THE LINUX FOUNDATION

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

WHITEPAPER

The Open Source Software Security Mobilization Plan

May 2022

Contents

Executive Summary

The modern software supply chain relies pervasively on open source software ("OSS") for both underlying components and operation. The ability for organizations (including companies and governments) to innovate faster, and at a higher level of quality, is often linked to their adoption of OSS components. Roughly 70-90% of any software "stack" consists of OSS[1],[2]. That shared benefit also comes with shared risk in the form of exposure to vulnerabilities in those OSS components.

Vulnerabilities and weaknesses in widely deployed software present systemic threats to the security and stability of modern society as government services, infrastructure providers, nonprofits and the vast majority of private businesses rely on software in order to function.

The software supply chain is complex and as susceptible to disruption and corruption as any physical supply chain. While the private sector constantly invests in protecting the software supply chain via standards, shared services, and solutions to reduce risk of both inadvertent errors and intentional attacks, just as with physical infrastructure (ports, power grids, telecommunications networks, etc) the public sector can and should play a role in hardening the systems. The public and private sectors must work together to meet the collective security and safety needs of citizens and stakeholders, facing even greater challenges in addressing these threats.

While there are considerable ongoing efforts to secure the OSS supply chain, to achieve acceptable levels of resilience and risk, a more comprehensive series of investments to shift security from a largely reactive exercise to a proactive approach is required. Our objective is to evolve the systems and processes used to ensure a higher degree of security assurance and trust in the OSS supply chain.

This paper suggests a comprehensive portfolio of 10 initiatives which can start immediately to address three fundamental goals for hardening the software supply chain. Vulnerabilities and weaknesses in widely deployed software present systemic threats to the security and stability of modern society as government services, infrastructure providers, nonprofits and the vast majority of private businesses rely on software in order to function.

With OSS we have the ability to directly and systematically mitigate the risks associated with OSS use that we do not have with proprietary software. This is due to the availability of the underlying source code, the way most development teams do their work in the open, and the significant amount of cross-project reuse of components and concepts.

1 "2020 Open Source Security and Risk Analysis Report" by Synopsys: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2020-ossra-report.pdf and "2020 State of the Software Supply Chain" by Sonatype: https://www.sonatype.com/resources/white-paper-state-of-the-software-supply-chain-2020

2 "2022 Open Source Security and Risk Analysis Report" by Synopsis: https://www.synopsys.com/software-integrity/em/ossra-report.htm

Smart investments into systematic enhancements in the way OSS is developed, recombined, distributed, and deployed, as well as into specific highly re-used "critical" pieces, would be a highly leveraged and cost-effective way to reduce the risk for all downstream users. This includes the many proprietary and custom software solutions that incorporate OSS.

All software should be assumed to have defects, and all software development teams will at one point or another inadvertently allow a defect into their code, despite best efforts. Working with the teams managing open source projects is critical to the success of any of the above objectives. The managers of OSS projects are typically called "maintainers"; they actively manage the roadmap and release cadence of an open source project, as opposed to the broader category of "contributors". Maintainers are also the first line of defense in holding off any malicious attempts to insert bad code into projects, or to protect against changes in other OSS projects that are implicitly or explicitly included in their software that may harm their users.

Thus, all forms of investment and intervention should be focused on delivering new value to OSS maintainers — from making it easier to adopt practices that enhance the security and integrity of their work, to funding activities like third party code reviews that most projects struggle to afford to perform on their own. Any investments or policies that place additional burdens on developers, increase their personal or professional liability for working on code, or issue unfunded mandates upon them, would struggle for adoption and potentially inhibit further advancements in open source software. Any successful approach must be collaborative, offering not only guidance but also education, support, and technical resources. This will be crucial in striking the balance between creating a strong impetus for change and winning the trust of OSS communities to drive adoption.

Companies who make extensive use of OSS long ago realized it was in their best interest to know what OSS was coming in through their supply chain. Initially this was for legal and software license compliance, but now increasingly is for tracking of vulnerable software assets. Those companies have tended to also create Open Source Program Offices (OSPOs) to systematically manage their engagement with the OSS projects they consume, as well as to follow professional and technical standards for the OSS code they upstream or release themselves. Those companies tend to put an emphasis on security best practices, train their employees on how to develop secure software, fund security audits for the OSS they rely upon, and invest in tooling to spot issues or projects of concern before they become an issue.

It's time we apply these software security best practices to the whole of the software ecosystem, and the OSS ecosystem is the critical place to start because of the shared dependency most organizations in the world have on the same commonly-used OSS components.

The Mobilization Plan: Overall Goals and Activity Streams

A discussion between the private sector, US government experts, and OSS foundations at a January 2022 meeting at the White House[3] discussed 3 overarching goals:

- **Securing OSS Production:** focus on preventing security defects and vulnerabilities in code and open source packages in the first place
- **Improving Vulnerability Discovery & Remediation:** improving the process for finding defects and fixing them
- **Shorten Ecosystem Patching Response Time:** Shorten the response time for distributing and implementing fixes.

We feel this is a useful frame to present a series of "activity streams" we propose to meet each of these three goals. A summary of the streams in each goal is given below. The plan for implementing each stream is then more fully described in the appendices at the end of this document.

The plan for each stream described in this document represents the collective effort of a small team of domain experts drawn from the OpenSSF community over the course of a few short weeks. As time goes on we expect these streams, and thus the overall plan, to evolve as we: perform further due diligence on the strategy for each stream; look for additional starting points or existing efforts we had not been aware of; identify individuals or organizations who may be able to perform some of the work on a voluntary basis; or other factors change as the plan becomes more widely known. We are also eager to engage the broader OSS community - individual developers, open source foundations, and organizations who make use of open source code of all sorts - in the further evolution of these plans. Finally, as sources of funding are identified for each stream, and the clock begins on a plan, short-term goals may be adjusted to fit the funding available. Please keep these humble beginnings and openness to change in mind while reviewing the plans.

To the degree that any of these plans call for the creation of new technical efforts at the OpenSSF, or create results for redistribution through the OpenSSF, then the technical governance processes used by the OpenSSF, in particular the oversight and management of the Technical Advisory Council, will help ensure that a consistent baseline level of quality and alignment with other OpenSSF efforts is achieved. Nothing in this document should be construed as a proposal to the TAC for a specific effort, or as a proposal to route around that governance. Alternatively, some of these streams may be more efficiently delivered by other existing or new organizations.

3 https://www.whitehouse.gov/briefing-room/statements-releases/2022/01/13/readout-of-white-house-meeting-on-software-security/

Goal 1: Securing OSS Production

Writing secure software is not easy. Several different, overlapping challenges contribute to the problem:

- formal and informal computer science education curricula typically do not cover security techniques
- security researchers constantly discover new forms of vulnerabilities in the ecosystem, even in programming languages and systems that have existed for decades
- existing security tools can help identify security issues, but require training and expertise to really benefit the overall quality and security of the code.

Security is often characterized as a process[4], not an end-state. Therefore investments here need to be focused on improving awareness and education, improving the processes used by maintainers of critical code, and improving the tools used by all developers for software development.

Stream 1: Deliver baseline secure software development education and certification to all.

It is rare to find a software developer, whether college-educated, educated in other fora such as “boot camps” and coding academies, or self-taught, who receives formal training in writing software securely. A modest amount of training — 10 hours at the very least, 40-50 hours ideally — could make a huge difference in developer performance. There exist such training modules available for free, such as the OpenSSF Secure Software Fundamentals. We propose bringing together a small team to iterate and improve such training materials so they can be considered industry standard, and then driving demand for those courses and certifications through partnerships with educational institutions of all kinds, coding academies and accelerators, and major employers to both train their own employees and require certification for job applicants.

Cost: $4.5M for the first year, $3.45M per year beyond.

Stream 2: Establish a public, vendor-neutral, objective-metrics-based risk assessment dashboard for the top 10,000 (or more) OSS components.

This stream collects the array of different open source methods (such as the OpenSSF Best Practices Badge and the Security Scorecard) for assessing how well a given OSS component, and the maintainer team developing it, measures up when it comes to practices and methods that would reduce the risk to end-users. Such a platform would also track vulnerabilities in dependencies via software composition analysis (SCA), to understand how a bug in an upstream component affects others. This provides “situational awareness” for organizations that deploy OSS and provides clear guidance to projects wishing to attract more users by reducing their risk.

Cost: $3.5M for the first year, $3.9M per year beyond.

4 https://www.schneier.com/essays/archives/2000/04/the_process_of_secur.html

Stream 3: Accelerate the adoption of digital signatures on software releases.

Digital signatures are a critical part of ensuring that all participants in the ecosystem, from software builders to end users, can verify that the components that they use are indeed the components that they intend to use. While signatures are already relatively commonly used at distribution end-points, in the upstream development process they are far less common. This stream will drive improvement of the existing signing tools and infrastructure and encourage adoption by open source projects. Through training for maintainers and direct source code contributions, the goal is to achieve signed source releases. The goal is to see 50 of the top 200 projects and 1000 of the top 10,000 projects using an interoperable software signing approach.

Cost: $13M for the first year, $4M per year beyond, with a one-time $10M push after the first year.

Stream 4: Eliminate root causes of many vulnerabilities through replacement of non-memory-safe languages.

Some programming languages, like C and C++, make memory safety challenging and can lead to difficult to detect and eliminate software defects. In contrast, most programming languages, like Go and Rust, handle memory management and other kinds of security-sensitive tasks safely by default, making it easier for developers to avoid entire categories of vulnerabilities. Much of the modern Internet software infrastructure is built on software written in C, and that leads to a large number of vulnerabilities each year. About 70% of Microsoft's vulnerabilities in 2006-2018 were due to memory safety issues[5], and in 2020 Google reported that 70% of Chrome's vulnerabilities were due to memory management and safety issues.[6] Identifying small, self-contained, critical components that are good candidates for being rewritten in a memory-safe language, and facilitating these rewrites, could help eliminate an entire category of such weaknesses. This stream would resource that development work, as well as the associated promotion, adoption, and community development activities to make the new code bases the industry standard.

Cost: $5.5M for the first year, $2M per year beyond.

5 "Microsoft: 70 percent of all security bugs are memory safety issues" by Catalin Cimpanu, 2019-02-11, https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues

6 "Chrome: 70% of all security bugs are memory safety issues" by Catalin Cimpanu, 2020-05-22, https://www.zdnet.com/article/chrome-70-of-all-security-bugs-are-memory-safety-issues

Goal 2: Improving Vulnerability Discovery & Remediation

Currently quite a bit of vulnerability discovery takes place in highly compensated arenas — from a response to a bug bounty to zero-days traded on darknets. OSS maintainers often do not have the resources to compete with well resourced adversaries in finding deep bugs in their own code. With sufficient funding, automation, and economies of scale, we can address this imbalance.

Furthermore, as Google VP of Infrastructure & Google Fellow Eric Brewer says, “We’re not too bad at finding defects, but we usually struggle to get them fixed.” We can’t assume existing maintainers on even well-resourced OSS efforts will be able to keep up with a moderate volume of new potential defects. Finding is not enough — that must be paired with resources to remediate. But remediation of an as-yet-undisclosed vulnerability is sensitive work, requiring a thoughtful approach.

Stream 5: Establish the OpenSSF Open Source Security Incident Response Team, security experts who can step in to assist open source projects during critical times when responding to a vulnerability.

At times, an OSS maintainer community can be overwhelmed by the enormity and complexity of dealing with a serious security notification they’ve been served, and may struggle to get the bug fixed in the right way quickly. In these moments, a small team of professional software developers, vetted for security and trained on the specifics of the language and frameworks being used by that OSS project, would be extremely helpful. They would need to be accessible on very short notice for what could be full time work for a number of days or weeks, and given clearance by their employers or other commitments to work confidentiality to protect against disclosure. To cover an array of popular languages and frameworks, and to be able to put 2 or 3 such experts on any given crisis, a stable of 30-40 such experts should be identified, trained, and managed.

Cost: $2.75M for the first year, $3.05M per year beyond.

Stream 6: Accelerate discovery of new vulnerabilities by maintainers and experts through advanced security tools and expert guidance.

Most open source maintainers do not have access to the kinds of security scanning tools (software composition, static analysis, fuzzing, and so on) that can catch vulnerabilities early. Not only are the tools often proprietary, just the server costs alone to run them on a recurring basis could be more than most can bear. This stream would pull together the major source code repository hosts, security tool vendors, cloud platform providers and OSS maintainers to provision scanning and monitoring tools in a vendor-neutral uniform platform. It would also pair those maintainers with selected, vetted, paid security experts who can work with them to validate any potential vulnerability and any proposed fix. This would help those maintainers more quickly discover new vulnerabilities in their code and guide them through a remediation and coordinated disclosure process. It would aim to eventually cover the top 10,000 OSS components.

Cost: $15M for the first year, $11M per year beyond.

Stream 7: Conduct third-party code reviews (and any necessary remediation work) of up to 200 of the most-critical OSS components once per year.

Writing secure software is hard — indeed, it is so hard that all of even the most highly-regarded technology companies and open source projects in the world produce code which is later found to have security flaws which require a software update to remediate. While the number of these flaws can be dramatically reduced prior to the code making it into real-world (“production”) environments, a lot of the technical tools used to find those vulnerabilities are unable to find some of the most critical and complex bugs — that remains the domain of human experts.

We propose an industry-wide coordinated effort to manage and facilitate third-party code reviews and associated remediation work of the most critical OSS software. By having reputable third-party security firms perform code review/security auditing of the most critical open source projects and working with OSS maintainers to publish a Public Report of the findings from each audit, we will: find and remediate yet-undiscovered high-impact vulnerabilities; improve developer, industry, and public sector confidence in critical software projects; set high standards for audit quality; remediate the vulnerabilities found, and report to the public all of the associated findings, to benefit the entire ecosystem. This is hard, human-intensive work, but the returns grow with every successful audit.

We aim to cover 50 of the most critical projects in the first year, and cover the top 200 in years 2 and beyond.

Cost: $11M for the first year, $42M per year beyond.

Stream 8: Coordinate industry-wide data sharing to improve the research that helps determine the most critical OSS components.

A major challenge to objectively determining which OSS is actually “critical” is that usage, download, and dependency data is often considered a proprietary advantage by the software distribution channels who are able to collect that data. The Harvard Census II has made great strides here in encouraging more data sharing in some areas, but much more could be done here to drive a more metrics-driven approach to understanding global OSS usage and risk. Provisioning resources to securely and safely manage that data from a greater number of sources, and then funding further interpretation and related research, would substantially improve our understanding of the on-the-ground reality and lower the risk of “surprises”.

Cost: $1.85M for the first year, $2.05M per year beyond.

Goal 3: Shorten Ecosystem Patching Response Time

Finding and remediating vulnerabilities in open source projects is a critical first step in addressing an issue. But the key goal for any such effort has to be to get the fixed versions of these software components universally deployed everywhere this component is used. This requires that every participant in the ecosystem, software vendors, intermediaries, service providers, and finally the end user engineering and infrastructure teams, understand where OSS components are part of their infrastructure or of their deployed products. We appreciate the role that Software Bills of Materials (SBOMs) play in this context and the attention that is being paid to SBOMs at the public policy level. There are, however, other, complementary efforts needed to ensure that all participants in the ecosystem update to the latest secure version of all components they use and release such updated versions of their components to the ecosystem participants downstream from them in a timely manner.

Stream 9: SBOM everywhere — Improve SBOM tooling and training to drive adoption.

When a major new vulnerability is discovered, enterprises are often left scrambling to determine if, how, and where they may be vulnerable. Far too often, they have no inventory of the software assets they deploy, and often have no data about the components within the software they have acquired. In addition, organizations consider acquiring software but often do not have a way to measure the risk that software components within them contain known vulnerabilities. Many have identified “Software Bill of Materials” (SBOM) as a fundamental building block for solving this problem. But to suitably address the challenge, their adoption must be widespread, standardized, and as accurate as possible.

By focusing on tools and advocacy, we can remove the barriers to generation, consumption, and overall adoption of SBOMs everywhere, we can improve the security posture of the entire open source ecosystem: producers, consumers, and maintainers. This stream addresses that by resourcing a team of developers to improve that tooling and bake it into the most popular software build tooling and infrastructure across all major programming languages. It will also resource educational materials, training videos, default templates, examples, and other community advocacy work to normalize SBOM creation and consumption. Most importantly, it will provision a team to work directly with critical projects to submit improvements directly to add SBOM support, removing last-mile barriers or resistance due to inertia. This stream addresses all three points and works directly with the SPDX team to implement them as a new SPDX security profile.

Cost: $3.2M for the first year, and a TBD amount per year beyond.

Stream 10: Enhance the 10 most critical OSS build systems, package managers, and distribution systems with better supply chain security tools and best practices.

Open source software is built using a range of language-specific build systems before being distributed to end users via package managers. This means wildly different levels of quality and risk permeate through the software supply chain as components from different ecosystems come together in a production environment, making efforts to place unified policies around risk management and mitigation very challenging.

The intent of this stream is to examine the most impactful security enhancements we can make to the distribution of those software artifacts; driving improvements at the package manager level to complement other streams focused on component level security and ecosystem risk.

The expected outcome will be a higher level of visibility and security of package management systems, finding improvement at a critical point of leverage in the open source ecosystems, to allow consumers of open source to attain a greater degree of trust into the composition and provenance of their open source software.

Longer term, packaging and distribution improvements around composition and provenance data should support shortened patch time through faster detection and remediation, improved transparency of vulnerabilities and patches to downstream users, and better security tooling for all developers.

Cost: $8.1M for the first year, $8.1M per year beyond.

Overall Costs

We asked the authors of each stream proposal to develop an approach that was both pragmatic and ambitious, that scaled up existing efforts or applied proven techniques from one part of the OSS ecosystem to the rest, and that set meaningful goals achievable within the first two years if sufficiently resourced. We asked them to develop a budget for these approaches that was both lean and realistic, so as to determine the rough magnitude of investment required to provide high confidence these goals could be met, though as noted earlier there are many factors that could influence these costs both upwards and downwards. And yet, we feel the estimates below adequately sets expectations for the scope of investment required, with perhaps a 50% margin of error in either direction. As with all such work, there are three variables at play: time, money, and scope. We anticipate all three evolving as our work on the plan continues, driving towards a moment where funding, leadership, and approach is locked in and the “first year” clock starts.

In that spirit, we feel the below represents an investment portfolio that, while non-trivial, has an immense potential for substantially greater returns in the form of fewer attacks and disruption. We are eager to work with academics and researchers who might help us quantify that value in the form of fewer data breaches (and fines), system downtimes due to rushed upgrades, fewer cybersecurity insurance pay-outs and lowered premiums, and other signs that these investments will pay off.

STREAM	FIRST YEAR	SECOND YEAR
1. Baseline Secure Software Development Education	$4.5M	$3.5M
2. Risk Assessment Dashboard for OSS	$3.5M	$3.9M
3. Digital Signatures to Deliver Enhanced Trust	$13M	$4M
4. Replacement of Non-Memory-Safe Languages	$5.5M	$2M
5. Open Source Security Incident Response Team	$2.75M	$3M
6. Accelerate Discover and Remediation of New Vulns	$15M	$11M
7. Third Party Audits/Code Reviews and Remediation	$11M	$42M
8. Data Sharing to Determine Critical Projects	$1.85M	$2.05M
9. SBOMs Everywhere: Security Use Cases, Tooling	$3.2M	TBD
10: Build Systems, Package Managers, and Distribution Systems	$8.1M	$8.1M
Total	**$68.4M**	**$79.5M**

Conclusion

Open source software is a form of digital public good, creating wealth and capability for society as a whole in a continuously-renewing form. The private sector's substantial investments into the creation of OSS, and further securing OSS, have created public value and positioned OSS as critical infrastructure, as critical as any highway or bridge. Ad-hoc efforts to improve security have reached a limit. We believe for the value invested these new, organized approaches to improving security across the board will have an outsized impact.
We are very eager to collaborate on further iterations of this document with software experts and organizations with involvement in or dependency upon the global OSS ecosystem. Help us take this plan to a reference-quality version 1.0 and beyond, and to help identify the resources required to implement. Please get in touch with us at operations@openssf.org or join the OpenSSF community to engage.

Acknowledgments

This document was authored with the support and collaboration of the following individuals and organizations: Josh Aas (ISRG), Andrew Aitken (Wipro), Neil Allen (Morgan Stanley), Pete Allor (Red Hat), Gaja Anand (Morgan Stanley), Abhishek Arya (Google), Stephen Augustus (Cisco), Mikael Barbero (Eclipse Foundation), William Bartholomew (Microsoft), Wayne Beaton (Eclipse Foundation), Brian Behlendorf (Linux Foundation), Aeva Black (Microsoft), Stewart Blacklock (Intel), Jeff Borek (IBM), VM Brasseur (Wipro), Josh Bressers (Anchore), Eric Brewer (Google), Bob Callaway (Google), Hilary Carter (Linux Foundation), Adam Cazzolla (Sonatype), Mark Curphey (OWASP), Vincent Danen (Red Hat), Emmy Eide (Red Hat), Jennifer Fernick (NCC Group), Brian Fox (Sonatype), Robbie Gallagher (Atlassian), Kathleen Goeschel (Red Hat), Sarah Gran (ISRG), Mike Hanley (GitHub), Jordan Harband (Coinbase), John Heimann (Oracle), Stephen Hendrick (Linux Foundation), Trey Herr (Atlantic Council), Justin Hutchings (GitHub), Dustin Ingram (Google), Shubhra Kar (Linux Foundation), Amelie Koran (Atlantic Council), Georg Kunz (Ericsson), Rao Lakkakula (JP Morgan Chase), Arnaud Le Hors (IBM), Adrian Ludwig (Atlassian), Brandon Lum (Google), Frank Nagle (Harvard Business School), Gunnar Nilsson (Ericsson), Declan O'Donovan (Morgan Stanley), Bruce Mayhew (Sonatype), Jonathan Meadows (Citi), Amir Montazery (OSTIF), Tim Pepper (VMWare), Alex Quesada (Wipro), Tracy Ragan (DeployHub), Sumod Rajan George (Wipro), Xavier René-Corail (GitHub), Christopher Robinson (Intel), Chris Rohlf (Meta), Trevor Rosen (GitHub), Matt Rutkowski (IBM), Stewart Scott (Atlantic Council), Michael Scovetta (Microsoft), Clyde Seepersad (Linux Foundation), Azeem Shaikh (Google), Nell Shamrell-Harrington (Microsoft), Ax Sharma (Sonatype), Laurent Simon (Google), Jim St. Clair (LF Public Health), Zach Steindler (GitHub), Jason Swank (Sonatype), Steve Taylor (DeployHub), David Stewart (Intel), Kate Stewart (Linux Foundation), Andrew van der Stock (OWASP), Eric Tice (Wipro), Santiago Torres-Arias (Purdue University), John Viega (NYU School of Engineering), David A. Wheeler (Linux Foundation), Michael Winser (Google), Chris Wright (Red Hat), Justin Young (Sonatype), and Alexios Zavras (Intel).

APPENDIX 0

Staffing Costs in the Workstream Plans

The following appendices represent the collective effort of many core participants of the OpenSSF community and several other individuals and organizations beyond. It builds upon existing working groups and efforts within the OpenSSF and the broader open source ecosystem, but is not limited exclusively to what exists today. Wherever possible, these plans reference existing programs, systems, practices, and other efforts so as to be as “shovel-ready” as possible.

Many of these plans call for the development of solutions in the form of services and tools, and the vast majority of the costs involved with those services and tools (even when delivered as a hosted or shared software service) is in the headcount required to specify, design, build, onboard, support, and advocate for the approaches presented. Recruiting of qualified talent is challenging in the best of times, and compensation in these technical fields has grown dramatically. For efficiency, these plans call for staffing that reflects an understanding of the essential functions required to meet the goals and timeframe with a high degree of confidence of success. But in most cases, hiring the team for each stream’s proposed approach can be expected to take 3 to 6, possibly even 9 months. For that reason, the first “year” of costs is actually best understood as laying in over 12 to 18 months, on average, from day one of the project. Hiring itself may be the single largest risk to the fulfillment of goals within a time frame posted by a given stream.

There are also often substantial costs above salaries that a budget must account for, from offering competitive benefits to handling vacations and leave to the overhead in any organization for HR, accounting, and other operational functions which, for efficiency, we did not separately spell out in this document.

Therefore we have used a shorthand rubric of $300k per year of employment for all technical roles, including in the first “year” of each component of this plan, to represent the **fully loaded cost of professional staff**. This is not to express that all hired hands will be offered the same salary, or that the salary is $300k no matter the job function. In some cases, the job function may indeed exceed a salary of $300k. But as an average for a mix of technical and leadership positions it felt reasonable.

We would love to explore reducing these headcount costs through the use of volunteer “secondments” from companies to the various teams this plan anticipates. That will not be possible in some circumstances that involve very sensitive information, and in any case those volunteers will need to meet the same kinds of requirements and performance as a hired or directly contracted individual, and a minimum duration of months and hours/week may be required. But for some companies, volunteering their staff on assignment to this effort as an alternative to a sabbatical, or as a full-year training exercise, may be highly valuable.

APPENDIX 1

Stream 1: Deliver Baseline Secure Software Development Education and Certification to All

The Problem:

Historically, little attention is paid in traditional software engineering coursework that highlights and teaches the importance of good cyber security hygiene and secure coding techniques. Complicating the lack of trained developers is the ever-growing shortage of trained cyber security professionals that can assist developers as they create, test, and release their code. The OpenSSF proposes the following multi-pronged approach to addressing this issue: collect and curate content; expand training; and reward and incentivize developers.

Our Proposed Approach:

1. COLLECT AND CURATE CONTENT

We will collect, curate, improve, and develop as necessary educational materials on secure development practices, modern code management, deployment methodologies, selecting software components (as a developer), as well as language or technology-specific documentation and training needs to be reviewed and tailored. The OpenSSF's Secure Software Development Fundamentals Courses will be a key basis of this work. Wherever possible, these good practices will be added to existing project or tooling documentation. Gap analysis will be performed and new materials identified for collection or creation. E.g.:

- Existing materials from the OpenSSF, OWASP, industry groups, and major open source projects will be collected and analyzed for gaps for needed content
- Create an Open Educational Resource (OER) library of secure development practices
- Focus not just on development skills and techniques, but also associated skills modern developers need such as access controls, testing/validation, and how to build, deploy, and maintain software in DevSecOps scenarios.

2. EXPAND TRAINING

Delivering training across industry is vital to up-leveling the skills of software developers in the OSS community and the full software industry. We propose a strategy of aggressive partnership in that delivery, working with many different organizations and developer-focused events. We also believe in the importance of a consistent approach to demonstrating one's competency in secure software development skills, and recommend a unified certification and badging approach to ensure interoperability, and to create the workforce that the global economy needs.

Thus we propose the following specific actions:

- Provide training on the above for at least four major open source developer or security conferences during the 1st year, such as FOSDEM, SCALE, and Open Source Summit.

- Create a corps of qualified trainers to deliver this courseware.
- Sponsor existing or create alternate delivery streams for educational content (podcasts possibly including an OpenSSF Secure Development Podcast, monthly webinars, more in-depth training videos on specific development topics, thought-leadership blogs).
- Create a unified approach to delivering certification and badging no matter where such training has been delivered, building upon the existing Linux Foundation certification infrastructure.
- Expand learning labs in SKF and other developer education tools.
- Develop 5-10 meaningful partnerships with Historically Black Colleges and Universities (HBCUs), community colleges, technical/trades schools, high school/secondary education organizations, "Girls Who Code", Codebar, Code2040, and other similar organizations to deliver content to learners in alternative or non-collegiate situations.
- Develop 5-10 meaningful relationships with leading universities to add secure development coursework to university curricula.
- Partner with ACM and IEEE to integrate secure software development material into general software development curricula recommendations and accreditation requirements.

3. REWARD AND INCENTIVIZE DEVELOPERS

A significant number, perhaps even the majority, of professional software developers are self-trained and eschew traditional signals of academic accomplishment in their field. In addition, many universities do not require learning how to develop secure software as part of their curricula for software development related degrees. Developers early in their career may not be able to justify the time to take courses and certify versus other paid priorities. Positive incentives for developers to take these courses — beyond simply being more likely to produce secure code — are called for. These incentives could also be used to quantify the reach and impact of investments into training.

- Work with the major "hubs" for OSS development (e.g., GitHub and GitLab) to reflect the certifications and accomplishments of contributors and maintainers to the open source projects they house, and in doing so encourage major OSS projects to have their maintainers complete these courses.
- Offer financial incentives (means-tested) to core maintainers of the most critical OSS to achieve certification. We propose an average of $1000 in incentives and a reach of 200 developers per year.
- Work with major job boards and recruiting sites (e.g., Indeed and LinkedIn) to ensure that these certifications can be presented when viewing a candidate's resume, and trusted to be accurate and up to date.
- Create a tiered badging program with escalating rewards.
- Integrate the new badges/certs into other existing LF/OpenSSF badging projects.
- Highlight badged/certified developers to organizations hiring developers.

First Year (6 to 18 Months) Goals & Costs

- Expand/improve existing content and training materials (including train-the-trainers) (2 people $300K): $600K
- Develop in-person educator-led variations of training materials (e.g., with hands-on lab work) & go to conferences to encourage training (4 person x $300K): $1.2M
- Expand education materials to cover 5 critical languages and ecosystems in more depth (e.g., JavaScript/Node/Typescript, Python, Java, Rust, Go, etc.) (0.5 person x 5 x $300K): $750K
- Expand labs based off of class materials (2 people x $300K): $600K
- Expand Secure Development thought leadership through blogs, webinars, podcasts (1 person x $300K): $300K
- Expand existing certification and badge systems to be displayed in source code repositories, OpenSSF Tooling (Best Practices Badges, Scorecard), and systems like LinkedIn to expand digital and physical badges/recognition rewards: $200k
- Staff a partnerships office to develop relationships for content delivery: (2 people x $300k): $600k
- Means-tested scholarships for taking and passing courses, $300k plus $50k in administrative overhead: $250k
- **Total: $4.5M**

Beyond First Year Annual Goals & Costs

- Localize the most critical educational materials (5 languages x $200K/language): $1M
- Manage relations with organizations and universities to adopt secure software content. (2 people x $300K): $600k
- Create and curate development secure practices library in a Open Educational Resource platform (1 person x $300K + infrastructure $100k): $400K
- Implement a system to track over time the number of people who earn various certifications and the number OSS projects who have a maintainer who has earned a certification, partner with other organizations to make that information easily available: $100K
- Expand Certification Badges and Rewards Programs: $500k
- Continue partnerships office: $600k
- Scholarships for developers: $250k
- **Total: $3.45M**

APPENDIX 2

Stream 2: Establish a Public, Vendor-Neutral, Objective-Metrics-Based Risk Assessment Dashboard for the Top 10,000 (or More) OSS Components

The Problem:

In the open source software supply chain, the lack of a standardized risk assessment and monitoring solution creates cyclical problems for various stakeholders:

Creators and Maintainers struggle to improve the security posture of their open source project/package which can harm the reputation of their work as well as downstream adoption if vulnerabilities are exploited.

End-User developers expose their products/projects to risk as they include popular open source packages as upstream dependencies. They don't know which upstream packages are secure or vulnerable.

Organizations using these open source packages as part of their business or mission critical software are exposed to numerous attack vectors from attackers and possible disruption of business.

Our Proposed Approach:

We propose to build a vendor neutral risk assessment and health monitoring platform to continuously analyze the top 10,000 open source projects and packages across major language and system ecosystems to deliver "situational awareness" regarding risk to developers and consumers of OSS. We propose a goal of 10,000 projects not to set a ceiling on what's possible, but to provide a basis for initial costs and expectations. We anticipate expansion to a larger number of projects as time and funds permit.

This platform will provide Key Risk Indicators for each project/package including but not limited to:

- **Activity Metrics:** Scoring engine processing metrics like code activity, adoption, contributor growth & retention, organizational engagement, dependency in downstream, etc.
- **Vulnerabilities & Best Practices Metrics:** Scoring engine processing metrics like publicly declared and unpublished vulnerabilities in the project code, vulnerabilities in dependent upstream packages, exposed code secrets, community risk metrics, best practices badge level, etc.

- **Compliance Metrics:** Policy engine discovering licenses used in the code base as well as dependencies and scoring against compliance risk levels defined in the project's licensing policy, and where possible on software attributes, or other compliance criteria. A SBOM artifact for the project can be auto-generated as well.

Overview

To develop a credible set of goals and cost estimates, this plan anticipates building on top of the Linux Foundation's existing and in-production LFX Security and Insights platform. Once funded, other approaches will be publicly discussed and considered during early phases of this effort to ensure the most cost-effective and suitable starting point is selected.
The Linux Foundation's LFX platform is the management, analytics, and risk management platform for Linux Foundation-hosted open source projects. Its Security functionality currently tracks a limited set of risk-related data, but it can be expanded to support additional sources of data through its hub-spoke architecture.
Metric data producers will publish or be queried via a connector framework and all metrics will be stored in the LFX platform Data Lake House (DLH). Processors and Views will be written on top of the DLH to serve dashboard-like user interfaces for various stakeholders, and APIs to enable outbound integration into third party systems.

Scanning of OSS packages will be done at a minimum frequency of 1 week (regular scanning) and maximum frequency of 1 day (at onboarding time). Once onboarded, metrics should be available in the dashboard within 1 hour.

DATA SETS AND PRODUCERS (EXISTING AND POTENTIAL CONNECTORS)

- LFX Insights and OpenSSF ScoreCards and Criticality Scores (to be merged)
  - Code Activity (Commits, PRs, issues, releases),
  - Adoption (pulls from container registries, package managers)
  - New Contributors, Dropping off contributors, churn
  - Scorecards information
  - If possible behind the firewall adoption (using LFX connector with stackshare.io)
- Dependency Analysis — Snyk/OSV/Deps.dev/libraries.io
  - Dependencies including transient and application stack map
  - Declared Vulnerabilities (CVE, CWE) in dependencies
  - Licenses in dependencies
- Static Analysis — Blubracket/Snyk/LGTM(CodeQL)/Sonatype Lift/Other OSS
  - Declared Vulnerabilities in package base code (CVE, CWE)
  - Unpublished potential vulnerabilities (code quality)

- Exposed Code Secrets
- Licenses in base code
- OpenSSF Best Practices Badge
- Automated SBOM generator, e.g., https://github.com/opensbom-generator/spdx-sbom-generator

OSS package source code will be cloned daily from source control systems into local container workspaces within the platform. Scanning engines from data producers (example: Snyk CLI or Blubracket CLI or libraries.io) may be packaged into a unified scanning agent, to run against these code bases.

An Admin utility will be delivered for onboarding packages, managing change control and maintaining the scanning Infrastructure. LFX may be able to refactor its project control center (PCC) tool to quickly meet this need. Access to the Admin utility will be RBAC controlled and available to community admins and project maintainers.

This approach is more expensive than the data producers autonomously scanning and publishing metrics to the DLH as the LFX platform will need to maintain and scale the scanning container infrastructure; however it is a more feasible approach since it does not require installation of any bots or vendor agents/apps by maintainers on their packages in the SCM systems.

MODELING

Modeling of data for scoring across areas of criticality, risk and compliance will be driven with a configurable policy based weighing structure specific for each project and available to maintainers to configure..

USER EXPERIENCE

Multiple stakeholder (Maintainers, Governance Groups, Sponsor Organization) Dashboards will be created. Role based access control will be implemented for authorized and elevated access levels (e.g., so only certain people can see information about potentially unknown vulnerabilities). LFX IAM (currently used in LFX Security) will be used as the foundation for RBAC. Public access dashboards (with aggregated and anonymized data) for wide community use may also be published. If required, egress APIs will be built for supporting organizations to query the metrics and scores.

First Year (6 to 18 Months) Goals & Costs

- ▸ Delivery of the proposed enhancements to the LFX platform

- Onboarding of 1000 most critical open source projects, using the Criticality Score and other criteria to select. (Note: Projects can contain 1 repository (monorepo) to hundreds of repositories. For example, Kubernetes has 196 repositories across 5 github organizations. 1000 projects could be the equivalent of 10k repos or more. This budget anticipates 1000 projects and 10,000 repositories)
- Dashboards at a global level and project level
- Admin utility to rapidly onboard and maintain project/package scanning
- **Development:**
  - 5 full-stack engineers with w/emphasis on backend — contractors
  - 3 full-stack engineers with w/emphasis on frontend — contractors
  - 1 Product manager — staff
  - 1 UI/UX designer — staff
  - 1 Technical Lead — staff
  - 1 QA engineer — contractor
  - Cost: $2.5M (12 people x $200k/yr)
  - Note: Estimates here are based on the minimum scope defined in this doc. Scope creep and additional requirements will impact staffing needs.
- Cloud operations expenses: $100k (see table below, plus buffer)
- Onboarding and advocacy: 3 people, 300k/yr = $900k.
- **Total: $3.5M**

Beyond First Year Goals & Annual Costs

- Maintenance and feature evolution of the platform
- Onboarding of 10,000 of the most critical open source projects, and 100k+ repositories
- Development of egress APIs to enable third party integration
- Development and maintenance will require most of the same staff, since successful features always drive new requirements. 10 staff and $2M
- Cloud operating expenses: $1M/yr (based on 10x expansion of tracked projects and repositories)
- As before, note: Estimates here are based on the minimum scope defined in this doc. Scope creep and additional requirements will impact staffing needs
- Continued onboarding, project support and advocacy: 3 people, 300k/yr = $900k.
- **Total: $3.9M**

NOTE ON INFRASTRUCTURE

As stated earlier, an open source project could have 1-100s of repositories / packages. If we extrapolate a median range of repos per Project to be “10”, then the infrastructure cost for 10K projects (100K repos/packages) comes to 820K/annum. We should ideally budget a minimum of $1M/yr for Year 2 onwards.

REPOS	INGEST/ MO	STORAGE/ MO	COMPUTE/ MO	VISUALIZATION/ MO	MONTHLY TOTAL	ANNUAL
10K (YR1)	$1,780	$1,250	$3,600	$206	$6,836	$82,032
100K (YR2++)	$17,800	$12,500	$36,000	$2,060	$68,360	$820,320

This data is generated based on data from 6 connectors. In case of additional data producers or metric streams, the cost will likely be higher.

APPENDIX 3

Stream 3: Using Digital Signatures to Deliver Enhanced Trust to the Software Supply Chain

The Problem:

Today, securing content on the Web is ubiquitous, thanks to decades of investment in protocols between web browsers and servers that guarantee the content you view is the same as the content the web site sent. Such guarantees are not nearly so ubiquitous across the software supply chain; digital signatures, when used or verified at all, often only cover the “last mile” or use a variety of approaches that are difficult to automate and audit. Digital signatures for software distribution must be addressed end-to-end — from original developer and development teams all the way to the end user and end device — to address a range of attack vectors increasingly being deployed. They must be easy for developers to apply and users to verify.

Digital signatures enable a human or automated system (aka computing “workload”) to assert the integrity of actions taken in the software lifecycle, creating a strong guarantee that the bits the end user received are the bits the author intended. By requiring creation and verification of digital signatures for events throughout the Software Development Lifecycle (SDLC), we can bind open source software artifacts to the source code they come from, the processes that build them, and the ingredients they’re composed of, creating unparalleled transparency and security in modern systems.

Our Proposed Approach:

Past digital signature systems have had either steep adoption costs or tremendous difficulty in correctly using them. The sigstore project (lowercase intentional) was founded in 2020 to enable widespread software signing through a simple and ergonomic experience that projects of any size can adopt. It's based on a modular design that supports industry standard cryptographic algorithms, signature formats, and identity protocols.

sigstore seeks to make signatures and their attendant infrastructure "frictionless" and "invisible". It boasts integration paths with a wide variety of systems, built and maintained by a vibrant and diverse community, and has significant traction with signing implementations, RFCs, or statements of intent from Kubernetes, Maven Central, RubyGems, PyPI, and npm.

sigstore's Components

sigstore has three primary components:

1. **A certificate authority** (CA) for issuing short-lived signing certificates (which bind developer or workload identity to ephemeral or long-lived cryptographic keys). It works with many trust models: standalone, derived from OSS foundation, or using sigstore's trust root.
2. **Transparency logs** which store an accurate, immutable and verifiable history of valid signatures/attestations. These are derived from the existing IETF-standard Certificate Transparency concepts in use for SSL/TLS certificates across major browsers today.
3. **Ecosystem-specific libraries** and utilities providing "native" integration with existing package repositories & tools (e.g. Python PyPI, Ruby Gems, container registries)

The identities underlying sigstore certificates can come from a variety of sources and technical implementations and need not be bound to human beings. As an example, sigstore's direct support for GitHub Actions has already demonstrated machine-to-machine artifact signature and provenance generation based on verified GitHub entity identity (organization/repo).

SLSA and sigstore

One area where sigstore's tooling shines is in use cases where OSS projects are being built in the open via cloud-based CI systems like GitHub Actions or CircleCI. Because many such systems have opted to be OpenID Connect (OIDC) identity providers, it's relatively easy to build statements of provenance bound to machine workload identities at the time of artifact creation. This avoids scenarios in which human-bound cryptographic keys or problematic long-lived certificates are bound up in notions of software trust. Using sigstore with cloud-based CI systems, OSS maintainers can attest to the processes and underlying source code used to create their artifacts and can achieve SLSA level 3 with relatively low effort. We believe this enables an expedited path towards adoption of NIST SSDF principles.

The ”Public Good” sigstore: A Network of Trust and Transparency

In order to support a public, frictionless experience and realize its potential to radically change OSS for the better, sigstore needs to manifest as a distributed transparency system assuring consistent and identical information. Independent monitoring and auditing by third-parties is critical, therefore the public-good sigstore infrastructure will be a network of multiple logs and witnesses that jointly attest to the shared state of the network. Just as with transparency infrastructure for TLS certificates, nodes in the public-good sigstore network must adhere to explicit security and neutrality requirements (e.g. government standards compliance such as NIST, regular security audits, mix of academic, government and corporate owners, etc) so as to provide guarantees against emergent threats and other operational concerns.

In 2022, the sigstore community has been working to stabilize the CA and transparency log projects toward a “1.0” release representing readiness for general availability. Meanwhile, the public-good sigstore network already exists in nascent form as a free, publicly available instance of the CA/ledger with a single monitor being run by Purdue University, and the community is working to expand the network with a variety of industry and public-sector partners.

The next step toward expansion is establishing a set of strong Service Level Objectives that each node operator can adhere to. Based on sigstore’s robust adoption curve to-date, we estimate that there will quickly be hundreds of thousands of signing events per day (projected to continue to increase with the overall trends of OSS software) that will need to be recorded across the network and independently verified. To assist with this, the sigstore community is in early discussions with the Internet Security Research Group around the assumption of operational responsibility of the current network nodes. ISRG has significant credibility and experience running similar shared services such as Let’s Encrypt, which powers the TLS infrastructure for over 260 million websites.

Drive Adoption Through Interoperability

Supporting a wide variety of deployment postures and compute capabilities is a primary goal of sigstore. Various identity systems and signature formats offer different advantages and sigstore is not in the business of picking winners. Whether in high-performance cloud environments or low-power IoT devices, sigstore aims for maximum relevance and impact across the needs of both consumers and producers of OSS software, including individual developers and enterprise software companies.

In operationalizing the sigstore network, we seek to **meet OSS maintainers where they are today**, investing in tooling and “on ramps” across ecosystems, prioritized toward high-impact projects. We’re seeking funding to support a limited — but broadly applicable — set of protocols and standards for those ecosystems. In general we support the emergence of generic interop standards so that artifact publishers can easily use a variety of build systems to “build in public” and publish on the public good network. For example, concrete examples of current or planned standards support:

- **Signing formats**: DSSE, COSE
- **Identity**: DiD, OIDC
- **Certificate formats**: X509, SSH

This approach also allows for interoperability with other signature and related identity systems deployed in large enterprises for managing similar needs across their internal systems or the products they build and distribute. Those systems are not going away, and technical bridges can be encouraged or implemented to enable a seamless experience.

Conclusion

The open source community has identified and built sigstore, a viable solution for signing OSS artifacts. sigstore has gained significant traction and mindshare over the last 18 months, and while there is much work yet to be done in sigstore, the necessary toolkit and critical path to global adoption are clear. By joining forces across industry boundaries and adhering to a robust philosophy of promoting standards and interoperability in software signatures, the community can begin to solve deep problems of software supply chain integrity to the benefit of all software consumers worldwide.

First Year (6 to 18 Months) Goals & Costs

- Establishing the public good sigstore network — initial core nodes (logs & monitors) with industry-standard operations (possibly through ISRG) — $0.5M fixed startup + $1.5M/year run rate
- The development work necessary to implement & maintain sigstore support natively in the top 5 most critical OSS ecosystems (as determined by OpenSSF leadership team) $2M per ecosystem x 5 = $10M
- “DevRel” and Advocacy for use of sigstore by open source projects, developers, and end-users (3 people) $1M
- **Total: $13M**

Beyond First Year Goals & Annual Costs

- One-time investment to expand adoption of sigstore tooling from top 5 to top 10 OSS ecosystems — $2M per ecosystem x 5 = $10M one-time
- Expansion of supported protocols, identity systems, and overall improvements in usability of network: $1M / year run-rate
- Ongoing operational expenses due to increased number of nodes in network as well as increased transactional load & storage of digital attestations — $2.5M / year run-rate
- **Total: $10M + $4M/year ongoing**

APPENDIX 4

Stream 4: Eliminate Root Causes of Many Vulnerabilities Through Replacement of Non-Memory-Safe Languages

The Problem:

It is common for vulnerabilities to result from a program mismanaging memory. These types of vulnerabilities are called memory safety vulnerabilities. Such vulnerabilities exist because certain unsafe languages, mainly C and C++, allow programmers to easily make memory management mistakes. Memory safe languages such as Rust, Go, and Java, do not allow for programmers to make the kinds of mistakes that result in memory safety vulnerabilities.

Microsoft estimates that 70% of vulnerabilities in their products over the past decade are memory safety vulnerabilities. Google estimates that 90% of vulnerabilities in Android are memory safety vulnerabilities. The percentages are similar in open source software, with a constant stream of memory safety vulnerabilities being announced and patched.

A 2021 Google Project Zero analysis of vulnerabilities that were detected and disclosed as being exploited in the wild found that 67% were due to a lack of memory safety. “Memory corruption vulnerabilities have been the standard for attacking software for the last few decades and it’s still how attackers are having success,” they stated.

The result of these vulnerabilities and hacks isn’t just technical. It resulted in significant financial loss ($B), interruption of critical services like hospitals and power grids, and invasion of millions of people’s personal lives and data (e.g., maliciously taking control of home security systems’ cameras).

Our Proposed Approach:

Unlike many other types of vulnerabilities, we know how to entirely rid ourselves of memory safety vulnerabilities, not just mitigate them. By moving software away from C and C++ to safer languages we can eliminate the memory safety vulnerabilities which account for a huge percentage of all vulnerabilities.

The work we plan to do to reduce the number of memory safety vulnerabilities consists of two parts:

1. Move the Internet’s most critical software away from unsafe languages such as C and C++ with an efficient strategy that emphasizes upgrading the most security-sensitive components first.

2. Investments in the tools that allow systems engineers to move lower-level systems-level code to a safer language. This will allow us to have impact beyond just the most critical projects. We will focus on tools for systems-level code because it is the most ubiquitous and vulnerable software in the ecosystem, and it is the most likely to be written in unsafe systems languages such as C.

MOVING CRITICAL SOFTWARE INFRASTRUCTURE TO SAFE LANGUAGES

This component of our strategy will rely on Prossimo, a project run by the nonprofit Internet Security Research Group (ISRG). The Prossimo project works to identify the most critical software infrastructure that needs to be moved to a safer language, plan effective and efficient migrations with stakeholders, and oversee execution of the plans.

The top-level risk criteria that Prossimo uses to identify potential investments is:

1. Very widely used (nearly every server and/or client)
2. On a network boundary
3. Performing a critical function
4. Written in languages that are not memory safe (e.g. C, C++)

Of software that fits those criteria, the following opportunity criteria are evaluated:

1. Is this a library or component that can be used in many different projects?
2. Can we efficiently replace key components with existing memory safe libraries?
3. Are funders willing to fund the work?
4. Are the maintainers on board and cooperative?

Based on all of the above criteria, there are Prossimo initiatives underway for the Linux kernel and applications and libraries related to Transport Layer Security (TLS), Domain Name System (DNS), and Network Time Protocol (NTP).

INVESTING IN SAFER SYSTEMS DEVELOPMENT TOOLS

Much of the world's most ubiquitous and critical software is lower-level systems software that underlies almost every computing device in the world (e.g. kernels, basic networking functionality, timekeeping). It's in banks, hospitals, and government systems. This software is typically written in an unsafe language called C, because for much of computing history C was the go-to language for systems software development.

Because lower-level software has more operational constraints than higher-level software (e.g. it typically cannot tolerate a runtime or memory management via garbage collection), developing a memory safe language suitable for systems software is particularly challenging. The Rust language has met that challenge, however, and is an excellent candidate for replacing C in many systems applications.

We plan to invest in the tools that allow systems engineers to move their software to Rust. This means investing in improving package management, compilers, and Foreign Function Interface (FFI) generators. In many cases this will include providing interfaces compatible with existing widely-used components to enable transition. With these tools, adoption of a memory safe alternative will scale much faster without replication of efforts.

We also propose to invest in similar efforts in the Go and Java communities. Both languages are frequently used for systems-level and network applications, including in security-sensitive environments. We would propose to reach out to the community through a series of grant-making efforts focused on the same priorities and criteria described above, focusing on existing projects that need an extra push to get to a production release and adoption and those that may help other workstreams in this plan.

These investments will be made primarily via ISRG, the Rust Foundation for Rust-related work, the Eclipse Foundation for Java-related work, and independent grants for Go-related work.

First Year (6 to 18 Months) Goals & Costs

- Improvements to RustTLS, and implementations of a DNS resolver and NTPd in Rust: $2.5M
- Improve Foreign Function Interface (FFI) generators to simplify transition and mixed language development for Rust and other memory-safe languages, $1M
- Grants to examine, support best efforts, and port other critical software to Go and Java based on the same criteria as posed above: $2M
- **Total: $5.5M**

Beyond First Year Goals & Annual Costs

- $1M/year for sustaining work and advocacy on Rustls, DNS resolver, and NTPd.
- $1M/year for sustaining work and advocacy on Go and Java security efforts.
- **Total: $2M/yr**

APPENDIX 5

Stream 5: Establish the OpenSSF Open Source Security Incident Response Team

The Problem:

The next world-altering critical open-source software vulnerability like Heartbleed or Log4Shell is almost certainly already existing, undiscovered, somewhere in our codebases today. When these vulnerabilities are eventually discovered by either a friendly researcher or a malicious threat actor, open-source maintainers need to make quick decisions that can dramatically impact the cybersecurity posture of entire industries. With this in mind, it is sobering to realize that in the event of a cybersecurity emergency on an under-resourced project, there is often nowhere to which open-source software maintainers can turn for well-vetted, highly-available, expert security vulnerability remediation support. Consequently, developers are suddenly (and sometimes unknowingly) indirectly responsible for an industry-wide security risk, and have no choice but to try to remediate it alone — often without the specialist security knowledge and coordinated disclosure connections to do so safely.

Our Proposed Approach:

We propose the creation of the OpenSSF *Open Source Security Incident Response Team (OSS-SIRT),* a coordinated group of experts from across the industry who will be available to help open source maintainers with all aspects of remediating high-impact security vulnerabilities and related security emergencies. From negotiating disclosure timelines with researchers, to writing software patches, to co-ordinating patching with high-impact downstream projects, open source maintainers will now have a trustworthy, vendor-neutral, vetted, well-orchestrated and experienced group of security professionals available to them, free of charge and independent of the maintainer's employer or geographic location, backed by the credibility of OpenSSF and the Linux Foundation.

EXPRESSLY OUT OF SCOPE:

- Anything involving vulnerabilities in closed-source/proprietary software
- Security improvements to open-source software that are not tactically essential to the patching of newly-reported, high- and critical-impact vulnerabilities in open-source software
- Helping projects or individual enterprises with remediating their own security exposures from another open-source project's security vulnerabilities

First Year (6 to 18 Months) Goals & Costs

GOALS:

- Understand and **document the problem space**, through discussing our proposal with a representative set of open-source developers and security incident responders, especially those with experience managing the disclosure and remediation of high-impact vulnerabilities in open-source software.
- Based upon the research above, identify a **core set of services (year 1)** that we would offer maintainers to resolve these problems. This likely includes some minimum subset of assisting with:
  - Ensuring secure communications between maintainers and researchers when receiving vulnerability reports and/or proof-of-concept exploit code
  - Communicating with security researchers and negotiating disclosure timelines
  - Vetting experts and offering security expertise to help maintainers understand the vulnerability & its impact
  - Writing proposed software patches to remediate identified vulnerabilities (only when specifically requested by maintainers)
  - Review proposed fixes before they are public to see if they fully resolve the reported vulnerability
  - Helping maintainers create and publish software patches and security advisories in a way that will minimize the likelihood of widespread vulnerability exploitation
  - Coordinating confidential communications (as appropriate) with downstream affected projects to assist in a risk-mitigating approach to security patch deployment
  - Helping to evaluate and suggest steps to take if maintainers receive reports of in-the-wild exploitation of novel vulnerabilities in an open source project
- Define **eligibility criteria** to utilize those services. This could be based upon things like:
  - Criticality of the open-source project
  - Criticality of the vulnerability (based on CVSS score or other metrics)
  - Whether or not there exists evidence of in-the-wild exploitation of the vulnerability
  - Whether the maintainers feel equipped to write the patch
  - Whether or not the project already has such resources available
  - Complexity of the associated coordinated vulnerability disclosure process
- Select the **IT and communications infrastructure** necessary to deliver these services, and make a plan for its deployment, operational availability, and security assurance.
- Augment a **playbook/guidance document directed at open-source maintainers** that gives generically useful guidance about what to do in the event of a cybersecurity emergency (e.g.: critical vulnerability is reported) to offer clear instructions on how & when to get our support.

- Define **contractual expectations (including vetting process and ethics agreement) and necessary skills/experience** that will be required of each “firefighter”/incident responder.
- Design an **engagement model** for “firefighters”/incident responders, which addresses things such as:
  - Compensation/funding model
  - Legal/contractual details
  - On-call rotation logistics / how we ensure adequate service availability
- Recruit our **initial cohort of incident responders** for a minimum commitment period of 2 years.
- Document publicly an **operational model** for this service including:
  - Commitments that will be upheld by each “firefighter”/incident responder, including preventing premature disclosure.
  - “How-to” guides for those maintainers/developers wishing to engage with us
  - Service-level agreements
- Perform **outreach to relevant developer communities** to educate them about this new service, including through the creation of a dedicated information webpage, as well as through conference presentations, webinars, “office hours,” media interviews, and speaking with leaders and industry groups involved in OSS development.
- Launch and deliver services for up to 30 emergency incidents.
- Define and **report on key metrics** to understand our success and impact in year 1.

COSTS:

- 24/7/365 First-line/triage/program management support:
  - 1 Project Manager
  - 3 On-call triage / security incident handlers
  - 4 staff x $300k/yr = $1.2M/yr
- Service start-up costs including legal, IT infrastructure, web design, outreach:
  - 1 Program lead: 300k/yr
  - Outsourced/contracting for legal, IT, design: $250k
  - Outreach/marketing: $250k
- Funding for retaining 10-20 security experts (as appropriate)
  - Engagement model to be developed with partner organizations. For the first year, we will look for zero-cost incentives. However this may be a place where further funding is determined to be required to hit first year goals. To be conservative here, we will budget 750k as a reserve for such incentives.
- **Total: $2.75M**

Beyond First Year Annual Goals & Costs

GOALS:

- **Evaluate year 1** in terms of service utilization and outcomes, to determine which changes would be beneficial in year 2+ and what is the likely funding needed to uphold our intended levels of service.
- **Expand our core set of services (year 2+)** to deliver a broader set of security emergency response support than in year 1.
- **Recruit** the appropriate incident responders for these additional services, and secure appropriate funding to achieve these goals.
- Define and **report on key metrics** to understand our success and impact in year 2+.
- Deliver 100 successful security incident engagements.

COSTS:

- As with year 1, there are baseline costs associated with providing 24/7/365 first-line triage, project management, legal, and advocacy. $2M
- As with year 1, we conservatively budget a reserve for incentives for participation of $750k
- To deliver a larger number of security incident engagements, additional program management and technical expertise may be called for, as a sort of second-line triage. Provide for one additional technical staff for this: $300k/yr
- **Total: $3.05M**

APPENDIX 6

Stream 6: Accelerate Discovery, Remediation, and Coordinated Public Disclosure of New Vulnerabilities by Maintainers and Experts.

The Problem:

The number of vulnerabilities is dramatically increasing, driven in part by the increasing velocity of software development. In 2021, NIST reported that more than 22,000 unique vulnerabilities were discovered that year and reported as CVEs. This has increased from 2016 where "only" around 6,000 new vulnerabilities were identified. 2022 is already on pace to match or eclipse that volume. Just as software has evolved from traditional methods of creation and delivery to more agile and modern techniques, so can those contributing to securing systems and remediating vulnerabilities. With average organizations taking 60 days to patch critically-scored vulnerabilities, blue team defenders (in-house and in-sourced vulnerability assessors) need every tool and advantage they can get to empower developers and decrease the window they are vulnerable to exploitation of this swelling tide of vulnerabilities.

Our Proposed Approach:

The OpenSSF proposes an open, multi-pronged approach to addressing this serious issue by increasing the use of software scanning and analysis tools (SAST, DAST, fuzzing, etc) by open source developers; providing use of those tools through a centralized managed service with security experts who engage with OSS maintainers; systematically scanning the OSS code landscape for classes of vulnerabilities as they emerge as new threat vectors; and working with OSS maintainers to improve the state of coordinated vulnerability disclosure. More details on each are below.

Much of this work can build upon the OpenSSF's Alpha-Omega project, in particular its "Omega" half, which launched in 2021 to address these issues. Aligning with other related efforts within the OpenSSF and across industry will amplify these efforts.

Increase Maintainer Use of High-Quality Security Tooling

We propose to conduct campaigns to increase maintainer adoption of security tools and services, focused on the following areas:

- Ensure maintainers understand what tools are available and how to easily use them, coordinating with security tool maintainers themselves, including an "OpenSSF recommended" ruleset where relevant.

- Ensure maintainers have a place to turn to for confidential guidance in understanding the results of those tools and initiate a disclosure process if necessary.

We propose to work with proprietary security tool providers to advocate for their free, "one-click install" availability to open source projects. These campaigns will initially target the top 10,000 most-critical open source projects, in parallel to the analysis work going on through Omega, and will include digital badging for maintainers and metric collection where possible to demonstrate tool efficacy.

Centralized Vulnerability Discovery Through Omega

Building upon work already started in the Omega project mentioned above, we propose to use leading-edge security tools (such as static analysis and fuzzing) to analyze the 10,000 most-critical OSS projects, triage those results, create/refine detection logic to minimize error rates, and work with OSS maintainers to apply fixes through coordinated vulnerability disclosure.

Vulnerability Class Elimination Campaigns

When vulnerabilities are found, they should also be examined to see if they are examples of critical classes of vulnerabilities that apply to many projects. When combined with security analyses already taking place as part of Omega, OpenSSF will conduct campaigns to meaningfully reduce or eliminate certain vulnerability classes across *all* open source projects. This work could include tool validation, advocacy (blogs/articles describing the problem and how to easily detect it), and centralized, automated detection through Omega. Examples of such security campaigns could include:

- JNDI Injection as in the Log4Shell incident (tractable due to its detectability and expected rarity within the ecosystem)
- Command/SQL Injection (due to its criticality and pervasiveness)
- Deserialization (due to its criticality and limited awareness)

Improving Security Tool Quality

We propose to improve security tool accuracy by regularly providing actionable feedback to tool maintainers, contributing improvements to open source security tools (core engine, rules), or analyzing the results of the Omega project to gain insights into tooling quality and gaps. We will also identify and target gaps in the state-of-the-art, managing research, building tools, writing rules, and working with the larger ecosystem on validation and adoption.

Maturing Coordinated Vulnerability Disclosure (CVD)

Discovering and fixing vulnerabilities are critical first steps to addressing the problem. Key to the last mile on how to inform downstream consumers and ecosystems of dependent projects, then coordinating the release of fixes to those constituents in a fast and efficient manner. The OpenSSF is actively working on and promotes the acceleration of some of the following actions to improve this stage of the vulnerability lifecycle:

- Working with the OpenSSF Best Practices WG to educate open source developers on good CVD practices (such as the Guide to coordinated vulnerability disclosure for open source software projects)
- Educate security researchers about effective ways to report to and engage with open source maintainers (forthcoming OSSF Guide to coordinated vulnerability disclosure for security researchers engaging with open source software projects)
- Ensure vulnerabilities, including malicious hijacks, injections, and sabotages, are properly documented using tools like CVE and communicated clearly in advisories including automatable identification of affected components.
- Advocate for use of CVD tools such as CERT-CC's VINCE tool that allows researchers reporting vulnerabilities and software maintainers a neutral, secure place to exchange vulnerability information.

First Year (6 to 18 Months) Goals & Costs

- Increase Maintainer Use of High-Quality Security Tooling
  - Conduct a “growth campaign” to try to get more maintainers to leverage high-quality security tools, and provide direct and confidential guidance on handling results from the tools. 8 staff, technical/advocacy: $2M
- Centralized Vulnerability Discovery through Omega
  - Increase Omega investment to fully cover the top 10,000 OSS projects,
  - cloud-scale analysis, etc. 10 persons x 300k, $2M in infrastructure: $5M
- Vulnerability Class Elimination Campaigns
  - Conduct three vulnerability class elimination campaigns (one every six months), Feed learnings to OpenSSF Developer Best Practices WG. $3M
- Improve Security Tool Quality
  - Deliberate investment (through grants and contracting) to improve commonly used tool quality: $2M
- Maturing Coordinated Vulnerability Disclosure (CVD)
  - 8 staff to cover technical work and content development: $3M
- **Total: $15M**

Beyond First Year Annual Goals & Costs

- Increase Maintainer Use of High-Quality Security Tooling
  - Continue to advocate for maintainer use of high-quality security tools: $1M/year.
- Centralized Vulnerability Discovery through Omega
  - Extend Omega to cover the top 30,000 OSS projects continually: $6M/year
- Vulnerability Class Elimination Campaigns
  - Conduct four vulnerability class elimination campaigns per year: $2M/year
- Improving Security Tool Quality
  - investment in understanding/improving commonly used tool quality,: $1M/year
- Maturing Coordinated Vulnerability Disclosure (CVD): $1M/year.
- **Total: $11M/yr**

APPENDIX 7

Stream 7: Conduct Third-Party Code Reviews (and Any Necessary Remediation Work) of up to 200 of the Most-Critical OSS Components Once per Year.

The Problem:

Writing secure software is hard — indeed, it is so hard that all of even the most highly-regarded technology companies and open source projects in the world produce code which is later found to have security flaws which require a software update to remediate. While the number of these flaws can be dramatically reduced prior to the code making it into real-world ("production") environments, a lot of the technical tools used to find those vulnerabilities are unable to find some of the most critical and complex bugs — that remains the domain of human experts. As a result, code that has not been reviewed carefully by an expert in secure code review generally contains security flaws which, if found and exploited by threat actors, could cause significant damage to enterprises and national security.

Leading research in open source security and third-party code reviews suggests two key points: (1) Many open-source software projects receive little to no third-party code reviews (see Threats, Risks, and Mitigations in the Open Source Ecosystem) and (2) Finding vulnerabilities often requires more in-depth auditing, logic review, and source code analysis, in order to go several layers deep (see Zero Days: Thousands of Nights). This requires dedicated funding, project management and know-how to facilitate and execute.

Our Proposed Approach:

We propose an industry-wide coordinated effort, led by OpenSSF, to manage and facilitate third-party code reviews and associated remediation work of critical OSS software. By having reputable third-party security firms perform code review/security auditing of critical open source projects and publishing a Public Report of the findings from each audit, we can:

1. Find and remediate yet-undiscovered high-impact vulnerabilities in critical open-source software components before they are found and exploited by threat actors
2. Improve developer, industry, and public sector confidence in critical software projects through transparent reporting of scope of the audit and associated security findings
3. Concentrate resources powerfully to enhance the impact of every dollar spent, through a coordinated approach which prioritizes the most critical open-source projects, sets high standards for audit quality, remediates the vulnerabilities found, and reports to the public all of the associated findings, to benefit the entire ecosystem.

Timeline: Within the first year, a coordinated effort could likely manage and deliver 50 third party code reviews of the most critical components, and grow to cover 100-200 yearly audits by the end of the third year.

Discussion of Costs: The cost of security audits and associated remediation efforts can vary by project size and complexity. Based on the collective experience of these authors, audits and remediation work on substantial open source projects typically range from $50k to $250k each time. We assume an average cost of $200k per audit, and will adjust over time based on experience. We estimate that based on current capacity, we can handle 50 audits in the first year, and can grow that through investments in the auditing landscape to 100-200 audits per year afterwards. Overhead to administer the program, select projects fairly, manage performance of the reviewers, organize results and measure impact would be another $1M in the first year to handle 50 audits, and $2M/yr afterwards to scale up.

First Year (6 to 18 Months) Goals & Costs

- **Publish an op-ed, blog post, or similar**, describing the goals (and non-goals) of publicly reported security audits, and best practices for security audits of OSS
- Publish a **blog post** announcing this program, including its scope, project selection criteria/process, vendor selection criteria/process, timelines, expectations of program participants, and other relevant details
- Define **third-party security audit Public Report** requirements and report formatting, to ensure consistency across audits/vendors and to have confidence that all reports will include the information that we need for them to be valuable, actionable and publishable
- Determine **program management** needs for this work stream, and contract or hire staff or vendor to manage the audit program in general
- Define **open-source project selection criteria and process**, to specify how we will prioritize and select projects to be recipients of these funded third-party security audits, including how we will work with project maintainers and core developers throughout this program, and whether (and how) projects can self-nominate
    - Announce this to the broader community
    - Select and announce up to 50 projects for auditing in year 1
- Define **security vendor selection criteria and process**, to clearly define our requirements for security vendors to be eligible, our selection criteria, and a process and timeline for vendors to bid upon the opportunities for year 1 of the program
    - Announce this to the broader community and invite vendors to partake in RFP/bid process
    - Draft a MSA template for all audits in this program
    - Select and announce vendors for up to 50 project audits in year 1

- Conduct and deliver up to **50 Public Reports of third-party security audits/code reviews** of critical open-source projects in year 1
  - Initial versions of these reports must be delivered specifically to project maintainers and those involved in the remediation of vulnerabilities found
  - Final versions of these reports will be published on openssf.org or in the OpenSSF github repo following remediation of vulnerabilities found
- **Remediate vulnerabilities found** during the audit program, within a pre-specified time window (e.g. < 90 days of receipt of initial report), and issue associated technical advisories
- Report on **key metrics** related to this program at the end of year 1
- Determine a **plan for years 2+** of this program of third-party security audits of OSS, based on any lessons learned during year 1
- Conduct 50 audits, at $200k average: $10M
- Administrative staff, program management, legal, IT, advocacy: $1M
- **Total: $11M**

Beyond First Year Goals & Annual Costs

- Year 2+: Conduct audits for up to 200 open-source projects per year
  - Re-visit most critical components on a risk-based schedule of 1, 3, or 5 years.
  - Projects with major releases and green code can request change audits.
- Remediate all vulnerabilities uncovered through this third-party security auditing program
- Report on key metrics related to this program on an annual basis
- Conduct 200 audits each year, again at $200k/yr average cost: $40M
- Administrative staff, program management, legal, IT, advocacy: $1M
- **Total: $42M/yr**

APPENDIX 8

Stream 8: Coordinate Industry-Wide Data Sharing to Improve the Research That Helps Determine the Most Critical OSS Components.

The Problem:

Underlying our efforts to improve the security of open source software is the critical need to know which projects are both widely used but under-maintained. A major challenge to objectively determining which open source software is actually “critical” is that usage, download, and dependency data is often considered a proprietary advantage by the software distribution channels who are able to collect that data. Further, some of this raw data can be misleading: just because a module has been downloaded a large number of times does not necessarily correlate to its criticality. A better list of critical open source software could be derived if concerns about sharing secret information can be addressed. In addition, there would be advantages to developing mechanisms for providing easier mechanisms for acquiring current cross-ecosystem public datasets for research on dependencies, licenses, and similar (similar to libraries.io and deps.dev).

Our Proposed Approach:

No single institution can address this challenge alone. The proposal is to create a multilateral framework and agreement with a small number of organizations willing to pool their anonymized data at a neutral home for access to academic and commercial researchers, under the strict condition that the information can only be used to address the matter of identifying highly popular but under-maintained open source software. Individuals with access to this flow will be required to agree to a set of usage terms and handling requirements.

The number of participating organizations should be kept modest (no more than 10-15 members) to keep the governance model simple while we determine how to scale further. The proposed members could include:

- The Linux Foundation
- The top 3-4 Linux Operating System Vendors
- The 5-7 largest Cloud Service Providers
- The largest 3-4 Software Composition Analysis vendors

The work product from this group will include the best information available from these leaders, though the sources of information and any PII contained within the data will be removed or fuzzed (through differential privacy) to preserve anonymity.

Once the legal framework has been established, this group can begin with sources such as the Harvard Census II database, adding their own sources of information to create a composite understanding of the most used and least maintained projects.

Technical Overview

Detailed technical architecture will be developed once analysis is done on the data sets and producer formats for each industry partner publishing the data to the platform.

A Data Lake House (DLH) hosted by the Linux Foundation's technology team, will be used as the system of record to store, process and query data for various stakeholders. Initially data will be ingested in existing models supported by the producers. Transformation of data feeds into a uniform model will be done via a connector framework to be built for ingres into the DLH.

Producers of the data will be provided with a single page application (SPA) to upload their data sets in supported formats like JSON, CSV to begin with. Additionally we will also expose an abstracted cloud storage endpoint for HTTP based uploads.

We anticipate the infrastructure will need to support ingest and search of 1-5 TB per month of unstructured or lightly structured data. We assume analysis and processing will likely be done off of our cloud resources, but some rate-limited query and BI tooling support would be useful. 100% uptime is not required, but quick-turnaround operations support during US business hours is desirable.

In the first release (MVP), Views will be delivered via light BI interfaces (for non-programmers) and SQL query strings (for programming background) consumers. No Dashboards, Portals or UI are in scope of the MVP. The primary consumer of the MVP solution is slated to be researchers.

In future as Industry partners want to access the data, Multi-tenant Dashboards or a portal may be developed with strict role based access control and appropriate data anonymization and partitioning to ensure no privacy data from one partner is shared to another. Future releases may include egres API to query the data following the same principles of data protection and privacy.

First Year (6 to 18 Months) Goals & Costs

- Initiation of the collaborative
- Definition of a legal framework agreed to by all parties
- Criticality and usage data has been shared by partners
- An initial draft of the consolidated source list will be released publicly.

- Data will be made available to researchers, and a subset or aggregate made available to the public as determined by the collaborative.
- Costs, Staff: 5 FTE x 300k = $1.5M
  - Cloud operations: 1 FTE
  - Developers for ingest, connectors, modeling and data science needs: 3 FTE
  - Researcher and data source relations: 1 FTE
- Operations and third party software: $250k
  - Admin and legal: $220k
  - BI tool access (as needed for researchers): $30K
- Costs, Compute & storage cost, based on Linux Foundation estimates: $100k/yr

DATA SIZE PER MONTH	INGEST & STORAGE PER MONTH	COMPUTE PER MONTH	VISUALIZATION PER MONTH	MONTHLY	ANNUAL
5 TB	**$2,827**	**$1,560**	**$2,800**	**$7,187**	**$86,000**

- **Total: $1.85M**

Beyond First Year Annual Goals & Costs

- Ongoing staff requirements/roles:
  - Administrative support including organizing meetings, taking and distributing notes, outreach to potential members.
  - Programming and technical consulting, for definition of the data to be ingested from the collaboration group, perform data anonymization, and synthesized reporting.
  - Consultation from legal staff on the structure of the collaboration framework and issue resolution.
- Ongoing staff costs: Those same 5 FTE as above, $1.5M
- Ongoing admin, legal, and third party costs: $250k
- Ongoing operations costs, assuming growth in data size, usage, demand: $300k
- **Total: $2.05M/yr**

APPENDIX 9

Stream 9: SBOM Everywhere — Improve SBOM Tooling and Training to Drive Adoption

The Problem:

When a major new vulnerability is discovered, enterprises are often left scrambling to determine if, how, and where they may be vulnerable. Far too often, they have no inventory of the software assets they deploy, and often have no data about the components within the software they have acquired. In addition, organizations consider acquiring software but often do not have a way to measure the risk that software components within them contain known vulnerabilities. Many have identified "Software Bill of Materials" (SBOM) as a fundamental building block for solving this problem. But to suitably address the challenge, their adoption must be widespread, standardized, and as accurate as possible.

An SBOM is a list of the software components in a software system. It is typically maintained as an inventory list that enables developers and organizations to effectively and efficiently evaluate risk management use cases such as vulnerability analysis. There are emerging SBOM use cases including software supportability, incident investigation, and run-time protection.

However, SBOMs are not yet widely generated or consumed in the software industry.

Our Proposed Approach:

By enabling SBOMs everywhere, we can improve the security posture of the entire open source ecosystem: producers, consumers, and maintainers. Just *publishing* SBOMs is insufficient, they need to be *proactively used*.

Removing the substantial barriers to further SBOM adoption lies in ensuring that:

1. the requirements needed to build use cases using SBOMs are clearly understood, documented and implemented in current SBOM specifications
2. there are "friction free" open source tools that generate SBOMs that meet these requirements
3. there is readily accessible education, awareness and implementation guidance and 3rd party support

This stream addresses all three points and works directly with the SPDX team to implement them as a new SPDX security profile.

SBOM Requirements to Enable Security Use Cases

We recognize that there are several SBOM specifications (delivery formats) including SPDX and CycloneDX. Unification into a single specification may be beneficial and remains a goal, but at this point in time is not our focus.

Agreement on common requirements implemented across all SBOM specifications will improve interoperability and integration across tools resulting in improvements to existing solutions and making new solutions easier to develop, implement and maintain. The effort described herein will include representatives from other SBOM specification communities to ensure consistency where possible. The goal is to enable **seamless interoperability** between various formats for key cases. This initiative includes collaborating with a set of advisors including regulators, Chief Information Security Officers (CISO's), security engineers, security tool developers, and other relevant stakeholders.

SBOM Tooling to Enable Security Use Cases

The more ubiquitous SBOMs are, the more valuable they can be to the industry. Generally, we believe that SBOMs should be produced automatically via well-vetted tools every time that software is built or distributed. For instance, the popular package management platforms, such as Apache Maven for Java and npm for Javascript, could ensure that an SBOM is available for every package distributed by the platform. Similarly, when higher level packages such as distribution packages and code repositories are created, SBOMs should be produced by default as a part of that process.

To get to the point where SBOMs are ubiquitous in software distributions, we need to remove all friction for adoption. To address this problem, we must provide open tooling that makes it easy for anyone to create and to leverage SBOMs. We then need to incentivize producers, consumers, and maintainers to adopt such tooling. These incentives can include changing default procedural and tool behavior or badging.

We must do the following (all focused on supporting the initial security use cases):

1. Contribute to existing open source tools
2. Build new open source tools as needed
3. Work with the developer tools ecosystems to incorporate and promote SBOMs

We will be taking a pragmatic approach to SBOM generation tools. We will start with tool sets that will be the fastest to release and economically efficient build while acknowledging that they will have some developer friction and technical limitations. Following these efforts we'll focus on tooling that will take more time to release and require more development effort but will reduce the friction for the majority of developers to as low as is possible zero.

Initial Implementation work will:

- focus on the inventory and vulnerability analysis use cases, delivering a set of written requirements and a machine readable data schema that when implemented in tools generating compliant SBOMs (see below) will reduce risk and improve the security posture of open source across the Internet.
- produce tooling that as well as generating and validating SBOMs enables lossless conversion between SBOM formats as well as related operations such as SBOM SBOM merge and version control.
- work with the SPDX team to implement these requirements as a security profile in the upcoming SPDX 3.0 specification.

Level 1 — clients and SDKs — Operating system and build system-agnostic command line interpreters (CLIs) that can process source and build output artifacts / as well as process operating system and other dependencies.That output a compliant SBOM that includes the necessary data that addresses all use cases. These tools should be able to be run in a manual or automated (e.g., scripted) fashion as part of an end-to-end CI/CD workflow. These tools will include SDKs that developers can use to customize and extend any base tools, for instance to support additional package managers.

Level 2 — package manager plugins — a set of plugins or modules that work natively with the major package managers and repositories such as Maven, npm, and PyPI. These tools will typically require a single line configuration change added in order to run with each subsequent build and will output compliant SBOMs. This work will enhance the best existing open source plugins where they exist.

Level 3 — native package manager integration — by adding native SBOM generation functionality to major package managers, all developers and all build systems will automatically generate SBOMs by default as part of their normal workflow. SBOM generation will become as common and seamless as tooling creating log entries for software builds in a log file behind the scenes.

Level 4 — containerization integration — by adding native SBOM generation functionality to the containerization build process, the system will use SBOM content provided by included packages plus additional artifacts added during container build to output an SBOM that specifies all the components that make up a container.

Level 5 — application/solution integration/deployment — When deploying an application consisting of multiple disparate components (containers, machine images, event driven services) the coordination manager should aggregate the constituent SBOMS to reflect all artifacts that are deployed.

Badging

A badging system that will enable SPDX security profile-compliant SBOMs to display badges for quick identification of use cases such as a vulnerability status.

Tools will be built by primarily providing grants to teams that develop and maintain existing tools and by using specialized contractors to develop and maintain new tools.

Advocacy

SBOMs are mostly new to the software industry. We will develop a broad set of education and awareness materials targeting audiences that include management, solution (use case) developers, security engineers, audit and compliance, developer tools teams, and open source projects.

Material will include written guidance, tutorials, videos, and webinars which will all be hosted on a website with a focus on user experience.

First Year (12 to 18 Months) Goals & Costs

1. A widely agreed upon, published set of requirements and a data scheme that supports the inventory and vulnerability analysis use cases. A first version of this should be published no later than August.
2. The requirements and data schema are implemented as an SPDX 3.0 security profile. The timelines here would be dependent on the SPDX community.
3. Free, widely available, high quality, maintained and supported open source tools that generate SPDX security profile SBOMs including
    a. command line interpreter and SDK (cli)
    b. package manager plugins for widely adopted platforms including Maven (Java), NPM (JavaScript), PyPi (Python), GoModules (GoLang), Ruby (rubygems), NuGet (.NET), Composer (PHP), Rust (Cargo), RPM, APT, dpkg and DEB (C/C++)
4. An SBOM portal for security professionals hosting the educational and awareness materials, serving as the hub for tool downloads and acting as a core news site for “all things SBOM”. Education and awareness content will be integrated into the Linux Foundation certification program.

COSTS:

SBOM Requirements to Enable Security Use Cases — This effort would be largely staffed by volunteers with support from the LF employees and contractors.

- Tooling to Enable Security Use Cases

- SBOM Generation Tools ($1,500,000) (10 ecosystems x ½ yr each x $300K/yr)
  - CLI and SDK
  - Package Manager plugins
- Developer Relations (DevRel) to encourage wide use (4 people x $300K/yr) ~$1.2M
- Reference implementation of vulnerability analysis with OSV.dev

▸ Education and Awareness (~$500,000)

- SBOM portal
- Content development
- Advocacy work

▸ **Total: $3.2M**

Beyond First Year Annual Goals & Costs

1. Native SPDX security profile generation in the world's most popular build and package management solutions: maven, NPM, PyPi, GoLang, .NET, rubygems, Composer and C/C++/Assembler.
2. Alignment of the tools developed in year one to standards track support for these use cases.
3. Support and maintenance of tools created.
4. SPDX security profile badging

An accurate estimate of the resources required beyond the initial phase of work depends on many factors that will emerge during the initial phase, so providing even a rough guide to such costs would not be appropriate at the moment.

APPENDIX 10

Stream 10: Enhance the 10 Most Critical OSS Build Systems, Package Managers, and Distribution Systems With Better Supply Chain Security Tools and Best Practices.

The Problem:

Open source software is built using a range of language-specific build systems before being distributed to end users via package managers. This means wildly different levels of quality and risk permeate through the software supply chain as components from different ecosystems come together in a production environment, making efforts to place unified policies around risk management and mitigation very challenging.

Our Proposed Approach:

The intent of this stream is to examine the most impactful security enhancements we can make to the distribution of those software artifacts; driving improvements at the package manager level to complement other streams focused on component level security and ecosystem risk.

The expected outcome will be a higher level of visibility and security of package management systems, finding improvement at a critical point of leverage in the open source ecosystems, to allow consumers of open source to attain a greater degree of trust into the composition and provenance of their open source software.

Longer term, packaging and distribution improvements around composition and provenance data should support shortened patch time through faster detection and remediation, improved transparency of vulnerabilities and patches to downstream users, and better security tooling for all developers.

Three lines of effort are proposed:

- **Line 1: End-to-end Package Management**: Focus on package managers: what baseline security capabilities are required (including ingesting from build systems, data they make available for validation downstream, patch notification and recall tools for developers, and robust detection and removal of malicious packages). Enumerate package managers, evaluate them against required security capabilities. Subsequently request owning organizations close the capability gap, offering funding in the case of non-profit or community ownership.
- **Line 2: Package Composition**: Focus on improvements to visibility into software composition, recognizing that the deep dependency chains in open source often span many languages and package formats, and visibility into the **full depth** is crucial in quickly detecting and remediating known vulnerabilities.

- **Line 3: Simplify Implementing Common Security Integrity Levels in Package Managers**: Facilitate the evolution of SLSA and build consensus on common security controls, consistent attributes, and normalized terminology and tooling. Support critical open source projects (as identified in stream 8) to adopt SLSA with their build system, package managers, and distribution systems. Improve package managers and repositories to more easily support SLSA, based on the experiences of improving those critical projects, thereby driving wider adoption.

First Year (12 to 18 Months) Goals & Costs

- Identify highest impact build systems and package managers; inventory their existing security capabilities to identify ecosystem gaps.
- Map out security capabilities to determine necessary improvements for build systems, the hand-off from build systems to package managers, and the information that package managers provide via UI / API. Publish for comment and implementation.
- Establish a common set of security capabilities and standards across identified package managers/repositories/registries to ensure interoperability and wider adoption, and implement them.
- Begin to implement this common set of security capabilities across 10 package managers/repositories/registries.

COSTS:

- Centralized Headcount estimate: (7 heads total x $300k/yr): $2.1M
  - 1 overall Program Manager
  - 1 Security Architect
  - 3 Project Technical Leads (one for each of three lines of effort)
  - 2 Developers for line #3
- Per-ecosystem headcount estimate: (10 repos x 2 heads/repo x $300k/yr): $6M
  - 2 developers to address lines of effort #1 & #2 combined:
    - Design and test implementation of these standards/capabilities in specific environments
    - Implement malware detection & remediation
    - Transition/rewire build systems to communicate more information
    - Validate all the above
    - Note: this is the fully loaded cost (20 devs, $300k/yr, $6M) but here there is the potential for substantial cost savings should committed and qualified volunteer FTEs emerge from supporting organizations.
- **Total: $8.1M**

Beyond First Year Annual Goals & Costs

- Most of the goals of the first year describe activities and targets that will really require a second year to complete. Both the core team of 7 FTEs and the ecosystem teams (2 each, 10 ecosystems) should be continued, leading to a second year cost of $8.1M
- However, such costs could tail off substantially after year 2 as the goals are met, support is widespread, and further efforts taken on by existing stakeholder investments.
- Additional goals that could be considered for year two:
    - Work with industry-owned build systems and package managers to implement standardized information sharing and security capabilities
    - Provide funding for non-profit and community-owned build systems and package managers to implement standardized information sharing
- **Total: $8.1M** (for year 2, TBD and likely much less for years 3 and beyond)