Response to Request for Information:

Open-Source Software Security:

Areas of Long-Term Focus and Prioritization

88 Fed. Reg. 54315 (Aug 10, 2023)

Docket No. ONCD-2023-0002

November 8, 2023

Google applauds the Biden Administration's continued and focused effort on addressing the challenges of open source security. As a producer and consumer of open source software, as well as a contributor to key software security initiatives, we welcome the opportunity to provide comments in response to The Office of the National Cyber Director's (ONCD) and other partner agencies' Request for Information[1] on areas of long-term focus and prioritization on open-source software security.

Google is a leader both in contributing to open source projects and to the security of open source. A vibrant and secure open source community, which is at the core of Google's infrastructure, processes, and culture, is a key differentiator in technology innovation. We believe that by being open and freely available, open source software (OSS) enables collaboration and the development of technology, solving real-world problems.

Similarly, open source software is a critical part of the digital infrastructure that has helped propel the U.S. software sector to its global leadership position. As one of the largest consumers of open source, the US federal government occupies a unique position of leverage to improve the security of OSS ecosystems. By working to develop, support, and adopt tools and resources that can be used by all software professionals to secure their use of OSS, the federal government can act as a beacon for the entire software industry.

[1] https://www.regulations.gov/document/ONCD-2023-0002-0001

Table of Contents

Recommended focus areas

We believe that federal efforts in the following five areas will be the most effective:

1. **Sustain open source software communities and critical infrastructure.** Open source software and related services underpin most modern software technology, yet they are largely maintained by volunteer communities and resource-constrained non-profits. The federal government should actively support and sustain open source software.
2. **Accelerate the transition to memory safety.** Widespread use of memory-safe programming languages will eliminate an entire class of security vulnerabilities. The federal government should champion the transition to memory-safe languages and frameworks and work with industry to set standards and improve the security of legacy codebases.
3. **Facilitate efforts to standardize and strengthen the software supply chain.** The federal government should actively establish, standardize, and incentivize new best practices for OSS use across industries and help establish a standard framework for OSS usage in software systems.
4. **Spur innovation by incentivizing novel research.** Even with significant recent progress in the areas of open source cybersecurity, there is still much room for innovation. The federal government should invest in research and provide rewards for novel discoveries.
5. **Create educational materials and curricula.** To bridge the gap in developer education during the transition to secure-by-default systems, the government should support the creation and adoption of curricula that teach secure software best practices.

Recommendations on securing open source software

We provide the following recommendations toward each of the stated focus areas.

1. SUSTAIN OPEN SOURCE SOFTWARE COMMUNITIES AND CRITICAL INFRASTRUCTURE

It is important to ensure that the government is using OSS in a secure, responsible manner across all agencies, that the government's interests in the open source community are represented, and that the government is able to support and sustain open source software ecosystems.

A. Supporting adoption of security tools and best practices

The Google Open Source Programs Office (OSPO) is one of the first OSPOs in the industry.[2] Our OSPO is focused on enabling Google to build on open source technologies, share Google-developed technology under open licenses, and support open source projects, communities, and maintainers across the entire open source ecosystem. As a result, as of 2021, more than 10% of all Google full-time employees contribute to open source.[3]

A federal OSPO, as is proposed in CISA's Open Source Software Security Roadmap,[4] would bring similar benefits by managing federal agencies' consumption of open source and contributions to it. This office would set policies for the government's use of open source software, including assessing

[2] https://opensource.google/about
[3] https://opensource.googleblog.com/2022/06/establishing-new-baselines-identifying-open-source-work-in-an-unstable-world.html
[4] https://www.cisa.gov/sites/default/files/2023-09/CISA-Open-Source-Software-Security-Roadmap-508c.pdf

and managing security risks, promoting best practices, and fostering collaboration between government agencies.

This OSPO would also coordinate and encourage government participation in the open source community: defining policies for how government employees can release and contribute to open source; teaching federal employees best practices of upstream participation; and advising them on how to advocate for their needs as open source users.

There are many opportunities for a federal OSPO to improve the security of the open source ecosystem. Google, for example, created the Open Source Security Upstream Team,[5] who spend 100% of their time making security contributions to critical, but under-resourced open source projects. A federal OSPO could organize a similar team to identify at-risk projects and empower federal employees to participate in open source security activities on a full- or part-time basis.

B. Supporting open source software foundations directly

Open source software is a critical element of US digital infrastructure that needs financial support through new and existing public funding sources. The federal government should partner with existing OSS foundations such as the Rust Foundation,[6] Python Software Foundation,[7] Eclipse Foundation,[8] Linux Foundation,[9] Open Source Security Foundation (OpenSSF),[10] and many others with expertise in this space. The federal government can leverage their collective knowledge and relationships to most effectively direct funding and support to key projects.

The federal government should consider expanding the availability of public funding programs for open source technologies, similar to grant programs such as the OpenSSF's Alpha-Omega Program,[11] US AGM's Open Tech Fund,[12] Mozilla's Open Source Support Awards,[13] and the German government's Sovereign Tech Fund.[14] The National Science Foundation's Pathways to Enable Open Source Ecosystems (POSE)[15] program is a promising experiment in direct public funding of open source innovation, and we applaud its mission to foster wholly new open source foundations and ecosystems. POSE would be most effective if paired with a funding vehicle in the form of an "Open Source Tech Fund," providing financial support for US and international organizations that maintain key open source projects. This support will help ensure the security and sustainability of critical, widely used and free public services like software repositories.

Further, we highly recommend that U.S. sector risk management agencies (SRMAs) and critical infrastructure operators play a more active role in open source software security by joining and contributing to organizations like OpenSSF and industry-led forums, such as FinOS[16] in the financial sector. These organizations play a critical role in sharing knowledge and best practices and in developing common open standards to manage risks.

[5] https://opensource.googleblog.com/2023/04/googles-open-source-security-upstream-team-one-year-later.html
[6] https://foundation.rust-lang.org/
[7] https://www.python.org/psf-landing/
[8] https://www.eclipse.org/org/foundation/
[9] https://linuxfoundation.org/
[10] https://openssf.org/
[11] https://openssf.org/community/alpha-omega/
[12] https://www.opentech.fund/
[13] https://www.mozilla.org/en-US/moss/
[14] https://sovereigntechfund.de/en/
[15] https://new.nsf.gov/funding/opportunities/pathways-enable-open-source-ecosystems-pose
[16] https://www.finos.org/

2. ACCELERATE THE TRANSITION TO MEMORY SAFETY

We strongly support prioritizing solutions to the challenges posed by memory safety, given the large occurrence of memory corruption vulnerabilities in open source compromises. Memory bugs in memory unsafe languages are too pervasive for reactive approaches to work. As a result, we see many memory safety vulnerabilities being exploited in-the-wild, despite state-of-art development practices, fuzz testing and vulnerability management. Consequently, we strongly believe that the bar going forward should be "safe by default."

To achieve this, federal agencies should widely promote programming languages, such as Rust and Go, that default to memory-safe development and safe coding patterns for use in applications where memory-unsafe languages would have previously been preferred. Memory-safe languages embody safe-by-default, since memory safety is inherently tied to the programming language used to write applications and services. New development should prioritize memory-safe languages. We expect that rewriting large, existing unsafe codebases will often be impractical. and recommend that old, unsafe codebases be updated gradually via interoperability or by enforcement of safer coding patterns rather than entirely rewritten. Furthermore, OS3I should set and track metrics for measuring memory-safe component usage broadly across the open source ecosystem.

A. Enable incremental adoption through ergonomic and safe cross-language interoperability

An analysis of Android[17] and other internal Google codebases indicates that most of our memory bugs occur in new or recently modified code (50% being less than a year old), and that most vulnerabilities are memory safety bugs (90% of Android vulnerabilities).[18] Our results in Android suggest that prioritizing new code development in memory-safe languages is both an efficient and effective strategy for increasing resiliency.[19]

The federal government should incentivize and support efforts which enable the incremental adoption of memory-safe languages in memory-unsafe code bases, rather than comprehensive rewrites. In particular, we believe that safe and ergonomic interoperability is key to incremental adoption, especially when it comes to C/C++ and potential successor languages (e.g., Rust): memory-safe systems can leverage memory-unsafe ecosystems during the transition, and OSS consumers can start using memory-safe libraries in existing memory-unsafe codebases.

Consequently, Google has invested in better understanding interoperability requirements and has created tooling to meet those requirements for Rust. See the Appendix for examples of our efforts (A) and specific recommendations for next steps in this area (B).

B. Measure and track OSS memory safety posture

Since the majority of severe vulnerabilities in large C and C++ code bases are from memory-safety bugs, including 80% of zero-days being exploited in the wild,[20] to assess a project's security posture we need to measure whether it uses memory-unsafe components, and if so, understand how the components are used.

[17] https://security.googleblog.com/2021/04/rust-in-android-platform.html
[18] https://security.googleblog.com/2019/05/queue-hardening-enhancements.html
[19] https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html
[20] https://www.memorysafety.org/docs/memory-safety/

Consequently, we recommend creating a way to evaluate: 1) whether a given code base relies on memory-unsafe components, 2) whether it uses tools to detect or mitigate memory safety issues, and 3) its progress in the transition to memory safety. This is especially important in mixed code bases utilizing interoperability, where the degree of memory safety falls on a spectrum.

Similarly, metrics should be captured about safe vs. unsafe usage of memory-safe languages. In our experience, unsafe API and code usage are sometimes necessary, but should be rare, and expert review is needed in these exceptional cases to catch misuses. Since "mandated expert review" does not translate to OSS ecosystems, we suggest measuring the prevalence of unsafe coding patterns in otherwise memory-safe projects and providing financial incentives towards improvement.

Those metrics could be integrated into existing sources of open source security information, such as deps.dev.[21] These metrics will enable a ratchet effect[22] to prevent regressions on improvements. By tracking and enforcing incremental improvements as they roll out, it's possible to avoid unending well-intended work to adopt and then re-adopt them without ever reaching an end state.

C. Incentivize expansion of a memory-safe OSS ecosystem

We recommend incentivizing or investing in the development of a memory-safe open-source ecosystem—an ecosystem of reusable components that support new development in memory-safe languages. This requires listening to OSS developers: what gaps are preventing them from adopting memory safe languages? Once the missing parts of the ecosystem are identified, ensure they are built so they can be reused widely. A mandate to rewrite all open source software in memory-safe languages is not a practical strategy in the short or medium term, but focusing on several specific areas will pay dividends in the long run by enabling more developers to adopt memory-safe languages, which will help spur development of the memory-safety ecosystem even further.

We suggest investment the following areas:

- Commonly used libraries that do not have acceptable memory-safe equivalents. This could include rewriting the library in a memory-safe language, or improving a pre-existing memory-safe library that is underused due to missing features or substandard performance;
- Libraries that are commonly used across the industry. While it may not be cost-effective for any one entity to fund a rewrite themselves, the ROI increases if the cost is shared across the industry through, for instance, a foundation like OpenSSF or ISRG;
- Code used in critical infrastructure where high assurance is particularly needed. This would be a continuation of the work by ISRG.[23]
- New codebases where there is a choice between using memory-safe and memory-unsafe languages, such as writing new Linux hardware drivers in Rust.

Just creating these memory-safe replacements for commonly-used and/or commonly-exploited libraries is not enough to drive adoption. Any replacements must come with evidence to show they are at par or superior to the libraries they intend to replace. The government can support building a body of evidence, which could include:

[21] https://deps.dev/
[22] https://en.wikipedia.org/wiki/Ratchet_effect
[23] https://www.memorysafety.org/docs/memory-safety/

- standardized benchmarks, to demonstrate comparable performance;
- test and conformance suites, to verifiably demonstrate compatibility;
- testimonials from relying parties, like cargo-vet, to demonstrated expert review;
- evidence of active maintainership or corporate sponsorship to demonstrate support.

This evidence can help advocate for use of these libraries and bring memory-safety benefits to the users that depend on them.

D. Publish a federal memory-safety roadmap

We recommend the Government, led by OS3I, adopt and publish a roadmap for their transition to memory safety, including the following priorities:

- Update policies and procurement guidelines to ensure memory-safe languages are the default for new government software;
- Examine whether relevant software security standards and frameworks should be updated to prioritize memory safety;
- Invest in interoperability tools that support incremental adoption, not large rewrites;
- Prioritize partial rewrites of memory-unsafe components that cannot be isolated;
- Invest in creating and sharing new memory-safe equivalents for those components; and
- Create milestones based on, for example, percentage of memory-safe lines or components.

By replacing components one-by-one, security improvements are delivered continuously instead of all at once at the end of a long rewrite. A full rewrite may eventually be achieved with this incremental strategy, but without the risks typically associated with those large rewrites.

3. FACILITATE EFFORTS TO STANDARDIZE AND STRENGTHEN THE SOFTWARE SUPPLY CHAIN

The federal government should play a critical role in defining cybersecurity requirements that take these standards and best practices into account during procurement, ensuring that vendors are providing components that meet industry standards for security and that are compatible with existing OSS tools and frameworks. By doing so, the federal government can help to establish and foster an overall culture of responsible OSS use in software systems, and ensure that related critical software systems are protected from future cyber threats.

A. Support transparency and integrity with modern software signing techniques

The government should promote the adoption of modern software signing techniques that protect against compromise with increased transparency. Software signing can ensure the integrity of software updates, downloads from package repositories, supply chain attestations, and more. Google has helped create and maintain Sigstore,[24] a modern software-signing service that makes it easy to create software signatures, without the complexities of key management.

Sigstore's novel techniques use ephemeral keys, transparency logs, and individual- or workload-based identities to create publicly auditable signatures. Each signing event is published in a public, append-only ledger, allowing systems to verify the authenticity and integrity of software

[24] https://www.sigstore.dev/

artifacts. The ephemeral keys expire shortly after use, which protects against key compromise, and individuals can prevent identity compromise by detecting unauthorized signing events in the transparency log.

Google supports Sigstore as a public good service and we promote its use based on our own experience using similar techniques with transparency logs[25] to protect Android, Chrome,[26] and Pixel firmware.[27] Signing techniques based on transparency logs become more secure when they are more widely used, as they increase the speed at which compromise can be detected, and help incentivize good behavior. We suggest the government help support widespread adoption of transparency-backed modern signing techniques by promoting and using solutions such as Sigstore, as well as supporting their integration into widely used open source software ecosystems.

B. Support verifiability and trust in software with provenance generation

The government should promote verifiability across the software supply chain. A key aspect of software verifiability is the production, availability, and consumption of attestations such as provenance. Provenance for OSS provides knowledge about where pieces of system software came from and how they were built, in a verifiable way. This helps software owners track the source of software components and verify they were created in accordance with set policies.

Google supports Supply-chain Levels for Software Artifacts[28] (SLSA) as a standard for evaluating and describing how securely software was built. SLSA provides incrementally adoptable guidelines for improving supply chain security. Over the last decade, Google has successfully used an internal version of SLSA to protect against insider risk, build system tampering, and unilateral code changes.

By promoting SLSA as a national standard for software provenance, the government can raise the bar for supply chain security standards. Support for SLSA needs to be built into open source ecosystems,[29] including SLSA-compliant builders, tooling for provenance production, and automatic verification solutions. Integrating SLSA tooling into core ecosystem tooling allows for both the generation of signed provenance at time of production, and the verification of that provenance at time of consumption, by default. Open source maintainers should not have to shoulder the burden of adopting SLSA; it should become part of the default fabric of the software ecosystems they already use for development.

C. Support vulnerability awareness, remediation and dependency insights tooling

Transparency enables insights into whether vulnerabilities actually affect software in use and potential paths to remediation. The government should adopt and promote vulnerability tooling and dependency mapping services that focus on open source and provide ways to automate remediation at scale.

[25] https://transparency.dev/
[26] https://chromium.googlesource.com/chromium/src/+/master/net/docs/certificate-transparency.md
[27] https://security.googleblog.com/2023/08/pixel-binary-transparency-verifiable.html
[28] https://slsa.dev/
[29] https://security.googleblog.com/2023/04/celebrating-slsa-v10-securing-software.html

Google collaborated with the OpenSSF to create the Open Source Vulnerabilities (OSV) Schema,[30] as well as OSV.dev[31] and OSV-Scanner[32] to simplify vulnerability detection and remediation for both open source developers and consumers. Combined, they help users triage a large number of known vulnerabilities and provide a foundation for automated vulnerability remediation, simplifying the sometimes complex process of determining how to update affected dependencies.

To understand affected dependencies, Google uses information from our deps.dev service,[33] a free, public service that allows organizations to understand all of their dependencies, including information about ever-changing dependency graphs, security best practices, known vulnerabilities, known malicious packages,[34] and licenses, all available via an API.

We believe the government should both use and promote the adoption of both the OSV and deps.dev services. Adoption of deps.dev would increase visibility into open source dependencies, both for ongoing monitoring and maintenance and to support responses during incidents (such as querying whether an organization was affected by the Log4Shell vulnerability,[35] and then planning the appropriate dependency updates to remove the vulnerability).

In addition to adoption of OSV as a comprehensive vulnerability database, the government should also further promote closer integration of OSV with the National Vulnerability Database (NVD), specifically encouraging NVD to adopt machine-readable naming/versioning schemes compatible with OSV (e.g. PURLs[36]) to replace CPEs, as well as using OSV as a reference data source for populating various fields (e.g. package names, impacted versions, etc).

D. Partner with critical infrastructure providers to identify and support sector-specific dependencies

Following Log4Shell, companies like Google, Microsoft, and Amazon collaborated to fund Alpha-Omega, a grant program aimed at supporting the world's most critical open source projects. We urge the federal government and SRMAs to fund and coordinate a similar exercise to support the most critical open source dependencies on a sector-by-sector basis. Infrastructure operators in sectors like water, energy, or manufacturing may share sector-specific dependencies on OSS projects, which could make for attractive exploitation targets for sophisticated cyber actors.

To date much of the federal government's efforts to defend critical infrastructure has focused on sharing threat intelligence, identifying risks, and initiating joint response efforts. "Shifting left" by directing resources to securing supply chains further upstream can help to mitigate the risks posed by widespread disruptions from cyber incidents, and strengthen the resilience of the U.S. economy.

E. Explore opportunities to leverage curated open source tools from trusted sources

It may be in the government's interest to consume open source libraries that are curated by a trusted provider. For example, Google Cloud offers a service called Assured Open Source Software, which redistributes more than 2,500 Java and Python libraries, and adds vulnerability testing,

[30] https://ossf.github.io/osv-schema/
[31] https://osv.dev/
[32] https://github.com/google/osv-scanner
[33] https://deps.dev/
[34] https://github.com/ossf/malicious-packages
[35] https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf
[36] https://github.com/package-url/purl-spec

fuzzing, and signing to verify integrity. Curation tools like Assured Open Source Software can help simplify compliance and achieve a high degree of assurance when using common OSS packages.

4. SPUR INNOVATION BY INCENTIVIZING NOVEL RESEARCH

There are many areas of open source security where we simply don't know enough to understand if a given solution or improvement would be possible or effective. By incentivizing research, we can fill knowledge gaps and help to advance the state of the art in security and safety.

A. Research techniques for automatic translation from memory unsafe to memory safe code

The government should support research into translation tools, such as c2rust,[37] that assist in automatically translating C programs to Rust. Even if this type of tool cannot generate idiomatic Rust for all code, since aspects of Rust translation require a human-guided redesign, it does reduce the manual effort required for such a translation, and it prevents some of the bugs caused by humans. Translation may also begin with changes to the C++ codebase to make object lifetimes more compatible with e.g. Rust's borrow checker. To this end, Google is exploring how to annotate C++ object lifetimes in Clang.[38]

That said, syntactic transformation is only the first step. The overall goal would be to convert C and C++ code into equivalent Rust code that is idiomatic and maintainable, but this is far beyond the capabilities of the translation assistance tools available today. It is possible that certain parts of code might need to undergo a redesign while being translated to Rust, and although generative AI might help with this in the future, it likely will still require some human input.

B. Research ways to lower the risk posed by necessary use of memory-unsafe code

The federal government should take a realistic approach that automatic translation tools have limits, and that not all existing, legacy memory-unsafe systems will be portable to memory-safe languages. Some of the most critical components of open source software (such as operating systems and device drivers) cannot be entirely written in high level languages at all. Furthermore, given that most bugs occur in newly written code, it should not be recommended to attempt to perform large, full-stack rewrites of stable memory-unsafe systems to memory-safe languages.

To protect this necessary memory-unsafe code, invest in research to further improve state-of-the-art fuzz testing ("fuzzing") technology, an automated testing technique primarily useful for detecting memory corruption vulnerabilities in C/C++ code. Recently, fuzzing has seen new advances, and our OSS-Fuzz service[39] can now catch broader classes of vulnerabilities beyond memory safety, including remote execution vulnerabilities[40] such as Log4Shell.[41]

Research should be done into expanding fuzz testing at scale, to detect even more types of vulnerabilities in additional languages across more open source libraries. For example, Google has used AI to automatically generate fuzzing targets,[42] which increases the scale of fuzzing tools,

[37] https://github.com/immunant/c2rust
[38] https://discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377
[39] https://github.com/google/oss-fuzz
[40] https://security.googleblog.com/2022/09/fuzzing-beyond-memory-corruption.html
[41] https://security.googleblog.com/2021/12/improving-oss-fuzz-and-jazzer-to-catch.html
[42] https://security.googleblog.com/2023/08/ai-powered-fuzzing-breaking-bug-hunting.html

reduces the barrier to adoption of fuzzing tools, and increases fuzzing coverage across critical legacy systems that are written in memory-unsafe languages.

Another avenue of research addresses protecting environments where memory-unsafe code runs. Some potential solutions are sandbox-by-default,[43] enclaves,[44] formal methods,[45] and low-level architecture improvements such as MTE[46] or CHERI,[47] which is a key component of the UK's Semiconductor Strategy.[48]

C. Make significant investments to incentivize novel security research

The government should incentivize research that expands both traditional security fields, such as vulnerability detection, and new areas, such as the applications of AI in cybersecurity.

Similar to Google's Open Source Software Vulnerability Rewards Program,[49] the government should offer financial incentives to the diverse community of academic and security researchers who find, responsibly report, and fix vulnerabilities in critical open source projects.

The government should also organize programs aimed at driving innovation in software security and creating a new generation of security tools, similar to DARPA's New AI Cyber Challenge.[50]

D. Research techniques to improve the safety of memory-unsafe languages

While we believe that a transition to memory-safe languages is the correct long-term solution, we expect to have a large body of memory-unsafe code likely for decades to come. This code can still be made safer—or in some instances completely safe—to have short-term impact.

We recognize that decades of attempts at making C and C++ safe have not succeeded. We suggest a different approach, which focuses on denying attackers those primitives in the first place, through the elimination of classes of vulnerabilities. Specifically, we believe that initialization, spatial,[51] and type[52] safety may be within reach for C++. Complete temporal safety may become possible with appropriate hardware support,[53] and several major improvements are already available to use.[54]

The government should incentivize research into the feasibility of such improvements for memory-unsafe languages. Additionally, the government should also incentivize language steering committees to make safety a priority.

E. Support applied research to enumerate and restrict capabilities of software components.

We encourage federal support for research into capability enumeration and restriction. Restricting software capabilities is an effective way to mitigate security risks. For example, if the Log4j library

[43] https://cloud.google.com/kubernetes-engine/docs/concepts/sandbox-pods
[44] https://github.com/project-oak/oak
[45] https://github.com/hacspec/hax
[46] https://source.android.com/docs/security/test/memory-safety/arm-mte
[47] https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/
[48] https://www.gov.uk/government/publications/national-semiconductor-strategy/national-semiconductor-strategy
[49] https://bughunters.google.com/open-source-security
[50] https://openssf.org/press-release/2023/08/09/openssf-to-support-darpa-on-new-ai-cyber-challenge-aixcc/
[51] https://discourse.llvm.org/t/rfc-c-buffer-hardening/65734
[52] https://i.blackhat.com/USA-22/Thursday/US-22-Bialek-CastGuard.pdf
[53] https://security.googleblog.com/2022/05/retrofitting-temporal-memory-safety-on-c.html
[54] https://google.github.io/tcmalloc/gwp-asan.html

had only write access to log files, with no other capabilities like network access or ability to define new classes, the impact of the Log4Shell vulnerability would have been significantly reduced.

Google's recently launched Capslock[55] project is one example of capability enumeration research. Capability restriction research includes sandboxing approaches and capability-based API design contracts, which allow consumers of OSS software to limit the scope of security-sensitive operations to a small and controlled subset through a well-defined system.

5. CREATE EDUCATIONAL MATERIALS AND CURRICULA

While the ultimate goal is security by default, developer education can address these gaps during the transition to safe-by-default ecosystems. For instance, developers may not know how to program in memory-safe languages; they may not know the risks posed by using software for which integrity cannot be verified; or they may not understand the risks posed by memory unsafety.

A. Invest in training to focus on software security, including memory-safe languages

Consistent with the recommendations of the Cyber Safety Review Board,[56] we recommend that the federal government invest in cybersecurity training in higher educational institutions and for federal employees focused on key software security best practices, specifically the usage of memory-safe languages and evaluation of open source libraries. We also recommend integrating safety curricula into existing training to convey the importance of safe-by-default principles and the tools and techniques that support them.

B. Invest in moving educational CS curriculums away from memory-unsafe languages

We recommend that computer science (CS) curricula move away from memory-unsafe languages like C and C++, especially in introductory classes, in operating systems and compiler classes, and in non-CS majors where only one programming language is taught. We also recommend including education on safe-by-default tools and techniques in existing CS training.

When teaching memory-unsafe languages is necessary (for example, training engineers to work on existing memory-unsafe codebases), we recommend requiring prerequisite experience with memory-safe languages, prefacing educational material with data about risks, and overall dispelling the misconception that they can ever be used safely (even by engineers with significant expertise).

Conclusion

We look forward to the next steps in the Biden Administration's efforts to address the challenges of open source security. While the problem space is large, it is tractable, with real solutions. We now have an opportunity to make a timely, sizable, and impactful improvement on the open source software ecosystem, and usher in a new, more secure era.

[55] https://github.com/google/capslock
[56] https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf

Appendix – Memory Safety

A. Google's efforts in understanding and developing cross-language interoperability

We have invested in both understanding the requirements for interoperability and creating new tooling as needed. For example:

- We have published our internal analyses on how we evaluated and determined our Rust/C++ interoperability needs for Android[57] and Chromium[58].
- We have developed and open-sourced autocxx,[59] a popular interoperability tool.
- We are also investigating more seamless C++/Rust interoperability in Crubit.[60] Apple is investing in seamless C++/Swift interoperability[61] for similar reasons.

B. Google's recommendations for future safe cross-language interoperability efforts

There remain unsolved challenges. Rust and C++ have different semantics and different requirements. However, we feel this is a promising direction, and the remaining open questions are likely either not going to hinder productive use, or are solvable with additional work and investment. In particular, we recommend:

- Investment to triage and fix bugs in the rust-bindgen project[62] (a project that generates Rust bindings for C/C++ code and supports other C++ interoperability tools) which would immediately and scalably improve C++/Rust interoperability for many users.
- Investment in the Rust language tweaks known to improve the quality and ergonomics of C++/Rust interoperability
- Extending Rust in prominent, useful open-source C++ projects to publicize the success of solving the interoperability problems, showcased through extensive blog posts and talks. We recommend two use cases: (a) a project where existing tools like cxx are sufficient, (b) a project where a more general, seamless C++/Rust interoperability is needed.
- Investment in Rust tooling to more easily enable mixed-language codebases. Existing users of C/C++ cannot easily use Rust's standard build tooling like "Cargo", and more formal support is needed for linking a mixed-language codebase with complex dependencies.

[57] https://security.googleblog.com/2021/06/rustc-interop-in-android-platform.html
[58] https://www.chromium.org/Home/chromium-security/memory-safety/rust-and-c-interoperability/
[59] https://github.com/google/autocxx
[60] https://github.com/google/crubit
[61] https://www.swift.org/documentation/cxx-interop/
[62] https://rust-lang.github.io/rust-bindgen/