REPORT TO THE CISA DIRECTOR

Technical Advisory Council

Open Source Security

October 11, 2024

Introduction:

The Cybersecurity and Infrastructure Security Agency (CISA) Cybersecurity Advisory Committee (CSAC) established a Technical Advisory Council (TAC) subcommittee (hereinafter referred to as the "subcommittee") to help CISA—in line with its Secure by Design goals—can encourage companies to be better stewards of the open source software they depend on and produce.

Continued high impact and high profile security incidents related to software vulnerabilities in commercial and Open Source Software (OSS) has drawn the attention of news media and governments worldwide. Increasingly complex software supply chains and product dependencies have become frequent targets for organized crime and nation states, seen as legitimate targets with a low barrier to entry and high reward. As great nation competition escalates, we expect the targeting of the open source software ecosystems to increase. In this environment, the status quo of willful ignorance of security in software dependencies will not be sufficient to provide the resilience, assurance, or stability necessary to protect against these threat actors and allow the public to reap the benefits of OSS.

CISA began their Secure By Design initiative in 2023 [1] to raise awareness, provide guidance, and build voluntary standards. It shifts the terminology to talking about software defects as predictable outcomes of not following Secure By Design principles. The subcommittee believes this is a welcome departure from accepting software vulnerabilities as an inevitable outcome of the modern development lifecycle.

Although the path to improved security for commercial software is fairly straightforward because it responds to all the "normal" market and regulatory incentives, such as market share and liability, the OSS environment largely does not. OSS is in an environment driven by community projects, individual contributors, and commercial developers often supporting just the OSS components on which their products rely. Open source software intentionally comes as is, without warranties or conditions of any kind. Underfunded OSS non-profit foundations can struggle in paying for development, and "Freemium" business models have not been sufficient to meet the need.

For example, the popular web server Nginx, owned by F5, is sold as an enterprise product and also provided in a free "community" version. Many innovations come from the community but then get incorporated and supported only in the enterprise version. The free/enterprise tiering structure is a common way to grow market share and product familiarity, and Nginx has about 34% of the web servers observed online. [2] However, if security improvements only are available in a premium or enterprise version, this can create incentives to 'save' on costs with a less secure version, to the detriment of the overall security of the ecosystem.

The questions asked by CISA, below, are the basis for our discussion and recommendations around OSS.

- How should CISA encourage the adoption of safe consumption norms for open source software, while also encouraging companies to contribute fixes and enhancements back to the open source projects?
- How should CISA work to encourage these norms and contributions from federal agencies?
- How can CISA shift the burden of securing open source software to rest on those companies who routinely build commercial products by offering modified versions of the open source project for a fee, often withholding capabilities from the free version to incentivize consumers' purchase of the enhanced version?
- Are there additional recommendations for CISA to (a) support secure by design outcomes in AI systems that are distributed under open source compatible terms, or (b) protect both the public and private sector from potential harms from misuse of foundation AI models with widely available model weights?

Background on Open Source Software

Over the last several decades, open source software has continued to increase in importance, and is now woven into most aspects of society, creating an estimated $8 trillion dollars in aggregate value [3]. OSS plays fundamental roles not only in computing and online services (e.g. Linux and most Internet infrastructure), but also critical roles in telecommunications, science, Industrial Control Systems (ICS), Operating Technologies (OT), and government at all levels. According to the State of the Software Supply Chain report, about 80-90% of modern applications are of open source origin, or include open source components [4].

Accordingly, "avoiding" the use of OSS is not a realistic option — we are well beyond that point, and in fact wide-spread use of OSS is a global competitive advantage for productivity and innovation. In response to the increased dependence on OSS in critical functions, the Federal government created the Office of the National Cyber Director (ONCD) within the White House, which now has a national strategy for OSS [5], as well as DHS' increased focus on OSS [6], both through CISA [7] and the Cyber Safety Review Board. [8]

Wikipedia provides a good overview of OSS, starting with a broad definition: "Free and open-source software (FOSS) is software that is available under a license that grants the right to use, modify, and distribute the software, modified or not, to everyone free of charge. The public availability of the source code is, therefore, a necessary but not sufficient condition. FOSS is an inclusive umbrella term for free software and open-source software.[a] FOSS is in contrast to proprietary software, where the software is under restrictive copyright or licensing and the source code is hidden from the users."[a]

Within this broad definition, there is an even more complicated space, including myriad software licensing regimes such as Apache, BSD, GNU General Public License (GPL), or MIT, some of which can be incompatible with others. Moreover, some widely used OSS projects have changed their licenses, sometimes to different open source licenses, and sometimes to go closed source in order to create commercial offerings.[b] With increasing frequency, open source software vendors with commercial backing are relicensing their software with a "source available" license model that restricts

[a] In this document we use "Open Source Software," to be inclusive of Free and Open Sources as described. As Wikipedia explains, "Although there is an almost complete overlap between free-software licenses and open-source-software licenses, there is a strong philosophical disagreement between the advocates of these two positions. The terminology of FOSS was created to be neutral on these philosophical disagreements between the Free Software Foundation (FSF) and Open Source Initiative (OSI) and have a single unified term that could refer to both concepts." [9]

[b] For a list of open source licenses, and their license's features, see Wikipedia[16]

third-party use. These vendors usually require a Contributor License Agreement (CLA) that permits relicensing of contributed code. The Open Source Initiative tracks over 115 different types of licenses. [10]

The corollary to Open Source *Software* is Open Source *Hardware* (OSHW)[c], which raises many of the same concerns and issues. For the purposes of this report, OSHW comes under the same umbrella of issues with which OSS must contend.

The US Government, NGOs, and commercial providers have been identifying gaps in the OSS ecosystem for as long as it has existed. The difference now is the consequences of software defects and vulnerabilities are much greater than in the past and supply chain attacks are growing in frequency and scope. Shared software, tooling, supply chains, and manufacturers means a defect in one piece of code can have wide and unforeseen impacts. Software development builds complex systems by interlocking thousands of small, rigid pieces together, such that the failure of any one piece can propagate cracks throughout the entire system.

Scope

The complexity of this problem space can be incredibly alluring to pontificate. There are many expert opinions delving into the nuances of software licensing, incentive structures, and liability. The subcommittee recognizes that many aspects of OSS security have been considered before, and there are excellent documents exploring all the various pathways. Accordingly, this document will focus on areas the subcommittee feels are underrepresented and ripe for improvement that would benefit those participating in the OSS space. This report only briefly references some of the pre-existing work, which describe problem areas, proposes maturity models, and recommends best practices for implementers, focusing instead on the underrepresented areas.

The practice of adopting and incorporating OSS safely is not a new concept, and there exists prior work that explores this topic from different perspectives. A small subset includes the following: "*Securing the Software Supply Chain: Recommended Practices for Managing Open-Source Software and Software Bill of Materials*" [11], The Secure Supply Chain Consumption Framework (S2C2F) [12], *Principles for Package Repository Security*, The Securing Software Repositories Working Group (WG) of the OpenSSF [13], *Surviving Software Dependencies* [14].

Outside the US, the European Union has notably created the Cyber Resilience Act (CRA) [15], which was formally approved by the European Parliament in March 2024, and is designed to increase software security with cybersecurity regulations and potential liability. The CRA's initial draft generated criticism from the open source community, which can help inform the committee of potential policy pitfalls.[20]

Core Problems to Solve

Most of the work to date on "safe" open source consumption has centered around two core questions that try to understand the risks. First, end users and software developers are trying to understand the dependencies of their

[c] Open-source hardware (OSHW) consists of physical artifacts of technology designed and offered by the open-design movement [17]. Both free and open-source software (FOSS) and open-source hardware are created by this open-source culture movement[18] and apply a like concept to a variety of components. It is sometimes, thus, referred to as FOSH (free and open-source hardware). The term usually means that information about the hardware is easily discerned so that others can make it – coupling it closely to the maker movement [19].

software choices: What *additional* OSS is required for this OSS I want to consume to function? Second, what are the versions of the various OSS components that will be required?

In addition, there is a foundational question of how to bridge the cultural gap between the legal contract-driven norms of commercial consumers and the social-contract driven cultural norms of OSS developers. This must be addressed in the process of balancing recommendations to solving the first two core questions.

The OSS ecosystem is composed of diverse developers, some paid, but more often not paid. Companies often rely on the enforceability of legal contracts to define and enforce norms. However, the OSS ecosystem often operates on community-driven values to inspire the development such as reputation, kindness, or personal development. This makes it sometimes a challenge to get these volunteer developers to respond with the urgency required by certain safety-critical norms of commercial entities. Simply put, it can be hard, not to mention unreasonable, for a commercial entity to convince an unpaid volunteer to drop all their other priorities and fix an issue, even an urgent security issue, on the time scale the commercial entity would expect of itself.

The Need for an Accountable Intermediary

The fundamental challenge in this space is that OSS in general is clearly labeled as "as is" software with no promises of fitness and no liability. This puts the accountability for security squarely on the *consumer* of OSS, and not its producers or maintainers. Conversely, governments and others set high-level requirements that must be met, putting the accountability squarely on the *producers*.

These are both valid world views for how software should work, but they are obviously incompatible on the surface. The solution in general is to have an accountable intermediary that bridges this gap, sometimes called a *curator*. When Red Hat sells Linux with various support promises they are playing this role: Red Hat is the accountable producer in the eyes of the consumer even though they receive OSS from the community as is with no promises. They do the work to ensure compliance is met despite whatever issues might exist in the raw Linux upstream.

It is important to name this as a "role" because many parties can play this role. An OSS community can choose to play versions of this role themselves, for example FreeBSD recently decided to help its consumers meet Federal Secure Software Development Framework (SSDF) requirements [21]. When companies centralize management of open-source packages, the central team is typically playing this role for internal consumers, including managing security patches. One option for the Federal government would be for one agency to play this role for not only other agencies, but for State, Local, Tribal, and Territorial (SLTT) governments as well.

When consumers use OSS directly, they implicitly assume this role, often without realizing it. Doing this role well can be a significant burden and cost, but for critical infrastructure we collectively need to make sure it gets done.

The ability to *pay* a group to play this role is nascent, but this kind of outsourcing is critical to broad improvements to security. Such curators are better equipped than most companies to fulfill the role and they get significant benefits from specialization and scale. Improved tooling and the use of AI will further increase the relative benefits of paid curators. Because the role of the accountable intermediary is fundamental, it will return throughout the rest of this report.

There are additional problems when trying to make an informed OSS consumption decision. First, there is no standardized method for an OSS consumer to identify if a project has performed any software assurance work. There is

also no standard method for an OSS project to share the level of security assurance work they have performed. This includes the different threat models that have been considered, the fuzzing tests performed, code coverage, static analysis rules, dynamic analysis results, software compiler flags for binaries, and many other features covered by the National Institute of Standards and Technology (NIST) standard on the Secure Software Development Lifecycle (SSDF) [22]. There are many kinds of metadata that describe the security environment in which software, or a project exists, the issue is there are few standard ways of expressing or collecting them.

Second, there is no standardized method to describe operational supply-chain security. When consuming binaries and source code from an OSS project the consumer inherits transitively any vulnerabilities, or attacks against, their supply chain. Currently, there is no reliable way of assessing a project's operational security when making consumption decisions. An open source developer might manage a project with outdated security patches without consumers knowing. A project may have the latest security and risk models in place, but there is no standardized way to communicate the exact security measures taken by a project to its consumers.

Finally, there is no common method to understand software "certifiability". For example, US Government customers often require Federal Information Processing Standards (FIPS) or other certification methods for open source libraries related to cryptography. There is no standard or simple way to describe what work has been done on a given piece of software that supports a given certification. This leads to a lot of duplicated work, where each vendor re-validates the same code to be compliant. A more efficient method would be to have a mechanism to publish what code has met a given requirement, and what still needs to be worked on. This would allow more of a community response instead of individual contributors duplicating work.

Solving any of these problems would be valuable in improving software security, but taken together each reinforces the other, which would lead to a generational improvement. In all three cases, the use of a curator can speed up adoption, as the intermediaries can act as both a mechanism of standardization and a reduction of duplicate work.

Software Bill of Materials (SBOM)

Currently one method of addressing these challenges is by analyzing a Software Bill Of Materials (SBOM) for the OSS you want to consume. The SBOM is an evolving standard that provides improved transparency over "what" is inside a given piece of software.

SBOM helps end-users with strict security and compliance requirements, such as financial institutions and regulated enterprises, make better informed choices about what software to consume. The SBOM should provide a detailed list of all the components, dependencies, and libraries included in a piece of software, and helps the end-user track and manage the software's security history. If the SBOM reveals old versions of software or those with known security vulnerabilities, end users might choose a less risky version, if available.

The Strengths and Challenges of the SBOM

An SBOM is currently most useful when end users need to rapidly identify what software sub-components, including their particular versions, are deployed in their infrastructure. The utility was shown by the response to the log4shell vulnerability. Organizations that had SBOMs could rapidly determine what software used log4j and see if it was a vulnerable version. Then these organizations could save time and effort by focusing their remediation efforts on the

identified systems. Those without an SBOM for their deployed systems had to perform a time-consuming audit of all their infrastructure, looking for log4j and identifying its version one by one.

Beyond the strong utility helping to remediate large-scale vulnerabilities, SBOMs are also quite useful for preventative measures. For example, a software development team could use the SBOM to look for possible problems associated with an OSS component *before* they start consuming it. The SBOM reveals the dependencies of the OSS component, as well as identify the subcomponents included, along with version numbers and possibly other data. This information helps the development team understand whether the parts involved are current or out of date, or perhaps includes previously discovered defects, showing that the maintainer is not keeping current. This provides the development team with an understanding of the additional reliability and security risks they will be inheriting should they select this OSS component. A good example is the public tool, provided by Open Source Insights, which shows the known vulnerabilities for a wide range of OSS packages and includes (transitively) the vulnerabilities inherited from dependencies [23].

Despite this high potential, SBOMs in practice have significant limitations. There are multiple "standards" that are largely not interoperable (e.g. SPDX and CycloneDX), which is particularly problematic for composition. Ideally, the SBOM of a package would be easily computed from the SBOMs of its components, even if they used different formats.

The tooling remains in early stages, in part due to complex and conflicting standards, leaving many users to generate SBOMs primarily for compliance, without receiving all of the actual benefits.

It is also common today for an SBOM to miss some components, especially if it is computed from a deployable artifact (like a binary), rather than generated during the process of its construction. Ideally, SBOMs should be generated as part of a trusted build process, after which they can also be signed for authentication. However, build systems are complex and often customized, making it difficult to add this capability broadly. Similarly, although SBOMs could be used during development to make better consumption choices, that process is rare today, as it is not yet integrated into the development process.

Finally, intermediate repositories are needed, such as package managers and Docker container repositories, to support SBOMs for their artifacts. This is nascent today. Without such support consumers must compute SBOMs after the fact on their own.

SBOMs have a goal of helping consumers of software understand the security of dependencies. However, SBOMs currently only tell a consumer what dependencies have been consumed. They do not address whether those dependencies are secure. If consumers want to understand the supply chain security or Software Development Lifecycle (SDL) applied to a dependency, SBOMs will not provide that. Consumers must care not only about the contents, but also the processes used to *create* packages or other artifacts.

The Supply-chain Levels for Software Artifacts (SLSA) is a security framework, which provides a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure [24]. The SLSA framework establishes *process* goals that should be followed to address supply-chain security risks. These goals include things such as using a trusted build process and producing signatures and secure hashes for produced artifacts for authentication and tamper prevention. SLSA and SBOMs are complementary [25]. In fact, it is hard to trust an SBOM unless these SLSA process goals are also met.

Overall, CISA should push for adoption of both SBOMs and SLSA and establish recommended formats for both kinds of attestations that enable interoperability and especially *composition*. Ideally, artifacts like packages should not only come with authenticated SBOM and SLSA information, but such metadata should be delivered in a way that can be combined with that of other ingredients to build the correct SBOM and attestations at the next level up. This is not an easy goal, but it is fundamental to safe consumption given the deeply nested nature of consumption.

Curators, the accountable intermediaries discussed above, can play an important role in driving towards effective composition by ensuring that their packages produce high-quality metadata, including interoperable SBOMs and signed SLSA provenance information. They will need to solve these problems for the packages they import and thus can help drive towards better consistency across OSS packages.

Longer term, package metadata can provide transparency for many different kinds of security practices, such as what kind of security testing has been done, use of multi-factor authentication, and other indicators of security best practice maturity.

CISA Questions

Question 1: *How should CISA encourage the adoption of safe consumption norms for open source software, while also encouraging companies to contribute fixes and enhancements back to the open source projects? How should CISA work to encourage these norms and contributions from federal agencies?*

Safe Consumption Norms

Safe Consumption of OSS is part of the totality of the software and project lifecycle. Safe Consumption should consider safe use, the user / developer feedback lifecycle to the OSS maintainers, notification to users of important issues, and ultimately lay the groundwork for the safe transfer or shutdown of a project.

The practice of adopting and incorporating OSS safely is not a new concept, and there exists prior work that explores this topic from different perspectives. A small subset includes the following: *"Securing the Software Supply Chain: Recommended Practices for Managing Open-Source Software and Software Bill of Materials"* [11], The Secure Supply Chain Consumption Framework (S2C2F) [12], "Principles for Package Repository Security" - The Securing Software Repositories Working Group (WG) of the OpenSSF [13] and their SLSA specification [24]. Outside the US, the European Union has drafted the Cyber Resilience Act (CRA). [15]

The EU CRA endeavors to address the issues legislatively, by imposing liability and other legal requirements. In 2027, vendors that do not comply with the CRA won't be able to sell products in the EU, so US companies that intend to sell in the EU will need to adapt.

When considering safe consumption, it can be helpful to look at how OSS projects maintain their source code, and how they send and receive updates upstream and downstream to consumers of their OSS. "Downstream" refers to software that is created by the original authors or maintainers, and distributed as source code flowing "down", often to other developers or consumers for potential inclusion in their software. "Upstream" refers to code sent by these users back upstream to the original development team for inclusion in the original project. [26][27] Together they are part of the key innovation of OSS, a circular life cycle, which often gets improved or customized by the users.

Downstream: Consuming OSS

When consuming OSS software, staying up to date is important. As NIST noted in the Secure Software Development Framework, developed using downstreamed software should "[o]btain provenance information (e.g., SBOM, source composition analysis, binary software composition analysis) for each software component," and "[i]mplement processes to update deployed software components to newer versions."[21]

The primary value of staying up to date is that it enables a fast response when a vulnerability is discovered. For critical systems in particular, they must be prepared in advance to apply security patches quickly. The farther out of date a system is the harder it becomes to patch. Typically, the security patch only applies to the current version, so the consumer must either get up to date to apply it or rewrite it to apply to their out-of-date system. The more out of date the system is, the worse either process will be.

Staying up to date is particularly important for online systems. However, many mostly offline systems, including air-gapped systems and embedded systems, such as automobiles, limit updates in favor of reliability and well-understood behavior. Such systems must instead put more effort in up front to limit and/or harden their dependencies.

Today, most organizations leave consumption issues to their developers, which naturally leads to inconsistent practices. The difficulty of doing consumption well leads to both more centralization and more processes, at least in larger organizations. The process of identifying when updates are required differs based on the method of consumption. There are three primary methods today for consuming open source:

1) Decentralized and in control of developers (The YOLO[d] method)
   Developers within large software companies self-manage what, where, when OSS is consumed. This distributes the onus of updating OSS software onto these developers and can carry clear risks for the organization.

2) Decentralized with scanners
   Software companies allow developers to manage dependencies but leverage a scanner to determine if those dependencies are safe. An improvement to the YOLO method but leaves a number of gaps: OSS dependencies are not always clearly observable from a scanner's vantage point, and scanners are imperfect detectors, so outdated or vulnerable dependencies may continue to go undetected.

3) Centralized safe consumption
   Large software companies create "known good" software repositories where projects that are in demand are maintained centrally. This includes managing supply chain, infrastructure, and security updates. It represents a significant resource cost but is more efficient than having each team do it individually. However, centralized systems also provide a single point of failure.

Centralized safe consumption is a form of curation. The curators are accountable for various security properties of their managed code, and they apply patches as needed to reduce risk or increase fitness. Consumption from a (paid) curator can greatly increase the actual level of safe consumption. This is a nascent area that we discuss more under Question 3.

Another important focus for downstream consumers is the choice of *which* packages to consume. Many packages, despite widespread use, are not good choices from a security perspective. They may have an unhealthy community, depend on a single developer, eschew security updates, or otherwise not be following best practices.

The open source community on GitHub has developed a culture around dynamic badges or shields [28] rendered in the project's markdown README to succinctly display markets of project health, including whether the last commit builds and passes its tests, test code coverage, and various other metrics or ratings. The u-root project's README is an example of a project with these dynamic project health badges [29]. Badges such as these could be used to prominently report and display a project's security health.

[d] "You Only Live Once." The YOLO acronym has become an aphorism that expresses the view that one is focused on the present moment and not worrying about possible consequences, taking on the risk.

Open source security health metrics tools, like the OpenSSF Scorecard [30], provide a structured, automated way to assess and mitigate the risks of using third-party libraries in your codebase. This tool allows enterprises to look beyond code vulnerabilities and evaluate adherence to broader security best practices.

The OpenSSF Scorecard provides a comprehensive set of checks that delve into various aspects of a project's security posture. For example, it scrutinizes the presence of a security policy and a code review process, indicating a project's commitment to secure development practices. Additionally, it verifies the use of branch protection mechanisms and signed releases, which can prevent unauthorized code modifications and ensure the integrity of software artifacts.

The Scorecard's analysis extends beyond code vulnerabilities, considering factors like the bus factor (number of maintainers), the project's maintenance state, and the presence of a security contact. This is crucial because unmaintained projects, or those with a single maintainer, are potential security liabilities if vulnerabilities go unpatched or ownership transfers to a malicious actor.

By incorporating the Scorecard's results into their decision-making process, enterprises can identify these red flags before integrating third-party libraries, minimizing the risk of introducing vulnerabilities and insecure practices into their own software supply chain. Moreover, the Scorecard can serve as a standardization metric for evaluating security curators who provide secured OSS components, ensuring a baseline level of security across the industry.

Upstream: Contributing Back

By providing their source code, the original authors allow the downstream consumer to more easily review and modify the software, for example, to fix security issues or add required features specific to their situation. Of course, after making the modifications, the revised code will still need maintenance.

This can create an ongoing burden on the downstream consumer, including making it hard to accept security patches. Because the original OSS project does not have their modification the consumer must apply the patch, and then re-apply their own modifications, which might need refining. This disincentivizes the down-stream consumer from staying up to date with patches, at a cost to security.

Upstreaming these changes can eliminate this burden. By submitting the improvements to the main OSS project, the maintenance burden will be shifted to the OSS project and shared by the community, and future security patches will be easy to consume. More important for OSS security, upstreaming security-relevant fixes increases security for other users of the project.

Unfortunately, upstreaming changes to OSS can have a lot of friction. For example, the OSS project may have specific code quality norms, which could require adjusting the submission. Some projects require a copyright assignment for contributed code, which can create friction if the contributor has a cautious or bureaucratic legal department.[e] Nonetheless, successful projects such as the Linux kernel show that upstreaming changes can be a critical enabler of quality open source software.

[e] OSS projects must be able to license all the code under their open source license. Thus, the copyright must either be assigned to the project or licensed to the project with a compatible license.

There is some nuance when discussing the contribution of *fixes* vs. *enhancements* to OSS projects. Fixes, including security fixes, tend to be narrow in scope and do not change the intended behavior. A code enhancement, on the other hand, may add entirely new functionality not included or even contemplated by the original software developers. The upstream maintainers may not accept such added functionality but will typically accept security fixes. To include new functionality is to implicitly accept the responsibility and overhead of maintaining it.

At the same time, from a corporate liability perspective it is easier for a company employee to submit a bug report to fix an issue with OSS that is being used at the company than it is for the employee to submit new functionality that may inadvertently reveal company intellectual property (IP) or strategy.

Large projects like the Linux kernel demonstrate another benefit of upstreaming changes, the complex social norms of open source software development. Many OSS projects are either explicitly non-commercial (e.g., Debian) or involve many different commercial entities working together (e.g., Linux). In both cases, social norms begin to influence the *direction* of the project and the project's willingness to work with contributors and consumers to adjust to their needs. These adjustments can pay dividends to downstream consumers in terms of future fitness and ease of adoption of the OSS project to their purposes.

Curators have the same incentives to upstream any changes they make, since it simplifies adoption of future changes. Upstreaming is simpler in practice for curators because they do it on a regular basis and typically have a long-term relationship with the relevant OSS communities and understand the expectations.

Findings:

- The problem is not finding software bugs but fixing them. With modern tools identifying bugs is much easier than fixing the bugs. With automation it is possible to identify hundreds of bugs, which when reported upstream may overwhelm an OSS project and lead to bug fatigue.
- Governments are beginning to propose secure by design and OSS consumption norms and legislate some of them (EU CRA).
- Staying up to date is important for critical online systems, as it enables the ability to accept and apply security patches quickly.
- SBOM and SLSA are unique ways to address two difficult problems, but they need greater adoption, development, and promotion to mature.
- There is a growing trend toward curated OSS software repositories being maintained by accountable providers. Curators make it easier to stay up to date by applying critical patches on behalf of their consumers. (Discussed more in Question 3)
- There is long term value to upstream changes, aside from improving the project for future consumers, the upstreaming of changes reduces maintenance burden on the consumer.
- OSS project norms may involve legal, technical, and social complications that make upstreaming new functionality difficult. In contrast, those issues generally do not apply for fixes.
- Upstreaming *all* changes to a project enables downstream consumers to approach the ability to auto-update downstream dependencies.
- Staying up to date transparently fixes many outstanding issues over time but requires automated build/test in practice.

- Not all OSS projects have good security hygiene and a vibrant community, and yet this rarely affects usage. Consumers should use project health as an important criterion for choosing their dependencies. OSS ecosystems can do more to clarify project health through projects like OpenSSF's Scorecards.

The act of sharing bug fixes and software patches to help others is a core part of the OSS ethos. Incentives to improve the up- and downstreaming of software, reduction of the friction for companies to contribute their fixes to OSS, and greater collaboration around building and maintaining trustworthy software repositories will help make software less vulnerable to known issues. Although these principles are well known, differing approaches to implementation creates friction and inefficiency in the ecosystem.

Finally, there is an unmet need for ecosystem-specific guidance, which remains mindful of the diversity of OSS while establishing consistent goals and capabilities. CISA has started work along these lines with its work on package repository security [31]. More work in that area is promising, especially if it leads to better integrated tools and workflow.

Recommendation

CISA should produce a guidance document on Open Source Consumption and Upstreaming. This guide would be used both as a technical guide for software engineers to understand the options around what to consider when selecting OSS components to use in their projects, as well as justification to management for upstreaming important changes that will benefit the source OSS project. The most important topics include:

- Control of the intake process for OSS;
- The value of staying up to date;
- The value of upstreaming at least fixes and often enhancements as well; and
- Stronger guidance on SBOM formats, and on how to use SBOMs and SLSA especially for composition of packages into larger artifacts, and a definition of actionable and testable composition goals.

***Question 2**: How should CISA work to encourage these norms and contributions from federal agencies?*

Discussion:

CISA has a number of ways to encourage OSS norms adoption, either within CISA and DHS or by promoting to other federal agencies or partners. There are at least four ways in which CISA can encourage these norms:

Procurement: Most procured software includes OSS components in some form, and procurement guidelines remain a strong mechanism to drive adoption.
Awareness: Promote OSS consumption best practices through visible indicators, such as including adoption of best practices as part of existing assessments, and product rating labels.
Clearing house: Facilitate transparency and the exchange of information regarding OSS package security among existing actors.
Centralized Curation: Focus much of the responsibility for consumption in one agency that then helps others with safe consumption. "Others" can include agencies, SLTT governments, and even critical infrastructure suppliers. This is a non-trivial job, but it is more efficient overall, and more likely to succeed, if centralized.

Procurement

When acquiring software or services there is an opportunity to request that vendors clearly elaborate what norms they follow, and what their roadmap is for enabling the principles of Secure by Design. This additional context will help differentiate offerings vendors and allow CISA to financially support those that are best aligned with their goals.

Most of the focus to date has been on reducing the presence of major vulnerabilities, which is a critical goal. However, there has been less focus on the reduction of *future* vulnerabilities, which should focus on the SSDF and in particular on the vendor's ability to stay up to date as vulnerabilities are discovered and fixed upstream. This ability should be a major requirement for vendors.

There is also value in requiring vendors to disclose their intent to upstream changes or not, and specifically with which projects. The use of SBOMs will already reveal the projects these vendors use. By revealing the intent to upstream, the vendor indicates that at least those listed projects will be roughly up to date and should be at lower risk in terms of future vulnerabilities.

Awareness: Existing Programs

Existing CISA programs that could help with awareness of OSS consumption best practices include:

- Cyber Infrastructure Survey. CISA could recommend a specific set of practices to include as part of the survey CIS.

- National Cyber Awareness System (NCAS). The NCAS "provides situational awareness to technical and non-technical audiences by providing timely information about cybersecurity threats and issues and general security topics", including weekly vulnerability bulletins and best practices. Vendor OSS best practice adoption assessments and ratings could be promoted through this system.

- Infrastructure Survey Tool (IST). The IST can be augmented to include adherence to OSS safe consumption norms.

Clearing House Activities

CISA already provides some clearinghouse capabilities [32][33] for distributing bulletins about threats, vulnerabilities, threat actors, as well as visualization tools [34]. It also already has an assessment program for external dependencies [35]. CISA could enhance its existing offerings to include a clearinghouse about OSS consumption. Some ideas include:

- A survey of OSS package usage by infrastructure. Who uses it, what version, and who is contributing back; how many active users are impacted by it. It is not realistic to hit 100% coverage, but a good amount could come from just crawling published dependency lists and commit logs.
- A clearinghouse of OSS package life cycles to answer questions such as whether it is actively maintained or is highly forked. This could be an extension to CISA's Cyber Threat Hunting (https://www.cisa.gov/resources-tools/services/cyber-threat-hunting). The task would be to apply the result of the survey and look for OSS projects that are vulnerable to cyber threat actors and create a clearinghouse for adoption.

Centralized Curation

Centralized curation could form a basis for a more substantial effort that both simplifies safe consumption and improves upstreaming of fixes and especially security patches. There are at least two variants of centralized curation in use by the government today, Iron Bank and Code.gov.

The Department of Defense operates "Iron Bank" [36], which is part of the larger Platform One effort to support a secure modern Kubernetes-based OSS platform. The Iron Bank is a collection of curated containers. These containers are kept up to date, and the curation team assists vendors with the hardening of critical images. This is a great example of how centralized curation allows the users of Platform One to be up to date and more secure with less effort. By working with vendors, they also encourage upstreaming of changes.

The General Services Administration's "code.gov" platform is a shared collection of OSS, but with less curation so far. This is intended for OSS produced and maintained by the government, not external packages used by agencies. Thus, this is mostly orthogonal to safe consumption, but it could be a good place for the government to lead by example and learn lessons, as there are many OSS dependencies in use by these projects.

There is also a need for specialized curation for each critical infrastructure sector. Each sector has some unique software dependencies that are unlikely to get attention from broader curation efforts. Sector Coordinating Councils[f] at the minimum should enumerate the critical OSS packages for their sector and push for some form of curation for those packages.

Findings:

- Regulation to date focuses on limiting the *presence* of vulnerabilities (e.g. FedRAMP), but says little about the development process, the SSDF or Secure by Design.
- Enhancing the existing awareness programs could help promote best practices in federal use of OSS.
- A Clearing house that enumerates the most popular or critical OSS by sector could help focus attention on the software on which most depended.
- Understanding what works for "Iron Bank" would help improve CISA Secure by Design guidance documents.
- For OSS consumed by governments, it would be better to manage consumption centrally. It would allow smaller agencies, SLTT governments, and even critical sectors, to consume OSS in a simpler and safer way.
- For critical sectors, Sector Coordinating Councils should play a role in defining, tracking and improving the OSS that is critical to their sector. The focus should be on the OSS that is relatively specific to their sector.

Recommendations:

CISA should enhance existing awareness programs to promote OSS information sharing and norms, including the creation and maintenance of a clearing house with up-to-date information about OSS consumption, working to enhance the existing centralized curation services. Investigate if the biggest improvements would come from centralizing and

[f] "The Sector Coordinating Councils (SCCs) are self-organized and self-governed councils that enable critical infrastructure owners and operators, their trade associations, and other industry representatives to interact on a wide range of sector-specific strategies, policies, and activities." [37]

sharing accountability for safe consumption and keeping dependencies up to date in one agency, perhaps in the style of DoD's Iron Bank, but intended for all agencies, SLTT governments, and critical sectors.

Question 3: *How can CISA shift the burden of securing open source software to rest on those companies who routinely build commercial products by offering modified versions of the open source project for a fee, often withholding capabilities from the free version to incentivize consumers' purchase of the enhanced version?*

Discussion:

The process of securing software is an immensely complicated task at any meaningful scale, and instead of exploring specific technologies or products to do so, this report explores the main characteristics of what a secure code base and deployed project would look like. It consists of roughly three different functions: How the code is developed and deployed, how that code is patched and securely maintained within a software lifecycle, and what the business model based on one or more OSS projects is.

First is understanding if the software code in question is being developed securely. Indicators would include if a SDL is implemented and working appropriately. Is the NIST SSDF being adhered to? The goal is to understand the provenance of the source code, how secure it is.

Once source code is compiled or interpreted and is ready for production, the next step is understanding how secure the default deployment configuration is. Are safe defaults selected, are modern exploit mitigations and compiler options used, is the code written in a memory safe language, etc.

The software is ultimately loaded on a computer hardware architecture to execute on, whose characteristics help determine how secure the final software/hardware combination is. The solution as delivered should be configured by default to fully utilize hardware and OS-provided security mechanisms that are available. For example, CPU-based memory protection technologies, Hardware Security Modules (HSM) to hold secret encryption keys, and virtualization-based system protections would be common here. How is the software securely deployed? Are critical bits of code running in secure enclaves? Are processes least privileged and sandboxed? What techniques are being used to enforce security boundaries?

Understanding the threat models around how the upstream project development process is secured from attacks against its supply chain, or from unauthorized code check ins should a developer's workstation be compromised would be important. Related to this would be protections against insider threats such as adversarial employees or rogue developers. Answering these kinds of questions helps the consumer better assess risks, with SLSA levels an example of an indicator.

The second characteristic to understand is around the maintenance life cycle of the deployed software. Does the project that the deployed software depends on have the necessary interest and resources to hunt for vulnerabilities in their code and computing infrastructure, or to respond to bugs reported in their software? Essentially how mature and sustainable is the OSS project?

Burden Shifting and Curators

In exploring methods of shifting the burden of securing open source software to rest on those companies who routinely build commercial products, it is important to separate the accountability for delivery of secure software from the ongoing maintenance of the underlying OSS project.

Broadly speaking, the burden of accountability, legal liability, is often imposed in an attempt to regulate safety through the efficient allocation of resources. If a toaster explodes in normal operation, the consumer will suffer harm. In the case of a toaster, liability is used to incentivize the manufacturer to allocate resources to toaster safety, internalizing the costs. However, if the consumer has been using the toaster in a particularly dangerous way, the consumer may bear some responsibility, and therefore must bear some of the costs. When determining who should bear the costs, liability theory looks to causation, as well as who is best positioned to mitigate the danger.

In general, people are liable for an injury caused by their breaching a duty of care. Without a duty of care, there is no liability. Thus, through imposing a duty of care, or voiding contract terms that disclaim one, the law can shift the costs of an injury, reallocating to a party better able to avoid or mitigate the risk, often because that party has more resources and better information and ability to address the issue.

In the case of OSS, there are huge differences in the resources, information, and ability across projects. For some community maintained projects, which are vital to our connected society, the developers are volunteers, contributing code because they use the software, and saw a need. These maintainers and contributors are poorly positioned to bear the potential costs of a software bug, even if they know the codebase well and are adept at its development.

As discussed in the introduction, OSS projects almost always include a license term providing that the software is “as is,” and thus the consuming company takes on *all* responsibility and liability for any security issues in that code. Unfortunately, the consumer may not be best positioned to mitigate these risks either, leading to the need for an accountable intermediary or curator.

When a consumer becomes aware of a security issue, they have essentially three choices to fix the issue:

1. **Direct fix**: If they have the skill they can fix the problem in their copy of the OSS software immediately, and then try to submit the changes upstream. This is a great approach, but it requires technical capability and management willingness in the consumer.
2. **Community fix**: They can work with the community for a fix. This is a great model *if* there is an active community around the project, but consumers regularly use projects with limited support in practice. Thus, the choice of *which* projects to consume, based on their ability to address security issues, is a critical part of any long-term solution.
3. **Curator fix**: A third party curator typically hired by the consumer, can either do the direct fix (and has the capability to do so), or work with the community for a joint fix. This is essentially the option to outsource to a more sophisticated entity to try and get the fixes developed and committed.

When companies offer support contracts that cover security issues in OSS, they are acting in the paid curator role for that software. Although such security fixes are typically also moved upstream in these cases, the upstream version tends to have the fix only in the latest version. In contrast, the curator typically also fixes the supported version, which is often significantly older. Fixes to old versions is one of the main values of such a support contract, exactly because it is often

difficult and specialized work. We should not expect such work to be free, even if the overall software is free. Unfortunately, most of the OSS in use do not have a support option today.

The OSS project is always responsible for the long-term evolution of their code, but they are not liable for flaws in any particular version. Thus, there is usually some integration work to do to accept a "direct fix" patch into the project. This negotiation of the right patch for upstream is an important part of the process — the project has more considerations than this single consumer when creating the right fix. The proposed patch is a great start and has significant value, but the final version of the patch comes from the community.

Curators Can Undertake the Accountability Burden

Although more sophisticated consumers can take accountability themselves, most lack the resources and skills to do it across *all* the software they consume, leaving a gap between "as is" OSS and the liability they take on by producing software. This gap leads to the need for an "accountable intermediary".

Curators can help solve this key challenge by getting paid to take on accountability and ideally legal liability as a business model. When done well the curator has sufficient resources and the ability to gain information on security flaws and implement fixes, and thus, can provide the necessary assurances. Although some big companies and the DoD do this internally, this kind of support is needed broadly, especially for smaller companies, local governments and less technical consumers of OSS.

There are at least two levels of support a producer of OSS-based packages could provide:

- **Best effort**: Packages are freely available but come with no promises of fitness or expectations for timely security fixes. Package managers mostly fit here.
- **Curated packages**: Collections of packages that have been tested and are kept up to date, possibly including older versions with up-to-date security patching. These typically come through a paid support contract. Paid Linux distributions tend to fit this model, such as the Linux distributor Red Hat that fixes security issues in packages as part of their support model.

Most consumers with security requirements should consider using curation rather than taking on accountability directly. Currently this is a nascent and evolving area, so the use of curators is not yet available universally, but there are some examples:

- Some OS distributions as described above, including versions of Linux and BSD. However, these typically only cover lower-level packages.
- The FreeBSD Foundation recently announced a service to aid commercial users with the NIST Secure Software Development Framework [38].
- Tidelift partners with open source maintainers and pays them to implement secure software development practices and validate the practices they follow.
- Google provides secure versions of 2500+ Java and Python packages, through a (paid) offering called "Assured OSS" [39]. These packages are kept up to date and security issues are addressed as they arise.
- The Nix Packages collection [40] includes automated and tested builds of over 100,000 open-source packages,[41] in addition to the NixOS Linux distribution based on them, and these packages may be used on other Linux and macOS-based systems.

The US government's primary lever to improve security has been to create accountability through regulation, such as FedRAMP or through procurement rules that generally place liability on the vendor. Encouraging the use of curation as a way to meet liability obligations would improve security in practice by shifting this burden to specialists. Specialists can help keep packages up to date, or apply security fixes to older versions, and they are more likely to drive automated build and testing capabilities. The existence of more curation options would benefit smaller companies as SLTT governments as well.

CISA can also help define the expectations for curators, including the adoption of various best practices, including use of SBOMs, SLSA, supply-chain security, and notification of newly discovered vulnerabilities.

Finally, curators can also help with certification. There are already vendors that produce FIPS-certified libraries, which greatly reduces the burden of proving FIPS compliance for their customers. This can be generalized to any kind of certification. There is significant, specialized and ongoing work for software certification, which makes it ideal for outsourcing to a curator, where that work can benefit many consumers.

Standardized Automation Templates

It is possible to help reduce the friction for OSS maintainers to participate in security evaluations by encouraging the creation and deployment of standardized continuous integration (CI) automation templates. Today, the processes for building and testing packages are typically written down in English, and thus require manual interpretation. For example, to build software you may need to set up a virtual machine with various tools in order to run scripts that actually do the build.

Templates for automation enable automated builds by third parties that are not already familiar with the package. This has several advantages:

- It encourages use of a trusted build system that meets SLSA guidelines and can sign the produced artifacts.
- It enables third parties to cover the costs of builds (and testing), rather than expecting maintainers to cover those costs.
- It forms an excellent way to enforce best practices and thus improve many aspects of security, including SBOM generation and SLSA compliances (by building those into the automation in a uniform way).
- It helps with incident response by simplifying the work of responders to rebuild related parts of the dependency chain as needed.
- It allows larger, better-staffed projects to assist smaller projects by sharing more of the CI pipeline and reporting process.
- This process can also lead to "reproducible builds" in which multiple parties can independently verify a given build by building it themselves.

An example project in this direction is "OSS Rebuild" [42], which uses standardized build descriptions to enable others to build those packages. In many cases, the description can be generated automatically, but if not, maintainers or others can write the description by hand, thus encoding their process in a form that enables automation.

The same approach should be used for automated testing, including unit tests, integration testing and fuzz testing (when used). As with builds, there can be significant operational costs to testing, which in practice disincentivizes adding more tests or running them frequently. But the most secure software should have many tests that ideally are run on every change. Automation templates can decouple the process of testing from how these costs are covered.

This can also address an important related problem, which is that during incident response, it is difficult to know if a proposed change will break functionality or not, as there are not enough test cases for many OSS packages in practice to make that determination. This is a sticking point for machine learning-driven automated patching as well: how do we know if a patch will work in practice or not?

Sorting out automation for builds and testing should also lead to more test cases over time, because of the way it enables consumers or curators to pay these costs. Automated generation of test cases is an active area of research that also merits more investment.

Templates should:

- Be standardized for key languages and build systems
- Originate from a single source point with a clear versioning information
- Have a mechanism to stay up to date as process goals evolve.
- Generate the metadata needed to address software supply-chain risks, including SBOM information and SLSA metadata.
- Facilitate the delegation of computationally intensive tasks, such as fuzzers or large test suites, to larger, better resourced organizations.

Delegated tasks can submit results to an aggregating entity on behalf of smaller projects with no loss of security, as the commit state is identical.[g] Delegation allows smaller projects to participate in SBOM generation with minimal burden on the maintainers (e.g. simply accepting a pull request for an automation file), while allowing better-staffed and resourced organizations to run batteries of expensive analyses on the repository at their discretion. This helps shift the burden of compliance to large, commercial organizations or curators that rely on small OSS projects for critical components.

Findings:

- OSS projects are responsible, but not liable, for their software whereas commercial software vendors are both responsible and liable for their software, subject to software license limitations.
- Small OSS projects lack the resources to keep up with evolving security norms.
- Many organizations would be well-served by having a commercial relationship with a curator that enables internal staff to self-serve and choose OSS projects among the curator's supported collection.
- Conversely, the accountability burden *cannot* be placed on OSS projects directly, as the consumer agrees not to do so just by using the software — the "as is" license makes this very clear.
- It is encouraging to see some OSS projects voluntarily agreed to provide more accountability in some areas.
- Promoting curation as a way to meet security obligations, particularly for critical systems, would improve security broadly for smaller or less technical organizations.
- Curation is also an easier path to certified software, including FIPS compliance for example.
- Standardized automation templates can drive better conformance to best practices and enable broader use of SBOMs and SLSA, especially for small critical projects. Such automation can also shift the cost burden from projects to better-funded consumers or curators.

[g] Ideally, this would be through reproducible processes so that others can verify the work. When that is not possible, signed output is the next best option, and then the consumer can decide if they trust the signing entity. For example, in Google's Assured OSS system, Google signs the produced artifacts.

- Organizations could provide automation that continuously evaluates and provides a status badge to OSS projects attesting to various aspects of security quality
- Security by scanning for vulnerabilities after they are built is not security by design.
- CISA could embrace and extend the Cyber Trust Mark (CTM) program to better expose to consumers what OSS norms are being followed from a high level perspective.
- CISA could encourage federal departments or agencies to consider CTM scores when procuring software.
- CISA should encourage curators of open-source software to adhere to the Supply-chain Levels for Software Artifacts framework for pre-built software artifacts that they provide

Recommendations:

CISA should endorse the curation model for open source and encourage its use by federal agencies and their vendors as a way to meet liability obligations and improve security in practice by shifting this burden to specialists. CISA should also encourage the use of standardized templates to automate and harmonize builds, tests and reporting, and to enable third parties to cover these costs, further shifting the burden to better-funded organizations.

Question 4: *Are there additional recommendations for CISA to (a) support secure by design outcomes in AI systems that are distributed under open source compatible terms, or (b) protect both the public and private sector from potential harms from misuse of foundation AI models with widely available model weights?*

Discussion:

As Wikipedia explains succinctly, "A foundation model, also known as a large AI model, is a machine learning[43] or deep learning model[44] that is trained on broad data such that it can be applied across a wide range of use cases[45]. Foundation models have transformed artificial intelligence (AI)[46], powering prominent generative AI[47] applications like ChatGPT[48]. The Stanford Institute for Human-Centered Artificial Intelligence's (HAI) Center for Research on Foundation Models (CRFM) created and popularized the term."

Foundation models are general-purpose technologies [49] that can support a diverse range of use cases. Building foundation models is often highly resource-intensive, with the most expensive models costing hundreds of millions of dollars to pay for the underlying data and compute required. In contrast, adapting an existing foundation model for a specific use case or using it directly is much less expensive.

Early examples of foundation models are large language models (LLMs)[50] like OpenAI's "GPT-n" series[51] and Google's BERT[52]. Beyond text, foundation models have been developed across a range of modalities—including DALL-E[53] and Flamingo for images, MusicGen for music, and RT-2 for robotic control. Foundation models constitute a broad shift in AI development: foundation models are being built for astronomy, radiology, genomics, music, coding, times-series forecasting [54], and mathematics."

Models require large quantities of data to be trained on, and for the training data to be useful it must be tagged with attributes that describe it. In the case of a picture, elements that make up the composition of the picture would be included. A picture of a car could include data such as the make, model, year, color, if the car is fast, how many people it can seat, etc. As data tagging sets grow in size and accuracy it becomes possible to use them to pre-tag new data before being sent to enormous numbers or people, often in other countries, for manual review.

Government Special Considerations

Government entities that wish to use AI models have some special considerations - if the model will be used for the provision of government services, it must be sure that the model provides a fair and even handed application, without bias from the weights or data sets. For example, the U.S. Commission on Civil Rights recently released a report on the civil rights implications of the Federal Use of facial recognition technology, noting the "significant risks to civil rights, especially for marginalized groups who have historically borne the brunt of discriminatory practices." [55] Likewise, the government should consider risks to civil rights and liberties before implementing AI in government services.

Moreover, government entities will want to ensure that the data set's origin respects legal norms and human rights. For example, "After releasing ChatGPT in 2022, OpenAI was widely criticized for outsourcing the data labeling work that helped make the chatbot less toxic to Kenyans earning less than $2 hourly."[56]

As this report is being finalized, reports of OpenAI threatening to remove access to their latest "o1" model, should researchers attempt to understand how the model reasons, is the antithesis of transparent AI and should be of concern to governments [57].
Consumer rights initiatives such as the Right to Repair [58] have made compelling arguments and legislative progress toward enshrining a consumer's right to know what is going on inside their computers. These arguments should be extended to include the transparency, auditability and reproducibility of AIs.

Enhancing Transparency, Auditability and Reproducibility

Having an open, robust and verifiable record of where AI data comes from, what model weights were used, and assurance that it has not been tampered with can help auditability and trust. The provenance of data used for AI models is key to reproducibility, ensuring that we know that a given data set is in fact the data set used for a foundation model, and has not been subject to modification or even malicious poisoning. This is especially important for OSS AI models available for wide adoption.

The data used to train the LLM can be proprietary or publicly available, or a combination of both, and there is a growing collection of common reference data sets that are used for training. Understanding both the data source and the model weights is necessary. While many different implementations may start with the same training data the weights that are applied might be quite different.

Given that much of the training data is publicly accessible, the weights that different companies or projects apply to it are frequently considered proprietary and provide their differentiation and competitive advantage. In some cases, access to the proprietary information may be restricted, by way of interpreting the Digital Millennium Copyright Act's (DMCA) prohibitions on circumventing copyright to apply to independent AI research into models.

However, the DMCA has safe harbor provisions which allow the Librarian of Congress to grant exemptions in a triennial rulemaking process. The exemptions currently include security research of technologies deemed critical, such as election technology or medical devices, which may not be broad enough. As the Department of Justice has recommended, "the Copyright Office [should] consider clarifying the existing exemption to ensure its application to good-faith security research regarding AI systems and other, similar, algorithmic models." [59] Extending these protections to AI research would allow the research needed help to inform both policy makers and technical coordinators.

Summary Information Cards Can Help Provide Needed Transparency

Summary information about models and weights can also enhance transparency. For example, Google helped create two types of summary information cards, Data Cards and Model Cards, which provide information to help describe the data sets and the model.

Google's "Data Cards" are a way to describe the data used in the training of an LLM. Google describes them as "structured summaries of essential facts about various aspects of ML datasets needed by stakeholders across a project's lifecycle for responsible AI development." Google has created a Data Card template that "captures 15 themes that we frequently look for when making decisions - many of which are not traditionally captured in technical dataset documentation." [60]

"Model Cards," created by Google in 2018 to help with AI transparency, are also a critical part of building responsible systems. [61] They help "to organize essential facts of machine learning models in a structured way." "Model cards take many different forms depending on the use case. Given the rapid evolution of AI technologies and as new, industry-wide benchmarks for performance and safety emerge, model card structures and content must be flexible. They're not one-size-fits all."

These types of cards, which we call transparency cards as a category, provide useful metadata about a model, and should be encouraged. However, they could be enhanced with more information on the origins or providence of the data, like what the coffee industry does with "fair trade" beans as a way to signal to purchasers that their product was ethically sourced.

In addition, transparency cards do not currently allow for reproducibility or a way to verify that a given model has actually used the datasets and models described in the cards. In software, "[r]eproducible builds, also known as deterministic compilation, is a process of compiling software which ensures the resulting binary code can be reproduced. … For the compilation process to be deterministic, the input to the compiler must be the same…" [62] This same concept can be applied to the AI model ecosystem.

To develop reproducibility for AI models, however, the key is not the source code. Instead, this would require keeping a copy of the data set(s) and the weights in a manner that can be verified as the basis of the model. For example, create and publish a hash of the data set after model creation, such that the data set used for inputs can be later verified by matching the hash.

While models may be different each time the weights are applied, by ensuring the provenance of the exact same data set, one can reduce the risk of security harms like data poisoning.

Findings:

- The use of transparency cards, like these model and data cards, should be embraced and expanded to provide better transparency.
- Openness in the provenance and design of AI models, allowing consumers, whether the government, corporations or citizens, to better understand the trustworthiness of the model's results.

- More information on models, their data set and their weights, can allow for third party audits and security research on the models.
- CISA could support the Librarian of Congress granting an exemption from the DMCA to independent researchers accessing an AI's training data and models.
- The government could promote openness by considering the level of transparency and openness when selecting models to use.
- Because of the government's unique position and perspectives in the AI ecosystem, it should consider participation in appropriate working groups.
- Additional data should be collected and published to better replicate and verify a model after the fact.
    - This would require keeping a copy of the data set(s) and the weights in a manner that can be verified as the basis of the model. For example, publish a hash of the data set after model creation, such that the data set used can be later verified by matching the hash.
    - The goal of this effort is the AI equivalent of software dependencies. By ensuring transparency in the provenance of all data used to train a model in the event of a poisoned and manipulated data source, impacted models become discoverable.

Recommendations:

CISA emphasizes the importance of "reproducible" builds and should extend that notion to encourage open source AI models to enhance transparency, auditability and reproducibility including providing information on the model, the data, as well as the underlying data set. These transparency cards, such as the model and data cards discussed above, should also include information on the provenance of the data and the model, sufficient for government consumers to understand any ethical implications, and CISA should support the Librarian of Congress to grant exemptions from the DMCA to researchers seeking to access the underlying information.

Conclusion:

We believe the long-term solution for safe consumption requires a structural change in the mechanisms around the consumption of open-source software. This is driven by the fundamental disconnect between the "as is" disclaimer used in open source, which is necessary for the developers, and the desire for top-down accountability.

This gap can be bridged by curation, where an accountable intermediary takes responsibility for a subset of OSS packages. Although the curation model has existed for many years in various forms, it is not widely used above the level of OS distributions, which only cover a fraction of the important OSS in use today.

The other findings are also critically important and will provide needed transparency and automation tools that will help CISA promote safe OSS consumption. When combined with the curation model, these will help provide the structural changes necessary for developers and other OSS consumers to manage and mitigate their risks.

Appendix A

TASKING

Technical Advisory Council: The federal government, critical infrastructure, and the broader public are highly dependent on open source software (OSS). As illustrated by the Log4shell vulnerability, vulnerabilities in widely used open source software can have widespread downstream effects. CISA recognizes the immense value that open source software has provided and wants to ensure that the OSS we depend on is secure. CISA is concerned about both vulnerabilities present in widely used OSS libraries and supply chain attacks targeting open source software producers and distribution channels, and **seeks to foster the adoption of more secure development practices within open source software**.

• Where We Are. In line with the National Cybersecurity Strategy, CISA is working to help secure the federal government's usage of OSS, as well as helping strengthen the security of the broader OSS ecosystem. CISA has engaged heavily with the open source community to date, including hosting a roundtable discussion with OSS community leaders, participating in the Open Source Software Security Foundation (OpenSSF)'s conference in Vancouver and summit in DC, and issuing a RFI with the White House, National Science Foundation (NSF), and the Defense Advanced Research Projects Agency (DARPA) on OSS security.

• New Initiatives. CISA's Open Source Software Security Roadmap lays out CISA's planned actions around OSS security. This includes engaging with package managers and code repositories to establish shared principles around actions to increase security, such as requiring multi-factor authentication for maintainers of critical projects. CISA is also undertaking steps to understand which OSS packages are most prevalent among the federal government and critical infrastructure. Furthermore, we are building out relationships with OSS communities, including by establishing a channel for communication.

• Where We Want to Go in 2024. CISA wants to ensure that we can depend on the security of critical open source packages. This includes the federal government and private companies acting as good stewards of the OSS they depend on, with financial and code contributions.

Questions for the Committee: We are eager to further explore how CISA, in line with our Secure by Design goals, can encourage companies to be better stewards of the open source software they depend on and produce. To that end, we present the following two guiding questions.

June report: OSS consumption norms
Guiding question: How should CISA encourage the adoption of safe consumption norms for open source software, while also encouraging companies to contribute fixes and enhancements back to the open source projects? Additionally, how should CISA work to encourage these norms and contributions from federal agencies?

September report: Disparities in commercial OSS security
Guiding question: How can CISA shift the burden of securing open source software to rest on those companies who routinely build commercial products by offering modified versions of the open source project for a fee, often withholding capabilities from the free version to incentivize consumers' purchase of the enhanced version?

Additional Context: some open source projects, which are developed by employees of corporations that sell related products, lack fundamental security-related capabilities that are present in related commercial products. This technique

– to give away, under an open source license, a limited form of a project while charging for a more secure version -- is commonly referred to as “commercial open source software” today. However, many small and medium enterprises, as well as OT/ICS service providers, never purchase the “enhanced” versions, opting to keep costs low by continuing to rely on open source software from free and public sources.

Appendix B Referenced in Document

1. https://www.cisa.gov/securebydesign
2. https://w3techs.com/technologies/details/ws-nginx
3. https://www.hbs.edu/faculty/Pages/item.aspx?num=65230
4. https://www.sonatype.com/hubfs/1-2023%20New%20Site%20Assets/SSCR/8th-Annual-SSCR-digital-0206%20update.pdf
5. https://www.whitehouse.gov/oncd/briefing-room/2024/01/30/fact-sheet-biden-harris-administration-releases-end-of-year-report-on-open-source-software-security-initiative
6. https://www.dhs.gov/dhs-digital-strategy
7. [https://www.cisa.gov/opensource
8. https://www.cisa.gov/resources-tools/groups/cyber-safety-review-board-csrb
9. https://en.wikipedia.org/wiki/Free_and_open-source_software
10. https://opensource.org/license
11. https://media.defense.gov/2023/Dec/11/2003355557/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN%20RECOMMENDED%20PRACTICES%20FOR%20MANAGING%20OPEN%20SOURCE%20SOFTWARE%20AND%20SOFTWARE%20BILL%20OF%20MATERIALS.PDF
12. https://www.microsoft.com/en-us/securityengineering/opensource
13. C3 Principles for Package Repository Security | wg-securing-software-repos (openssf.org)
14. https://dl.acm.org/doi/10.1145/3347446
15. https://newsroom.eclipse.org/news/announcements/open-letter-european-commission-cyber-resilience-act
16. https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses
17. https://en.wikipedia.org/wiki/Open-design_movement
18. https://en.wikipedia.org/wiki/Open-source#Society
19. https://en.wikipedia.org/wiki/Maker_culture
20. C5 https://opensource.org/blog/what-is-the-cyber-resilience-act-and-why-its-important-for-open-source
21. https://csrc.nist.gov/Projects/ssdf
22. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf
23. https://deps.dev
24. https://slsa.dev/
25. https://slsa.dev/blog/2022/05/slsa-sbom
26. https://en.wikipedia.org/wiki/Downstream_(software_development)
27. https://en.wikipedia.org/wiki/Upstream_(software_development
28. https://shields.io/
29. https://github.com/u-root/u-root?tab=readme-ov-file#u-root
30. https://github.com/ossf/scorecard
31. https://repos.openssf.org/principles-for-package-repository-security
32. https://www.cisa.gov/resources-tools/services/automated-indicator-sharing-ais-service
33. https://www.cisa.gov/resources-tools/services/cyber-threat-information-sharing-ctis-shared-cybersecurity-services-scs
34. https://www.cisa.gov/resources-tools/services/infrastructure-visualization-platform-ivp
35. https://www.cisa.gov/resources-tools/services/external-dependencies-management-assessment

36. Platform One | Services | Iron Bank (dso.mil)
37. https://www.cisa.gov/resources-tools/groups/sector-coordinating-councils
38. https://freebsdfoundation.org/news-and-events/latest-news/freebsd-foundation-announces-ssdf-attestation/
39. https://cloud.google.com/security/products/assured-open-source-software
40. https://github.com/NixOS/nixpkgs
41. https://repology.org/repository/nix_unstable
42. https://github.com/google/oss-rebuild
43. https://en.wikipedia.org/wiki/Machine_learning
44. https://en.wikipedia.org/wiki/Deep_learning
45. https://en.wikipedia.org/wiki/Foundation_model
46. https://en.wikipedia.org/wiki/Artificial_intelligence
47. https://en.wikipedia.org/wiki/Generative_artificial_intelligence
48. https://en.wikipedia.org/wiki/ChatGPT
49. https://en.wikipedia.org/wiki/General-purpose_technology
50. https://en.wikipedia.org/wiki/Language_models
51. https://en.wikipedia.org/wiki/OpenAI
52. https://en.wikipedia.org/wiki/BERT_(language_model)
53. https://en.wikipedia.org/wiki/DALL-E
54. https://en.wikipedia.org/wiki/Time_series
55. https://www.usccr.gov/news/2024/us-commission-civil-rights-releases-report-civil-rights-implications-federal-use-facial
56. https://time.com/6247678/openai-chatgpt-kenya-workers/
57. https://www.wired.com/story/openai-threatens-bans-as-users-probe-o1-model/
58. https://www.ifixit.com/Right-to-Repair
59. https://www.copyright.gov/1201/2024/USCO-letters/Letter%20from%20Department%20of%20Justice%20Criminal%20Division.pdf
60. https://sites.research.google/datacardsplaybook
61. https://modelcards.withgoogle.com/about
62. https://en.wikipedia.org/wiki/Reproducible_builds

Appendix C

The following TAC subcommittee members contributed towards this report:

- Mr. Jeff Moss, Chair
- Mr. Eric Brewer
- Mr. Dino Dai Zovi
- Mr. Luiz Eduardo
- Mr. Royal Hansen
- Mr. Andrew Huang
- Mr. Karl LeBoeuf
- Ms. Maria Markstedter
- Mr. Kurt Opsahl
- Ms. Runa Sandvik
- Mr. Yan Shoshitaishvili
- Mr. Kevin Tierney
- Ms. Rachel Tobac
- Mr. Shawn Webb
- Mr. David Weston
- Mr. Bill Woodcock