic approach, versioning, lifecycle management, as well as unique teams to develop, support, test, configure and deploy the solution. For each one of these, a considered approach of the best model, open or proprietary, belongs in the architectural decision process. Are specialists necessary, are they available in the open-source community, are they qualified and incented to work on the project? These just scratch the surface of the questions that will need to be answered. It's hard to ask them all, which is why looking at varying threats and how they have evolved since the research literature changed is worth our effort.

Abbreviations

API	Application Programming Interface
CISA	Cybersecurity and Infrastructure Security Agency (part of Department of Homeland Security)
CVSS	NIST Common Vulnerability Scoring System
DHS	Department of Homeland Security
FOSS	Free or Open Source Software
HSM	Hardware Security Modules
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
PLD	European Union Product Liability Directive
SBOM	Software Bill of Materials
SLSA	Supply-chain Levels for Software Artifacts

References

[i] https://wiki.openssl.org/index.php/Contributions
[ii] Schryen, Guido & Kadura, Rouven. (2009). Open source vs. closed source software: Towards measuring security. Proceedings of the ACM Symposium on Applied Computing. 2016-2023. 10.1145/1529282.1529731.
[iii] Schryen, Guido, "Security of Open Source and Closed Source Software: An Empirical Comparison of Published Vulnerabilities" (2009). *AMCIS 2009 Proceedings*. 387.
https://aisel.aisnet.org/amcis2009/387
[iv] Schryen, Guido, "Security of Open Source and Closed Source Software: An Empirical Comparison of Published Vulnerabilities" (2009). *AMCIS 2009 Proceedings*. 387.
https://aisel.aisnet.org/amcis2009/387
[v] CrowdStrike. (2023) CrowdStrike 2023 Global Threat Report., https://go.crowdstrike.com/rs/281-OBQ-266/images/CrowdStrike2023GlobalThreatReport.pdf
[vi] Ghafoor, Imran & Jattala, Imran & Durrani, Shakeel & Tahir, Ch. (2014). Analysis of OpenSSL Heartbleed vulnerability for embedded systems. 314-319. 10.1109/INMIC.2014.7097358.
[vii] Thompson, K. Reflections on Trusting Trust. *Comm. Of the ACM. 27*, 8 (1984), 761-763

[viii] Schryen, Guido, "Security of Open Source and Closed Source Software: An Empirical Comparison of Published Vulnerabilities" (2009). *AMCIS 2009 Proceedings*. 387.
[ix] Fox, B., *The EU's Product Liability Directive could kill open source,* Tech Radar, 10 July, 2023, https://www.techradar.com/pro/the-eus-product-liability-directive-could-kill-open-source
[x] ibid
[xi] CrowdStrike. (2023) CrowdStrike 2023 Global Threat Report., https://go.crowdstrike.com/rs/281-OBQ-266/images/CrowdStrike2023GlobalThreatReport.pdf
[xii] https://wiki.openssl.org/index.php/Contributions
[xiii] Proofpoint, Inc. v. Samuel Boone, Case# 1:2021cv00667, July 29, 2021, US District Court for the Western District of Texas, Judge Lee Yeakel
[xiv] https://www.vice.com/en/article/m7gb43/microsoft-employees-exposed-login-credentials-azure-github
[xv] Raymond, E. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary troff edition* (1999) https://lists.gnu.org/archive/html/groff/2021-11/pdfRt8tRop5yy.pdf
[xvi] Levy, E. *Wide open source*, http://www.securityfocus.com/news/19, 2000.
[xvii] Goodin, D. *Numerous orgs hacked after installing weaponized open source apps*, Ars Technica https://arstechnica.com/information-technology/2022/09/north-korean-threat-actors-are-weaponizing-all-kinds-of-open-source-apps/ , 29 Sept 2022
[xviii] Fox, B., *The EU's Product Liability Directive could kill open source,* Tech Radar, 10 July, 2023, https://www.techradar.com/pro/the-eus-product-liability-directive-could-kill-open-source
[xix] https://blogs.vmware.com/opensource/2020/08/25/boost-your-career-through-open-source-contribution/
[xx] Hann, Il-Horn; Roberts, Jeff; Slaughter, Sandra; and Fielding, Roy, "Economic Incentives for Participating in Open Source Software Projects" (2002). ICIS 2002 Proceedings. Paper 33.
[xxi] US Bureau of Labor Statistics: https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm 8 Sep, 2022
[xxii] Cybercrime Magazine: https://cybersecurityventures.com/jobs/ 14 Apr, 2023.
[xxiii] Fennessy, C. International Association of Privacy Professionals: *Privacy engineering: The what, why and how* https://iapp.org/news/a/privacy-engineering-the-what-why-and-how/ 8 Aug, 2019
[xxiv] https://www.zdnet.com/article/lawyers-ride-shotgun-for-open-source/
[xxv] De Luca, S., European Parliament, "Briefing: EU Legislation in Progress: New Product Liability Directive" https://www.europarl.europa.eu/RegData/etudes/BRIE/2023/739341/EPRS_BRI(2023)739341_EN.pdf May, 2023
[xxvi] Ibid, pp 5.
[xxvii] Document 52022PC0495, Proposal for a DIRECTIVE OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on liability for defective products, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52022PC0495, {SEC(2022) 343 final} - {SWD(2022) 315 final} - {SWD(2022) 316 final} - {SWD(2022) 317 final}, 28 Sep 2022
[xxviii] Harris, J., *Open Letter to the European Commission on the Cyber Resilience Act,* https://newsroom.eclipse.org/news/announcements/open-letter-european-commission-cyber-resilience-act , 17 April 2023
[xxix] Alexandre Meylan, Mauro Cherubini, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2020. A Study on the Use of Checksums for Integrity Verification of Web Downloads. *ACM Trans. Priv. Secur.* 24, 1, Article 4 (September 2020), 36 pages. **https://doi.org/10.1145/3410154**
[xxx] Constantin, L., *SolarWinds attack explained: And why it was so hard to detect*, CSO, Dec 15, 2020, https://www.csoonline.com/article/570191/solarwinds-supply-chain-attack-explained-why-organizations-were-not-prepared.html
[xxxi] Schneier, B., *Microsoft Signing Key Stolen by Chinese*, Aug 2023, https://www.schneier.com/blog/archives/2023/08/microsoft-signing-key-stolen-by-chinese.html
[xxxii] https://www.cisa.gov/sbom
[xxxiii] McGuire, A., *Ecological Theories, Meta-Analysis, and the Benefits of Monocultures*. Washington State University
[xxxiv] JM Sirota, E Grafius, W Boylan-Pett, B Ferrari, P Kolarik, A Pyle, B Scriber, Colorado Potato Beetle Control... Insecticide and Acaricide Tests 18 (1), 153-155, https://academic.oup.com/amt/article-abstract/18/1/153/4573455, 1991
[xxxv] Schneier, B., *Microsoft Signing Key Stolen by Chinese*, Aug 2023, https://www.schneier.com/blog/archives/2023/08/microsoft-signing-key-stolen-by-chinese.html

[xxxvi] Fox, B., *The EU's Product Liability Directive could kill open source,* Tech Radar, 10 July, 2023, https://www.techradar.com/pro/the-eus-product-liability-directive-could-kill-open-source
[xxxvii] Federal Communications Commission FCC 22-84 *Protecting Against National Security Threats to the Communications Supply Chain through the Equipment Authorization Program* https://docs.fcc.gov/public/attachments/FCC-22-84A1.pdf 25 Nov 2022
[xxxviii] Office of the National Cyber Director, *Request for Information on Open-Source Software Security: Areas of Long-Term Focus and Prioritization*, https://www.federalregister.gov/documents/2023/08/10/2023-17239/request-for-information-on-open-source-software-security-areas-of-long-term-focus-and-prioritization , 10 Aug 2023
[xxxix] Baig, A. DevOps.com 15 DevSecOps Best Practices, https://devops.com/15-devsecops-best-practices/ 15 Apr 2022
[xl] SLSA specification version 1.0, Open Source Security Foundation, https://slsa.dev/spec/v1.0/ 2023
[xli] CISA *Resources and Tools* https://www.cisa.gov/resources-tools/all-resources-tools
[xlii] FCC, *CSRIC Report on Recommended Best Practices to Improve Supply Chain Security*, https://www.cisa.gov/resources-tools/resources/csric-report-recommended-best-practices-improve-supply-chain-security 21 Sep 2022
[xliii] CISA, Reducing ICT Supply Chain Risk in Samll and Medium-Sized Businesses https://www.cisa.gov/sites/default/files/2023-05/fs_reducing-ict-supply-chain-risk-smb_fact-sheet_508.pdf May 2023