MITRE's Response to the EOP RFI on Open Source Software Security

November 8, 2023

For additional information about this response, please contact:

Duane Blackburn
Center for Data-Driven Policy
The MITRE Corporation
7596 Colshire Drive
McLean, VA 22102-7539

policy@mitre.org
(434) 964-5023

About MITRE

MITRE is a not-for-profit company that works in the public interest to tackle difficult problems that challenge the safety, stability, security, and well-being of our nation. We operate multiple Federally Funded Research and Development Centers (FFRDCs), participate in public-private partnerships across national security and civilian agency missions, and maintain an independent technology research program in areas such as artificial intelligence, intuitive data science, quantum information science, health informatics, policy and economic expertise, trustworthy autonomy, cyber threat sharing, and cyber resilience. MITRE's 10,000-plus employees work in the public interest to solve problems for a safer world, with scientific integrity being fundamental to our existence. We are prohibited from lobbying, do not develop or sell products, have no owners or shareholders, and do not compete with industry—allowing MITRE's efforts to be truly objective and data-driven. Our multidisciplinary teams (including engineers, scientists, data analysts, organizational change specialists, policy professionals, and more) are thus free to dig into problems from all angles, with no political or commercial pressures to influence our decision making, technical findings, or policy recommendations.

MITRE has a long history in the Software Assurance and Software Supply Chain areas. MITRE founded the Common Vulnerabilities and Exposures[1] (CVE®) in 1999 and since has partnered with industry and government leaders to create additional foundational efforts such as the Common Weakness Enumeration[2] (CWE™) and the open-source software scoring project Hipcheck.[3] MITRE has also been a key player in the quarterly Software and Supply Chain Assurance Forum.[4] Currently, MITRE is working with community partners and the Open Source Security Foundation[5] (OpenSSF) to establish a better understanding of open-source software through standard metrics.

Introduction and Overarching Recommendations

The rise of connected digital space has presented opportunities for companies, governments, and individuals across the globe. It has also introduced an unprecedented level of risk to mission execution across these same company, government, and individual pursuits.

In this response, we distinguish between two broad categories of open-source software: Free and Open-Source Software (FOSS) and Open-Source Software (OSS). They are often managed in different ways. The two categories are gross generalizations in nature and used here for simplicity only; not all FOSS is unmanaged, and not all OSS is managed responsibly.

FOSS offers a valuable form of expression to those who create and post it. Communities of developers with shared interests and concerns can rapidly produce and share code, collaborate to solve shared problems, and achieve significant satisfaction when their code and libraries are used

[1] CVE. 2023. MITRE, https://cve.mitre.org/. Last accessed November 1, 2023.

[2] Common Weakness Enumeration. 2023. MITRE, https://cwe.mitre.org/. Last accessed November 1, 2023.

[3] Hipcheck. 2023. MITRE, https://github.com/mitre/hipcheck. Last accessed November 1, 2023.

[4] Cybersecurity Supply Chain Risk Management. 2023. National Institute of Standards and Technology, https://csrc.nist.gov/Projects/cyber-supply-chain-risk-management/ssca. Last accessed November 1, 2023.

[5] About OpenSSF. 2023. Open Source Security Foundation, https://openssf.org/about/. Last accessed November 1, 2023.

directly or incorporated in other software products. Healthy forms of growth, competition, learning, and collaboration can often be seen in FOSS communities.

In the early days of widespread connectivity, the potential for harm from the use of FOSS was low and the benefits of growing readily available code libraries were considerable. Encouraging the use of FOSS served the community well for a time.

In some situations, encouraging fast-paced, uncontrolled development and propagation of FOSS may still serve a learning-and-growth purpose. One might liken the FOSS environment to an innovation lab; deployed software, by contrast, might be thought of as a productized offering. Both have value. Unfortunately, FOSS today conveys more unidentified risk than potential benefit. Contributors around the globe, with or without attribution, can accidentally or deliberately inject means of compromise into a code library and release it into the wild. Those who leverage FOSS will have no insight into the developers of the software or the quality and security of the code.

OSS can be more controlled with a limited number of authorized contributors, and the backing of a stable, known organization(s). The source code is provided to encourage openness and collective understanding, but changes and modifications are controlled (to varying degrees), thus eliminating some of the attack avenues seen with FOSS.

Given these general distinctions between FOSS and OSS, this response focuses on the role of FOSS and the overall management of quality in FOSS as a critical national security risk that must be prioritized by public and private sectors alike. Software runs mission-essential functions across the United States and other countries. FOSS is incorporated into that software and then licensed to Critical Infrastructure providers, federal and state governments, Fortune 500 companies, and other pillars of societal function. Several studies have estimated that more than 90% of delivered software products contain FOSS.

To date, quality control of FOSS is modest or nonexistent. The result is that users of FOSS have no reasonable assurance that the software they use for their most critical missions is free of basic defects, much less high-consequence flaws. The disconnect between the criticality of the missions that software executes—from transferring funds to controlling the energy grid to managing Personal Health Information—and the confidence such FOSS warrants is profound.

MITRE shares the following observations and recommendations in the public interest.

Strengthening the Software Supply Chain

The development and delivery (i.e., the supply chain) of FOSS has become a known launching ground for exploits. The lack of controls, workable mechanisms for visibility, and protections—and the general non-prioritization of security in this portion of the lifecycle—has created a relatively unmonitored and untraceable avenue for many different types of attacks. More attention is needed to ensure that the FOSS being developed is not manipulated by adversaries and is delivered as expected to the end users.

FOSS developers are vast, distributed, and self-directed; changing the FOSS ecosystem will take tremendous investment over a very long period and might or might not yield any measurable return on the investment. It could also result in harm to overall global innovation. Instead, changing risks through supply chain management needs to come in the form of multiple

interventions at key leverage points. Effort is needed to design tools to enable secure, privacy-preserving security attestations from software vendors, including their suppliers and open-source software maintainers, while also making it easy to describe the assemblage of software delivered. The integration of security checks, attestations about what practices have been followed, and listings about what has been included in the software needs to become the standard expectation for those who deliver software to users. This will require standardized data formats and interoperability. From a MITRE survey involving 19 federal agency participants, the majority indicated that it was important for a "vendor-neutral" entity to host security attestation information. Further, choosing a vendor-neutral data format for security attestation and vulnerability information was preferred while formats for describing what is in the software are maturing and ready for use.[6]

Members of the open-source software community utilize integrity checks and controls on who can publish changes to software in trusted repositories; however, these practices are not uniform. Policies are needed to increase awareness and use of cryptographic signatures, hashes, and trusted distribution systems without restricting them to specific implementations.

Additionally, the detection and mitigation of software vulnerabilities can be performed at multiple points in the software supply chain. From a theoretical perspective, it would be best to identify and mitigate software risk throughout the development lifecycle, via an efficient engagement at multiple points in that lifecycle. Practically speaking, resources are not available to engage with all relevant FOSS projects for purposes of continuous risk identification and remediation. As a result, those who use, leverage, or incorporate FOSS need to vet it for security risk prior to use, leverage, or incorporation.

To be clear: from an ecosystem structure perspective, the least-cost avoider for both FOSS and end-user delivery appears to be the software developer(s) using, leveraging, and/or incorporating FOSS into software for delivery to users. FOSS developers have vast, distributed, fluid presence and unique incentives; end users have little or no control over development and quality or security management.

A concrete example may be seen in the aforementioned study of 19 federal agencies: Open-source software vulnerabilities were identified but could not be directly resolved by the respondents as the vulnerable software was written and the code controlled by third parties. Participants used information about the vulnerabilities to determine appropriate responses that were within their control (e.g., contacting the software provider that used vulnerable open-source software or taking other measures to limit the impact of the vulnerabilities), but this individualized approach places a resource burden on end users who have no guarantee of effective results. This may address a vulnerability for a particular use case but does not guarantee the underlying code is fixed for subsequent or future uses.

Defending FOSS Maintainers from Undue Assignment of Liability

The FOSS ecosystem is both international and overwhelmingly volunteer based. This volunteer nature raises well-known challenges with sustaining FOSS activity over time. Without funding, even critical projects may be left with limited ability to respond to vulnerabilities, produce

[6] Software Bill of Materials. 2023. National Telecommunications and Information Administration, https://www.ntia.gov/page/software-bill-materials. Last accessed November 1, 2023.

essential new features, or address technical debt, which induces high defect rates and production of new vulnerabilities. The OpenSSL project, as one example, was severely under-supported when the Heartbleed vulnerability was discovered, even though OpenSSL's software itself was integrated in an enormous number of systems and used by many highly successful for-profit companies. To address these issues, many FOSS maintainers may accept donations via Patreon, GitHub Sponsors, or similar platforms, to help defray the costs of building and maintaining the software they publish. At the same time, these donations generally fall far below any reasonable hourly compensation equivalent and should not be viewed as comparable to a sustainable security budget.

The European Union's (EU) draft Cyber Resilience Act (CRA)[7] is intended to improve the state of cyber security in the EU. It includes language that assigns some degree of liability for vulnerabilities found in FOSS that is deployed by third parties to the maintainers of those open-source packages. This assignment of liability is misplaced and has the potential to severely chill participation in open-source, or to result in attempts to geographically limit use of FOSS to exclude the EU (with the persistent risk of litigation by EU entities regardless). Given the international nature of FOSS, which is published on the Internet and under most common licenses is available to any user, this assignment of liability in the EU creates some risk of reducing FOSS generation, and/or incentivizing an increase in intentional software-producer obfuscation to avoid liability, which would impact the U.S. and the immense economic benefit we have enjoyed from the reuse of this software commons.

This assignment of liability hinges on the CRA's definition of "commercial" activity as it applies to FOSS maintainers and projects. On the one hand, there are corporate-funded open-source projects, whose maintainers are full-time employees of a corporate entity that funds and sustains the project. On the other there are volunteers who may receive some minimal compensation in the form of donations, or who may offer commercially licensed extensions or support in addition to the open-source software they publish. Crucially, the final language in the CRA must be drafted to exclude from liability those volunteer individuals who receive only minimal compensation from donations or similar sources, at least as far as their activities involve solely publication of FOSS into the commons.

The U.S. government should work with and lobby the European Union to modify the current draft language of the CRA to ensure these developers are protected from liability. If we fail to do so, and the CRA is passed with the current language, then the negative effects on the U.S. and global FOSS ecosystem may be substantial.

Incentives for Commercial Producers of Software that Embed OSS Components

The use of FOSS components in commercial applications is an important consideration both to the risk of using such commercial software and to the incentives to improving the security posture of these open-source components. FOSS components make up a significant portion of the code base among commercial applications. For example, related studies released in 2020[8] and

[7] Cyber Resilience Act. 2022. European Commission, https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act. Last accessed November 1, 2023.

[8] Synopsys Study Shows 91% of Commercial Applications Contain Outdated or Abandoned Open Source Components. 2020. Security Magazine, https://www.securitymagazine.com/articles/92368-synopsys-study-shows-91-of-commercial-applications-contain-outdated-or-abandoned-open-source-components. Last accessed November 1, 2023.

2023[9] by Synopsis show 91% of commercial applications contain out-of-date or abandoned open-source components. Similarly, Gitnux reports 96% of applications have at least one open-source component and makes up over 80% of the code in use in modern applications.[10]

With commercial software producers leveraging FOSS as they do, they have a responsibility to contribute to the maintenance of the open-source code base. This responsibility is in line with their overall responsibility to their customers for security and addressing defects in their products and software-based services. Organizations that consume such software are in a position to incentivize commercial software to contribute to improving the FOSS code they use.

We recommend organizations that consume FOSS develop policies, procedures, and ways to verify and document the risk inherent in using such software. Measurement and management can begin as early as the acquisition phase and continue through acceptance and ongoing operational use of these software and software-based services. This creates commercial incentives for the software producers to actively contribute to the maintenance and secure coding of the FOSS components they choose to embed in the software they sell.

Supporting a Transition of Critical FOSS to Memory Safe Languages

Memory safe languages eliminate entire classes of vulnerabilities by providing language-level guarantees about program behavior. While users of memory unsafe languages may require additional code analysis or human review to ensure that memory safety requirements are met, in memory safe languages this guarantee is provided automatically by the language itself. This is especially important given that memory safety-related vulnerabilities remain prevalent in major systems and are generally critical in their severity, enabling denial of service attacks, remote code execution, and more when exploited.

Memory unsafe languages have historically been required in certain contexts—like embedded software, where the performance requirements or resource constraints of the system precluded memory safe languages—but those challenges have been addressed. Newer languages, including Rust, Swift, Zig, and Go, provide varying degrees of memory safety while still providing the efficiency and performance needed to be used in contexts where memory safety was not previously feasible.

The U.S. government must support and encourage the transition to memory safe languages. This can be accomplished by incentivizing transitions to memory safe languages, and by reducing or eliminating barriers to transitioning to memory safe languages.

Barriers to transition may include a lack of software developer training or experience in memory safe languages, or the unavailability of specific functionalities or open-source libraries in those languages. The U.S. government can, through partnership with existing open-source supporting foundations like the Linux Foundation, Rust Foundation, Eclipse Foundation, and others, work to develop and disseminate training materials in memory safe languages, and to provide funding directed toward developing critical mission software in those languages.

---

[9] Open Source Security and Risk Analysis Report. 2023. Synopsys, https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2023.pdf. Last accessed November 1, 2023.

[10] Open Source Software Statistics. 2023. Gitnux, https://blog.gitnux.com/open-source-software-statistics/. Last accessed November 1, 2023.

In the specific case of FOSS software, supporting transitions to new languages will require organizations funding developers to implement a transition over time. To use the Curl FOSS project as a case study, the project's lead maintainer, Daniel Stenberg, recently had this to say regarding a transition of Curl (which is itself widely deployed and used, and entirely open-source) to the Rust programming language (a memory safe language):

> *Yes, this family of flaws would have been impossible if curl had been written in a memory-safe language instead of C, but porting curl to another language is not on the agenda. I am sure the news about this vulnerability will trigger a new flood of questions about and calls for that and I can sigh, roll my eyes and try to answer this again.*
>
> *The only approach in that direction I consider viable and sensible is to:*
>
> 1. *allow, use and support more dependencies written in memory-safe languages and*
> 2. *potentially and gradually replace parts of curl piecemeal, like with the introduction of hyper.*
>
> *Such development is however [sic] currently happening in a near glacial speed and shows with painful clarity the challenges involved. curl will remain written in C for the foreseeable future.*
>
> *Everyone not happy about this are [sic] of course welcome to roll up their sleeves and get working.*
>
> – *Daniel Stenberg*[11]

As shown in this quote, although Stenberg is not opposed to transitioning the Curl implementation to be written in Rust, his experience has been that such a transition is slow without the resources to sustain it and can only be successful with the investment of interested third parties.

It's also worthwhile to note that rewriting existing code itself carries the possibility of introducing new defects that may be exploitable vulnerabilities. If existing code is already highly assured through other means, the value of a complete rewrite in a memory safe language may be more limited than expected. Decisions on whether rewrites are worth undertaking should be made on a case-by-case basis.

This raises an additional point that must be understood about identifying opportunities for transition to memory safe languages: not all code is a high priority to transition, and the focus must be on critical code. To paraphrase Kelly Shortridge, a senior principal engineer at Fastly, the most critical code to secure can be identified at the intersection of three attributes: (1) code processing untrusted inputs, (2) code written in a memory unsafe language, and (3) code that runs without a sandbox.[12] Code for which all three of these things are true ought to be considered the highest priority for being rewritten in a memory safe language, ideally with the addition of other assurance techniques, including sandboxing, thorough testing, and automated fuzzing.

---

[11] D. Stenberg. How I Made a Heap Overflow in Curl. 2023. Daniel Stenberg, https://daniel.haxx.se/blog/2023/10/11/how-i-made-a-heap-overflow-in-curl/. Last accessed November 1, 2023.

[12] K. Shortridge. The SUX Rule for Safer Code. 2023. kellyshortridge.com, https://kellyshortridge.com/blog/posts/the-sux-rule-for-safer-code/. Last accessed November 1, 2023.

Finally, FOSS projects are generally volunteer-based, participatory projects. What happens in a FOSS project is often heavily influenced by social and individual attributes and attitudes that are not usually influenced by money or a threat of non-adoption. FOSS projects often use the language(s) that their creator(s) and contributors are most comfortable with; transitioning languages in the open-source context may take a project out of the knowledge space of a maintainer, and thus be a difficult sell for that maintainer to undertake. Forking projects (creating your own duplicate of the code that you then maintain and modify) or introducing new projects with different language choices or goals, are always valid options.

Questions Posed in the RFI

1. Which of the potential areas and sub-areas of focus described (in the RFI) should be prioritized for any potential action? Please describe specific policy solutions and estimated budget and timeline required for implementation.

The following areas presented in the Request for Information (RFI) should be prioritized:

- *Behavioral and Economic Incentives to Secure the Open-Source Software Ecosystem*
- *Strengthening the software supply chain*
- *Fostering the adoption of memory safe programming languages*

We believe that focusing on these areas can lead to significant progress. More important, there are tangible tasks that can be accomplished in the short term to advance these areas. As described in recommendations section above, the community should:

- Facilitate the integration of security attestations across the software industry
- Safeguard FOSS developers from liability
- Create incentives for commercial software vendors to contribute to the maintenance of FOSS
- Develop and disseminate training materials in memory safe languages
- Support the development of critical mission software in memory safe languages

Implementing these solutions will require time and resources. It is essential to distribute this responsibility among all parties within the ecosystem, rather than placing the burden solely on the shoulders of the resource-limited FOSS projects.

2. What areas of focus are the most time-sensitive or should be developed first?

The most time-sensitive area is *Behavioral and Economic Incentives to Secure the Open-Source Software Ecosystem*.

Addressing this area is crucial, as it directly impacts the security and sustainability of the FOSS ecosystem. More information can be found in the Introduction section of this response.

We must find ways to meet this objective, such as the EU's draft CRA, but without penalizing those volunteer individuals who receive only minimal compensation for their activities that solely involve publication of FOSS into the commons.

3. What technical, policy or economic challenges must the Government consider when implementing these solutions?

A key consideration for this policy endeavor is to ensure that it does not hinder the collaborative development model or diminish the benefits of sharing open-source software. Imposing excessive responsibilities or liabilities on developers could potentially drive them toward adopting a closed-source model instead.

One significant challenge lies in identifying the most effective approach to incentivizing commercial vendors for their support of FOSS. This should be accomplished in a manner that exclusively rewards those vendors that genuinely shoulder the burden of FOSS support, without inadvertently benefiting those that do not contribute meaningfully.

Additionally, it is crucial to strike a balance between enhancing security measures and maintaining the flexibility and innovation inherent in the open-source community. Over-regulation could stifle creativity and impede the rapid development that characterizes FOSS. Therefore, the government should explore ways to foster an environment that encourages innovation while also ensuring the security and sustainability of the FOSS ecosystem. Striking the right balance between security, innovation, and incentives will be essential in preserving the essence of FOSS while advancing its security and reliability.

4. Which of the potential areas and sub-areas of focus described (in the RFI) should be applied to other domains? How might your policy solutions differ?

The policies implemented under the initiatives "Strengthening the Software Supply Chain" and "Fostering the Adoption of Memory Safe Programming Languages" hold relevance for both commercial off-the-shelf and government off-the-shelf acquired software. These initiatives emphasize a comprehensive approach to enhancing software security, transcending the boundaries of open-source software.

As such, these policies should be regarded as valuable tools for promoting security across the entire software spectrum. By applying these principles to all types of software, including proprietary and government-developed solutions, a more robust and secure software ecosystem can be achieved. This holistic approach ensures that the benefits of strengthened supply chains and memory safe programming languages are extended to the broader software landscape, fostering a more secure digital environment for all stakeholders.