



To: Cybersecurity and Infrastructure Security Agency (CISA)

20 February 2024

Re: Shifting the Balance of Cybersecurity Risk:


Principles and Approaches for Secure by Design Software


Docket number: CISA-2023-0027


We are pleased to submit our comments on behalf of the Open Source Security Foundation (OpenSSF). We have thoroughly reviewed the Cybersecurity and Infrastructure Security Agency (CISA) Request for Comment (RFC) on its Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software white paper and have crafted our comments to effectively address the needs and challenges specified. We hope that you will partner with us and others to address these important issues.


Point of Contact (Primary): Omkhar Arasaratnam, OpenSSF General Manager

Website: <https://openssf.org/>

DocuSigned by:

8AAAEFF7FCCA496...
Arun Gupta
Vice President and General Manager for Open
Ecosystem, Intel
OpenSSF Governing Board Chair

DocuSigned by:

39419DF60AD6478...
Omkhar Arasaratnam
OpenSSF General Manager

DocuSigned by:

7059F609FFDE46B...
Brian Fox
Chief Technology Officer, Sonatype
OpenSSF Governing Board Member
Chair, OpenSSF CISA-RFC Committee

DocuSigned by:

2D9352DEF41B456...
David A. Wheeler, PhD
Director of Open Source Supply Chain Security,
Linux Foundation
Facilitator, OpenSSF CISA-RFC Committee

OpenSSF Comments to the CISA Request for Comment (RFC) on “Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software”

The Open Source Security Foundation (OpenSSF) appreciates the opportunity to provide comments in response to the Cybersecurity and Infrastructure Security Agency (CISA) Request for Comment (RFC) [“Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software”](#) (including comments on related topics, as described)

[OpenSSF](#) is a cross-industry organization that brings together the industry’s most important open-source security initiatives and the individuals and companies that support them. The OpenSSF is committed to collaborating and working upstream and with existing communities to advance open-source software (OSS) security for all. We have over 100 [members](#), including some of the largest tech companies, financial services, and other organizations involved in developing OSS. Below are our responses to the paper and the proposed additional topics. We welcome the opportunity to continue the conversation.

[Comments on the Paper](#)

[Comments on Additional Topics](#)

[Incorporating Security into the SDLC](#)

[Secure SDLC / Software Supply Chain By Design](#)

[Available Resources](#)

[Challenges for Small Organizations](#)

[Education](#)

[Hardening/Loosening Guides](#)

[Economics of software vulnerabilities](#)

[Economic Costs of Customer Demand](#)

[Customer upgrade reluctance](#)

[Upgrade instability](#)

[Lack of visibility](#)

[Threat modeling](#)

[Artificial Intelligence \(AI\)](#)

[Threat Modeling for AI](#)

[Operational Technology \(OT\)](#)

[Conclusion](#)

Comments on the Paper

This is overall an excellent paper. We believe the principles in this paper are important.

The paper does have some confusing terminology, as it uses the same term “secure by design” in two subtly different ways. It states: *The term “secure by design” encompasses both secure by design and secure by default.* That’s confusing; it’s saying, “X includes X and Y,” so we have a higher-level concept and a narrower concept using the same term. The paper later states “Secure by default is a form of secure by design.” This simpler formulation could be used consistently to describe the relation of secure by design and secure by default.

Note that on 2024-01-04, one of our participants proposed making “not secure by default” a Common Weakness Enumeration (CWE) entry. (Source: CWE Research mailing list, title “Proposal: Add “Insecure default” as a general CWE category (per “Secure-by-design” paper).”) This could be done by creating a new CWE entry or by modifying an existing CWE entry. The proposal specifically cited this report as a justification. Should this CWE change be accepted, it will be possible to label a failure to be “secure by default” as a vulnerability in the CVE process. In addition, we believe that all CVE entries should be required to include a CWE, as this would enable efforts to identify and reduce entire classes of vulnerabilities over time. These changes might encourage the application of the ideas in this paper. We would encourage people to help develop this new or revised CWE so it can be accepted and applied.

Page 26 discusses “leading from the top,” but focuses on industry. We also think all organizations, including public organizations, should lead from the top. That includes synthesizing different guidance and best practices so that organizations will have fewer places to consult and fewer conflicting guides. Instead of many having documents to review, it’d be best to consolidate.

In response to page 28, “Secure By Design Tactics” Memory safe programming languages: The OpenSSF fully supports the proposed practice to consider adopting memory-safe programming languages. Yet, adopting memory-safe languages on a large scale presents significant challenges:

- Firstly, it is unaffordable to rewrite all existing C and C++ code into another language. There are tens or potentially hundreds of billions of lines of code written in C or C++.
- Second, even promising alternatives, such as Rust, currently rely on dependencies built on C, C++, or code that requires ‘unsafe’ language extensions.

Therefore, it is prudent to complement activities toward adopting memory-safe programming languages with educational material to reduce the risk of introducing memory-related vulnerabilities in unsafe programming languages. For this reason, the [OpenSSF has developed a guide for selecting C/C++ compiler options that reduce risks when using C/C++](#). Please see

the OpenSSF response to the Request For Information (RFI) on open-source [software \(OSS\) security and memory-safe programming languages](#) for more.

The paper's formatting could be improved for adoption. The unnecessarily large use of dark colors in backgrounds makes this report very expensive to print by small businesses and individuals who only have cartridge inkjet printers. We suggest changing the format so that only a few pages use dark backgrounds and limiting the size and amount of dark images. A non-graphical "clean copy for printing" could also be provided alongside the embellished report. It would also be helpful to be able to more easily copy text from the paper without formatting artifacts, so that it could be incorporated into other materials for adoption.

Comments on Additional Topics

Incorporating Security into the SDLC

The discipline of software development began in the late 1940s. It is obvious today, with hindsight, that security was not always a priority as software was created, shared, and used. Since the seminal text [SDL - The Security Development Lifecycle](#) by Howard & Lipner was published in 2006, many organizations have been working to integrate security practices into their development workflows better. SDL, SDLC, SSDLC, however, an organization may refer to it, was further codified with NIST's 2014 [Secure Software Development Framework \(SSDF\) SP 800-218](#) to help illustrate practices and expectations of software development organizations. The SSDF was updated to reflect academic and industry feedback in 2022.

To effectively implement the principles of secure-by-default and secure-by-design, these values must be instantiated from the outset of a development effort to be integrated most simply, most effectively, and most cost-effectively. "Bolting on" new requirements after a project has begun creates unique challenges and complications and often leads to less successful desired security outcomes.

Before a single line of code is committed to the new product, the basic functional and non-functional requirements must be basically understood and laid out so that as the software engineers begin their work, they can be accelerated to meet these objectives through tooling, templates, and automation to reduce their burden. Understanding core security principles, such as the OpenSSF's [Secure Software Guiding Principles](#), helps set vital tenants for developers of any hardware or software solution should follow.

It is critical to conduct threat modeling to understand how the proposed system can be broken or exploited so that the most common design and execution mistakes can be avoided. Threat modeling that helps the developers think through how downstream users could misuse their proposed solution should be embodied in automated and repeatable tests. There are often large disconnects between the initial creator of software's intended and desired use cases and how

downstream implementers use it. Operators can oftentimes be very creative and stretch the limits of a package with unintended use cases and configurations that are not directly forbidden or blocked by the software's logic and configuration.

On the dependency side of the equation, understanding and following the OpenSSF's [Open Source Consumption Manifesto](#) helps organizations that use open source components in their developed applications or as components within their enterprise systems be more informed and aware participants in the open source software supply chain. Suppliers and downstream consumers must invest efforts to check all software used within their proposed design at the beginning of their product creation. They also must install a robust program to monitor and track their upstream sources for changes and vulnerability reports and ensure they can react quickly to provide updates to correct functional or security issues and build, test, and publish mitigated packages to their downstream.

When repackagers or vendors leveraging other components within their system begin threat modeling, they need to take the entirety of their application into account rather than just connect the threat models of the sub-components. The wider threat model may have threats that are not obvious at the sub-level.

Secure SDLC / Software Supply Chain By Design

Secure-by-design software and SDLC needs to depend on secure-by-design software that itself was run through a secure-by-design SDLC using dependencies that themselves ran through secure-by-design processes. This should be done recursively as much as possible. Ensuring that most, let alone all, software, hardware, etc. involved in your project and supply chain was built with secure-by-design principles is impractical and expensive. In addition, mistakes can happen at any level. This is why it is important to build mitigations, like security scans, into a project's SDLC to account for these things while also working to iterate on SDLC processes and systems relied on by the community to decrease our reliance on those mitigations as the primary line of defense. This is well described in the NIST Special Publication SP 800-204D Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines.

To achieve success at scale, SDLC security must also be approached from a secure-by-design/default perspective. If you are not setting up new projects to do the right things, e.g., generate SBOMs, and have secure builds, it is so much harder to retrofit those projects later. Further, it is difficult to enforce and measure the effectiveness of SDLC security policies and procedures for security-by-design software when the SDLC itself isn't following the same principles. This includes configuration as code to automate access control and policy for SDLC systems and resources and integrations with observability systems to detect security issues and measure effectiveness. It is not enough to e.g. ask for projects to implement a secure build process like SLSA, but SLSA build conformance should be enforced by the SDLC systems themselves.

Following secure-by-default/secure-by-design practices means that all producers within the supply chain should be generating metadata like digital signatures, SBOMs, SLSA attestations, etc., so that downstream consumers can then also implement secure-by-default and secure-by-design as they integrate the software. Third-party code is likely opaque, so downstream organizations must test and evaluate the binary product leveraging risk analysis and different security tools. It is important that organizations need to define the minimal security standards that applications/products/services need to meet before they become operational. Logging, monitoring, IAM, and access management are part of the minimal standards. Each organization should design a minimum SBOM security policy (i.e. according to PCI, SSDF or other standards and enforce it on all collected SBOMS - leveraging automation to accept or deny software or components if necessary. This policy when communicated correctly can help manage suppliers, as well as set expectations of security with downstream users.

As the software is being developed (or composed/integrated by downstream consumers), the developers should have access to secure-by-default tools like IDE, security testing and automation, and an assortment of simple scanning tools that test code quality, fuzz for unexpected conditions, and perform an array of static, dynamic, and software composition analysis (SCA) scans. The defined security policy can be leveraged throughout the developer work cycle to ensure software remains secure by default.

These additional tools need to be seamlessly integrated into the developers' environment and workflow tools. [Research by Facebook](#) found that when they integrated SAST feedback into PRs they saw a fix rate of 70%. When using the same tool to do out-of-band scans to provide hand-curated feedback, the fix rate was essentially 0%. This showed that having immediate feedback in code review made all the difference. A large variety of free open-source and commercial tools exist that development organizations can leverage.

A secure-by-design SDLC contemplates how to protect developers and development infrastructure from [intentionally malicious attacks](#). Many of the latest supply chain attacks are focused on tricking developers into downloading fake components that then execute smash-and-grab style data exfiltration and/or try to provide a toe hold via a backdoor or poisoning the attacked software being built by the SDLC. Defending against these types of novel attacks needs to be done at the SDLC infrastructure level, e.g. using tools like a repository manager or a repository firewall to prevent malware entry.

Secure-by-design SDLC also includes the tools and processes downstream from the development and build systems. This includes how software, and the supply chain/SDLC metadata like SBOMs and SLSA attestations are securely stored and distributed. A lot of the secure-by-design effort is wasted if you have no way of communicating information about the SDLC to downstream consumers both internal and external. This is especially important in open source where there are multiple handoffs between software producers and the eventual software consumer. This includes third party build tools not managed by the open source maintainers or without explicit contract as well as public service package repositories like npm or Maven.

Securing the SDLC extends all the way into production because things like security control settings, might drift unintentionally when a product runs in production. Having the right level of drift detection and correction design, and implementation enhances the software's availability, reliability, and security.

Most organizations practice continuous delivery, so a principle of constant automation leveraging a shared set of policies across all the lifecycle phases of a software is important to adopt to ensure sustainable implementation.

Available Resources

The Linux Foundation through groups such as the OpenSSF and CNCF offers a wealth of free publicly available resources for development and operational best practices that make developing software in a secure-by-default/secure-by-design way more straightforward and easier to adopt. This includes the best practices badge, tooling around detecting drift from best practices like the [OpenSSF Scorecard](#), or enforcing best practices like [OpenSSF AllStar](#). On the SDLC side, there are [SLSA](#) and [S2C2F](#) projects that are defining maturity models organizations can use to assess their supply chain maturity. There are also incubating open source tools like GUAC for the distribution and analysis of SDLC metadata like SLSA attestations, scorecard output, SBOMs, etc. that help enable end users to conform to secure ingestion frameworks like S2C2F.

Challenges for Small Organizations

It is well understood that large highly regulated organizations have an easier time complying with new standards, regulations, and other compliance requirements because they typically already have dedicated staff, processes, and expertise to address such things. Conversely, typical small-to-medium enterprises have a substantial hurdle to vault their processes and offerings up to the heavily regulated equivalent bar. They do not have the same access to expertise and funding required to provide those more stringent services and requirements.

However, smaller software producers often have one major advantage over larger, more established enterprises: They are more flexible and tend to have less technical debt from a large portfolio of legacy applications. Whereas an established enterprise will often have to spend a lot of time and money building out systems, tooling, and processes that comply with new requirements and integrate with existing systems, tooling, and processes, smaller organizations can often just adopt the default without too much work.

Education

What are some examples of commercial entities signaling their demands to universities for knowledge of security and secure coding in graduates of computer science programs? Is knowledge of security evaluated during the hiring stage, or are employees reskilled after being hired?

Many universities continue to act as if security is not necessary for software development:

- No top 40 US “coding” or top 5 non-US Computer Science (CS) school required secure coding in 2019 [[Forrester 2019](#)]
- Of U.S. News's top 24 CS 2022 schools, [only one \(University of California San Diego\) requires security for undergraduates](#).
- One 2021 article pointedly noted, [“Universities don’t train computer science students in security”](#)

Industry has been protesting this lack of mandatory security education in academia for years. The proposal asks, “*What are some examples of commercial entities signaling their demands to universities for knowledge of security and secure coding in graduates of computer science programs?*” Here are some examples:

1. In 2008, a [well-publicized letter by Mary Ann Davidson, Chief Security Officer, Oracle Corporation](#) was sent to various universities. Most universities didn’t reply and none expressed interest in changing curricula to *ensure* that graduates of computer science programs (and related programs) know how to develop secure software. This lack of concern by universities was later noted in [“Remarks to the U.S. Senate Committee On Commerce, Science, and Transportation of Mary Ann Davidson Chief Security Officer, Oracle Corporation, February 23, 2010”](#)
2. SAFECode is a consortium of various companies, including Dell Technologies (formerly EMC Corporation), Juniper Networks, Inc., Microsoft Corp., and Symantec Corp. In [2014 SAFECode](#) noted that the “lack of security engineering awareness and education among the software engineering workforce can be a significant obstacle...” and in [2017 SAFECode](#) noted “You cannot become a building engineer without being trained on fire safety, but you can earn a software engineering degree without having to take any courses on security. Colleges, universities, coding bootcamps, and other developer training organizations must address the security education of software developers or the software security problem will perpetuate for decades to come.”

One survey claimed developers know how to develop secure software, but it is misleading. [The State of Developer-Driven Security Survey, Secure Code Warrior, 2022](#), found that 89% of developers reported receiving sufficient training in secure coding skills. However, what this survey showed is that developers know so little that they think they know more than they do (an unfortunate example of the Dunning–Kruger effect). More than half of those respondents were unfamiliar with common software vulnerabilities, how to avoid them, and how they can be exploited. 92% said they needed more training on security frameworks, and 86% stated they found it challenging to practice secure coding. In short, they thought they knew enough, yet most knew almost nothing.

We in the OpenSSF have developed some training & education materials on how to develop secure software. We would welcome efforts to ensure that those who graduate in fields related to software development (e.g., Computer Science, Software Engineering, and Information Technology) know how to develop secure software. “Graduation” in this context includes not just

colleges and universities, but also alternatives such as coding bootcamps. This minimum necessary knowledge should include secure design principles (such as those identified by Saltzer & Schroeder), common vulnerabilities (at least those identified by the OWASP Top 10 and CWE Top 25), how to systematically prevent those common vulnerabilities, and verification approaches to detect vulnerabilities in code before it is deployed (e.g., fuzzers, web application scanners, SAST tools, and secret scanning).

Linux Foundation Research, supported by the OpenSSF, intends to create a survey of IT professionals to determine more information about the need for education in developing secure software. In particular, we want to understand the kind of security-focused education various groups want and need and identify where gaps in the education landscape currently exist. We would love it if some personnel in CISA and its various partners (contractors, other governments, and their contractors) would participate in the survey so it could more accurately represent the real needs.

We agree that academia [must consider software developers a key part of the cybersecurity workforce](#).

There are some positive developments in some part of professional organizations, with NICET (The National Institute for Certifying Engineering Technologies) [adopting a software supply chain security component](#) into the latest upcoming versions of the Systems Software Integrator certification. This although a positive first step, is certainly not enough to turn the significant skills cap in the industry as a whole.

While there is no such thing as a silver bullet that will solve all software development challenges, there is nothing as foundational or as frequently used throughout the software and product development lifecycles as training and knowledge. Automation and tooling are useful to conduct certain tasks at specific stages, but the human element is present during all steps of software development - creation, assembling, implementation, and retirement. Defensive coding skills, threat modeling, security code review skills, and much more are always with the trained engineer, and provide omnipresent insights during all activities. Quality training in critical cyber and secure development skills becomes a force multiplier that is always present in each task the development teams execute.

Hardening/Loosening Guides

How do software manufacturers decide on their products' default configurations, and how do those decisions affect the length and complexity of the hardening guide?

We support reducing the reliance on hardening guides by implementing safe defaults. Currently, hardening guides are often a tempting solution for product development organizations to offload some responsibility for product hardening to integration teams or customers themselves.

Experience shows that this can result in unnecessary extra work and avoidable weaknesses. For example, it's no longer defensible to have routers sold with a known default password like "admin" and then an accompanying hardening guide that suggests changing the default password.

In general, the solid security of a product is based on its configuration and **operational** aspects. While loosening guides primarily address changing the configuration of a secure-by-default product, some constructs cannot be captured as "loosening" guides but also need to be captured as general operational security configuration guidance. E.g., while "least privilege" and "capture audit trail" are important principles that apply to any system, different organizations use different logging systems that can often not be determined automatically by default. Thus, for example, many systems require operational security configuration guidance, not as a hardening guide but also not as a loosening guide. Instead, they are simply guidance on how to properly configure the system in various environments. In other words, not all configuration information is strictly "hardening" or "loosening."

Another challenge is that different customers often have radically different environments. Manufacturers often supply a default for the products' expected or most common use. However, other uses are often possible. While defense-in-depth is a good practice in general, in some critical environments, it is extremely important to have additional layers when implementing a defense-in-depth approach. Thus, there are still cases where "hardening" guides are important because they help users better defend their more critical systems.

Economics of software vulnerabilities

a. Impact of vulnerabilities on software manufacturers.

i. How do software manufacturers measure their costs for each vulnerability?

The economic repercussions of software vulnerabilities for organizations are substantial and multifaceted. To understand this impact, it is essential to delve into the various aspects shaping modern software manufacturers' financial landscape.

Measuring Costs for Each Vulnerability: Software manufacturers quantify the costs associated with vulnerabilities primarily in terms of time and effort. They significantly emphasize reducing false positives and making optimal upgrade decisions. This approach streamlines the remediation process and aids in cost management.

Financial Impact Over Time: Over time, the financial strain vulnerabilities impose on manufacturers escalate. The rising frequency of cyber-attacks and the consequent necessity for frequent updates compound this economic burden.

Remediation Costs (In-house vs. Post-deployment): Comparatively, in-house remediation is more economical than post-deployment fixes. This conclusion stems from the evident advantages in time and effort savings achievable through well-planned upgrade decisions.

Determining Remediation Methods: Manufacturers' choice of remediation strategies hinges on the specific nature of vulnerabilities and the efficacy of available solutions. Utilizing high-quality security data to facilitate efficient remediation is a testament to this approach.

Tradeoffs Based on Financial Data: In their decision-making processes, software manufacturers lean towards solutions that promise long-term financial gains. This strategy revolves around minimizing wasted time and enhancing long-term efficiency.

ii. *Do software manufacturers measure the financial impact of vulnerabilities over time? If so, what are some examples of common patterns that emerge?*

The economic impact of software vulnerabilities extends far beyond the immediate costs to software manufacturers, encapsulating a broad spectrum of financial implications for producers and consumers. Organizations can mitigate the adverse effects of these vulnerabilities by prioritizing efficient remediation strategies, adopting a secure-by-design SDLC, leveraging high-quality security data, and making informed decisions based on long-term financial outcomes. The emphasis on reducing the burden on consumers through improved open-source consumption practices and strategic management of remediation efforts highlights a comprehensive approach toward minimizing economic repercussions. As evidenced by industry research, such as the [Sonatype State of the Software Supply Chain Report](#), a focused effort on open-source vulnerability identification and mitigation can significantly alleviate the financial strain caused by software vulnerabilities and ensure software manufacturers retain efforts to maximize efficiency and cost-effectiveness. For organizations addressing these challenges, a balanced and proactive approach is crucial for sustaining the economic health and integrity of software supply chains, with benefits extended to software manufacturers and their customers in the long run.

b. Impact of vulnerabilities on customers.

i. Do software manufacturers calculate costs for consumers? If so, how do software manufacturers determine the average cost for customers to deploy software updates to mitigate a software vulnerability?

Of course, financial impact is relegated to the organizations that produce software; its customers also bear a significant part of that burden, influencing their financial and resource allocation.

Cost Calculations for Consumers: Manufacturers account for the costs incurred by consumers, particularly in terms of the time and resources needed for deploying software updates. Efficient vulnerability management and quality security data can alleviate the

associated update burdens. The average customer cost can be assessed based on update complexity and patching efficiency.

Determining Aggregate Customer Costs: The aggregate cost can be estimated by considering factors like the prevalence of the vulnerability, the complexity of the patching process, and the overall customer impact. The focus on reducing unnecessary downloads and wasted time means that aggregate costs can be calculated by evaluating the cumulative effect of these factors across the customer base.

Economic Costs of Customer Demand

The economic costs between customer demand and improved software security present a complex dynamic that is just beginning to play out. However, it's important to note that we should not wait for customers to demand more secure products, as this is akin to saying we should not build safer automobiles until customers demand them. Unlike with a physical good, it is hard for a consumer to judge the underlying quality and security of most software. This is where the government can play a role, in balancing out the market forces that are currently leading us to poor outcomes.

For software manufacturers, the relationship of expected safety pertains to liability and aspects of responsibility like due care and duty of care. This isn't without cost, and the economic impact of delivering safe and secure software products to customers is actively being shaped by the rise of software vulnerabilities and increased attention to cybersecurity due to the exponential increase in attacks. It's important to consider all of these variables when guiding software manufacturers to deliver software to their customers, regardless of any absence of direct demand.

a. In what ways do customers ask software manufacturers to make products more secure?

Customer Expectations for Enhanced Security: Customer demand for increased security can be aligned to the increased focus on managing their software supply chain, which extends to the importance of managing software vulnerabilities and the desire for information through deliverables like a Software Bill of Materials (SBOMs). This underlines an implicit customer expectation for a fundamental level of security in software products, viewed not as an optional add-on but as a standard, unspoken requirement. The industry's increased efforts in reducing vulnerabilities can be seen as a response to this expectation.

b. In what ways do customers ask for specific security features rather than asking for products that are secure by design?

Demand for Security Features vs. Secure-By-Design Products: The industry is gradually recognizing the need for security to be an integral part of the design process rather than an

additional feature. The focus on efficiently managing open source components supports CISA's push towards a secure-by-design philosophy. This shift also recognizes that security should be a primary responsibility of software manufacturers, reflecting evolving customer demands.

c. How can customers measure the security of a product? Can they take that measurement and translate it into long-term costs to decision-makers in a business?

Measuring Product Security by Customers: Customers can assess the security of a product by examining the regularity and effectiveness of updates, along with the manufacturer's approach to vulnerability management. These aspects can be translated into long-term costs for decision-makers in businesses, as products with robust security typically entail lower remediation expenses and a reduced risk of security breaches.

d. What are the inhibitors to customers creating a strong demand signal that software should be secure by design?

Inhibitors to Demanding Secure-By-Design Software: One of the barriers to creating a strong demand for secure-by-design software is the general lack of awareness or understanding among customers about the complexities of software security, especially in open-source environments. This gap in understanding can lead to a devaluation of the importance of designing inherently secure software.

In summary, a shifting industry paradigm shapes the economic costs associated with customer demand for enhanced software security. In many ways, this is the wrong way to look at or quantify security, representing a more typical pattern by software manufacturers to shift the burden and responsibility to users. In many ways, waiting for explicit customer demands for improved security is no different than delaying safety improvements in automobiles until drivers ask for them. Instead, software manufacturers must recognize and acknowledge their responsibility to deliver inherently secure products in response regardless of any implicit customer expectations. We believe this is at the heart of the shift towards a secure-by-design approach highlighting the obligation of manufacturers. While measuring product security through update regularity and vulnerability management provides insights, this is often a case of what the customer doesn't know, which creates the most risk. As a voice for these users, we must emphasize that security is an intrinsic aspect of software and work to help organizations foster a culture where secure-by-design is the standard, not driven by customer demand.

Customer upgrade reluctance

a. What are the primary barriers to customers investing in upgrades that should reduce their risk?

Upgrade instability

Upgrade reluctance is a real problem. Even when software is open source and there is no cost for retrieving an update, there's often a reluctance to upgrade due to other costs and risks. This is a multi-level problem.

Typically, upstream projects develop and release software in an agile cadence that produces rapid changes. Oftentimes, upstream software will produce “breaking” changes with new versions that adopt the latest libraries and protocols that may not be compatible with older releases. These changes are often intended to make the interfaces “cleaner” and may make the software project easier to maintain, but they often create serious problems for end-users. End-users don't get the software to admire its technology; they get the software to do a task. Backward-incompatible changes create significant extra work for end-users that isn't relevant to their goals, costing time and money. In some cases, these changes can be substantive, as new software or hardware has to be purchased or significantly changed. While change is *sometimes* necessary, often it's not strictly necessary.

While in rare cases, change is truly necessary, constantly churning interfaces are not necessary. The Linux kernel, for example, famously keeps its interface extremely stable. As [Linus Torvalds \(leader of the Linux kernel project\) noted in 2005](#), “We care about user-space [external] interfaces to an insane degree. We go to extreme lengths to maintain even badly designed or unintentional interfaces. Breaking user programs simply isn't acceptable... We know that people use old binaries for years and years and that making a new release doesn't mean that you can just throw that out. You can trust us. ... I'm not talking about never obsoleting bad interfaces at all. I'm talking about the unnecessary breakage that comes from changes that simply aren't needed, and that isn't given proper heads-up for.”

This is, in part, an education problem. Software developers need to learn that software development is fundamentally engineering, that is, it's fundamentally about solving human problems by applying resources such as time, money, and knowledge. Focusing on solving (instead of causing) human problems is an important aspect that often isn't given enough attention.

Upgrading a complex system, possibly while meeting extreme availability requirements, is no small feat but often a complex and costly project in itself, involving (re-)integration and acceptance testing. Therefore, operators of complex systems often consider the costs of upgrades in the context of an overall risk assessment, i.e., weighing off the risks of upgrading in relation to addressing a potentially non-exploitable vulnerability. Based on this assessment, upgrades are often considered economically not viable. If more developers understood that backwards-incompatible changes have high costs for their users, and took aggressive steps to avoid unnecessary incompatibilities, users would feel less reluctant to upgrade.

Lack of visibility

We believe that an even larger part of the reluctance for organizations to upgrade is actually due to a lack of visibility. In other words, many organizations can't accurately produce even a primitive bill of materials or a list of their organization's software dependencies. Without this understanding of the dependencies, it shouldn't be surprising that many of them go un-updated. [Research](#) recently determined that when vulnerable software is being consumed, 96% of the time, the open-source project already has provided a fix, pointing squarely at the lack of updates as the largest source of risk.

Threat modeling

a. What are some examples of threat models that software manufacturers have made public?

b. What are some best practices for publishing a high-level threat model that will demonstrate to customers that the software manufacturer has adopted a robust threat-modeling program as part of its SDLC?

The OpenSSF End User working group is working on developing a threat model for enterprise open-source supply chain consumption. We welcome participation in its [development](#). There are classes in how to do threat modeling, e.g., the Linux Foundation has a class in threat modeling, and the OpenSSF secure software development course includes a section on threat modeling. As far as examples go, the [OpenSSF best practices badge has an assurance case](#), including a publicly available threat model, and references a paper on how to generalize this assurance case for other projects.

Artificial Intelligence (AI)

At this time, there is little known about how to develop *adequately* secure AI. We believe there's an urgent need to fund research in the area of developing AI that is *practical* and adequately secure against real-world attacks. While there are many academic papers in the area, current incentives encourage academics to publish papers that are novel but do not necessarily improve security.

[NIST paper "Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations"](#) (January 2024) states that "Mitigating [evasion attacks during use of an AI-supported system, such as adversarial examples] is a well-known challenge in the community and deserves additional research and investigation. The field has a history of publishing defenses evaluated under relatively weak adversarial models that are subsequently broken by more powerful attacks, a process that appears to iterate in perpetuity. Mitigations must be evaluated against strong adaptive attacks.. many of the proposed mitigations against adversarial examples have been ineffective against stronger attacks. Furthermore, several

papers have performed extensive evaluations and defeated a large number of proposed mitigations.” Poisoning attacks (where an adversary intentionally “poisons” training data) have similar problems; it’s known such attacks are relatively easy, but mechanisms to counter them are weak.

There’s a need for novel ideas, but there also needs to be a way to winnow out the thousands of ideas that don’t work. A formally specified set of criteria for “effective AI defenses” should be developed for countering evasion and poisoning attacks, and papers should be evaluated on how well they meet them. In particular, an AI defense must be effective even if the attacker knows that the defense may be deployed.

AI-supported software development tools (both open source and closed source) like Github Copilot consume open source code as training data and have streamlined numerous routine tasks for developers. While this automation holds the promise of establishing guardrails to assist developers in adhering to security best practices, further field experience and research are required to thoroughly examine new attack vectors arising from introducing tools like Copilot.

There is also an emergent software supply chain and associated SDLC tooling for AI, that need to be secured according to the same principles outlined in the SDLC section. AI base models are modified, shared, fine tuned and further modified using new tooling. Creating an adequate threat model and mitigation of the supply chain risk will be an important step for most organizations leveraging AI in software production or feature development to take.

Threat Modeling for AI

Software is the invisible writing that whispers the stories of possibility to our hardware. And we are the storytellers. - Grady Booch.

OpenSSF aims to inspire and enable the community to secure the open-source software we all depend on, including AI.

Threat modeling for AI software should be completed at each layer (the software, the data, and the models and operation). These layers work together to enable the creation, training, and deployment of intelligent applications. AI data and AI models are often stored in a manner that may appear more similar to each other than to AI software due to the nature of their content, purpose, and lifecycle. AI data includes input features and labels used for training models, while AI models store the learned parameters and configurations for making predictions on new data.

Securing an AI system, whether in domain-specific or cybersecurity applications, necessitates the integration of mitigation measures for both AI data and AI models into the secure software development lifecycle (SDLC).

Care and consideration should also be paid to leveraging prebuilt models - i.e. open LLMs such as LLaMa or others. These prebuilt models are often shared and fine tuned, managed very similarly to larger binaries in other parts of the SDLC. Organizations should pay attention to making sure they are leveraging the best practice and workflow already established whilst responding to newer threat models brought by the specific circumstances of AI.

While artificial intelligence continues to advance and may approach human intelligence, it remains crucial for humans to retain control. This control is exercised through software development.

After all, we humans should be the storytellers, not AI.

Operational Technology (OT)

The OpenSSF, along with government, industry, and OT/ICS partners, collaborated last year on a [fact sheet](#) detailing useful steps for OT/ICS product suppliers and operators to follow as they manage the open-source software used within those environments. This collaboration highlighted many of the SDLC, Supply Chain, and software management activities that have been successful within the Information Technology space (IT) and applied them to the more highly constrained OT/ICS environments.

Conclusion

We believe that developing software that is secure by design is important to society. We thank the US government for the opportunity to comment on these important issues.