Crossing Boundaries: Breaking Trust?

Illuminating the increasing Global Complexity of an Open-source Software Supply Chain

---

Thank you for downloading this Lineaje report. Carahsoft is the distributor for Lineaje Cybersecurity solutions available via multiple contract vehicles.

To learn how to take the next step toward acquiring Lineaje's solutions, please check out the following resources and information:

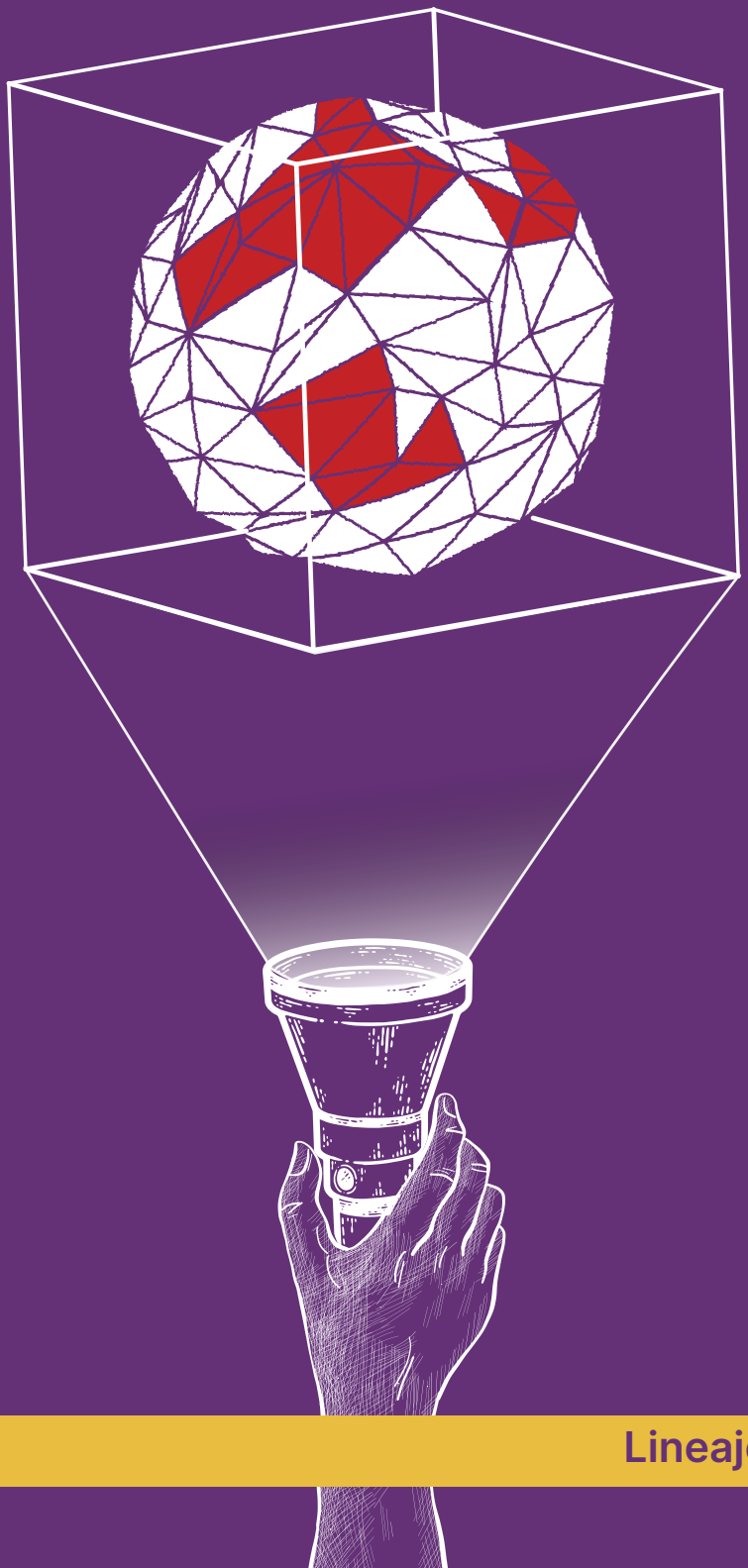
For additional Lineaje solutions:
carah.io/LineajeSolutions

For additional Cybersecurity solutions:
carah.io/Cybersecurity

To set up a meeting:
Lineaje@carahsoft.com
(844)-445-5688

Crossing Boundaries: Breaking Trust?

Illuminating the Increasing Global Complexity of an Open-source Software Supply Chain



Lineaje AI Labs Report

Contents

Contents

Shining a Light on the Continuous Open-source Software Supply Chain of Vulnerabilities and Risks

Currently, 90% of modern applications use open-source components. A typical application uses about 70% open source, the rest is private first-party code or third-party code. Lineaje's research shows some applications with more than 99% open-source – especially those delivered by software contracting firms. On average, the following holds for modern applications:

- 95% of applications' vulnerabilities and risks come from their open-source dependencies.
- Open-source dependencies in a typical application can span 100+ languages. Most development teams effectively support 5-6 languages.
- 6.96% of open-source components are of unknown or dubious origin which means your developers cannot know what they do or whether they are trustworthy.

Open-source packages integrated by application developers are comprised of dependencies from other open-source packages. These dependency chains can extend up to 60 levels deep, with every level developed by a different set of contributors. In fact, 68% of components in an open-source package you source are dependencies pulled from other open-source packages – a pattern that repeats at every level of the chain. The implication is that your open-source suppliers have no real control over what they pass on to you and ingest from them – just as you have no control over what you inherit from them. These suppliers depend on their sub-suppliers for software quality, security, features, and fixes. With No Service Level Agreements (SLAs) to ensure accountability, unmaintained open-source components age like milk.

Even if enterprises get their open-source dependencies from well-known open-source organizations, these open-source organizations cannot fix the vulnerabilities in their own dependencies or have any influence over them. **Lineaje's research of the Apache Software Foundation (ASF)** software revealed, 82% of components in ASF projects are highly risky.

Open-source Software Crosses Boundaries that Private Code Does Not

As we use more and more open-source software, it is critical to understand the differences between managing first-party enterprise code and third-party open-source code. Open-source code essentially runs with the same privileges as first-party private code and, hence, creates the same risks. However, it crosses boundaries that private code does not.

Trust in Critical Software Requires Trust in its Ingredients

Brands are built on trust – in every industry. Trust is built on products crafted with care to deliver an experience the buyer wants – and needs that are met safely.

Lineaje measures trust using two dimensions: integrity and risk. Integrity ensures that each component is known and is exactly what we believe it to be – it's untampered. Risk assesses each component for "Inherent Risk" – the risk it brings into your software. Together, they create a trust score for each component and each application you source, build, sell, deploy, or buy. Start by fixing your least trusted components.

You can't secure what you can't see, and you can't improve what you can't assess. Therefore, a clear understanding of our software supply chain – its components, origins, and dependencies–is paramount. Armed with this knowledge, we can forge a path toward a truly secure and resilient digital future.

Lineaje Illuminates the Complete Software Supply Chain

Over the last year, Lineaje has assessed thousands of business applications, source code repositories, container images, and shipping products across multiple versions. Lineaje dependency crawlers continuously assess 5 million plus open-source packages.

As part of these assessments, Lineaje decomposes all applications using its unique dependency crawler technology, discovering all direct and transitive dependencies and linking their packaged code to their source code. Lineaje Unified Scanner Hub runs a battery of Lineaje-built and open-source scanners, collecting more than 170 attributes for each component.

Lineaje's unique attestation technology deeply fingerprints each component's package and source code, enabling Lineaje to decipher whether the package is derived from the source code it claims to be. Linked with deep dependency crawling technology, Lineaje can attest every shipping component, creating a full supply chain of trust, discovering tampered, unknown origin, and dubious components in every application, open-source package, or shipping product.

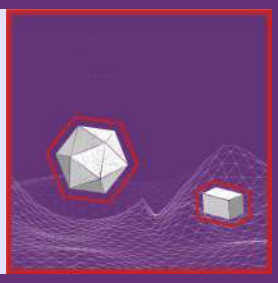
This report summarizes this data to create insights that Lineaje believes are relevant. Lineaje's primary data analysis is creating a deeper understanding of the modern software we source, build, deploy, sell, or buy.

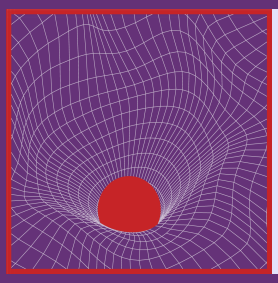
Lineaje serves as a primary data source, providing detailed insights into the composition and origins of modern software. Simply put, Lineaje offers a comprehensive view of the lineage of modern software.

10 Critical Facts About Open-source Security You Can't Afford to Ignore

Pervasive

Open-source contributes two to nine times the code your developers write.



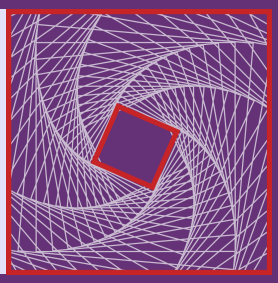


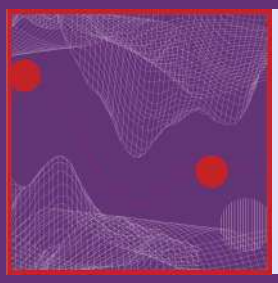
Deep

Open-source embeds **20-60** layers of components from dozens of open-source organizations assembled in a complex LEGO-like structure in a single dependency your developers include in your application.

Unattested

5%-8% of components in open-source dependencies of any application are unknown, tampered with, or are of dubious origin.



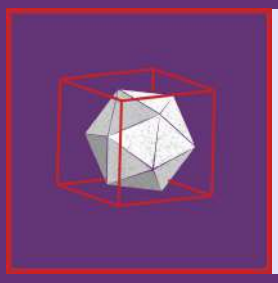


Global

United States contributors commit more code to open-source projects than those from any other country, with Russia following closely.

Anonymous

20% of American contributors choose to remain anonymous, twice the ratio of Russian contributors and three times that of Chinese contributors.

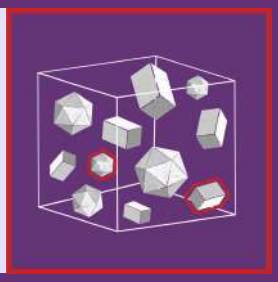


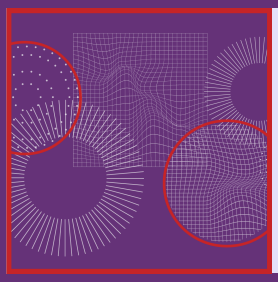
Unmaintained

Shockingly, unmaintained open-source is less vulnerable than well-maintained open-source, which is 1.8 times more vulnerable, and mid-sized teams represent the least risky projects.

Version Sprawl

More than **15%** of components have multiple versions in a single application.



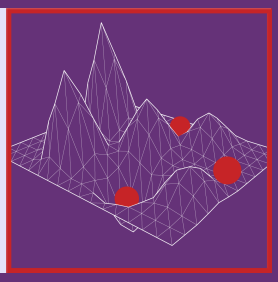


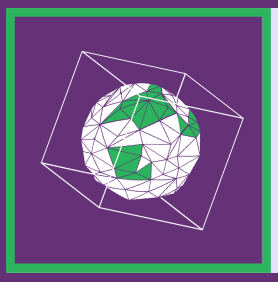
Polyglot

A mid-sized application, on average, pulls in **1.4 million** lines of code in **139** languages and drags in more risky memory-unsafe languages.

Vulnerable

95% of all vulnerabilities come from your open-source dependencies. Knowing which your developers can fix, and which they should not, eliminates at least **50%** of vulnerability fix effort.





Safe

Lineaje Open Source Manager illuminates and manages your open-source dependencies so you can source better software.

CHAPTER 1

Opaque Open-source Is Pervasive

Open-source code is present in your applications and containers, pulled in by AppDev and DevOps at multiple stages of software development, integration, and deployment. Just like Shadow IT, which was pervasive in the last decade, this “Shadow Code” in your applications is pervasive.

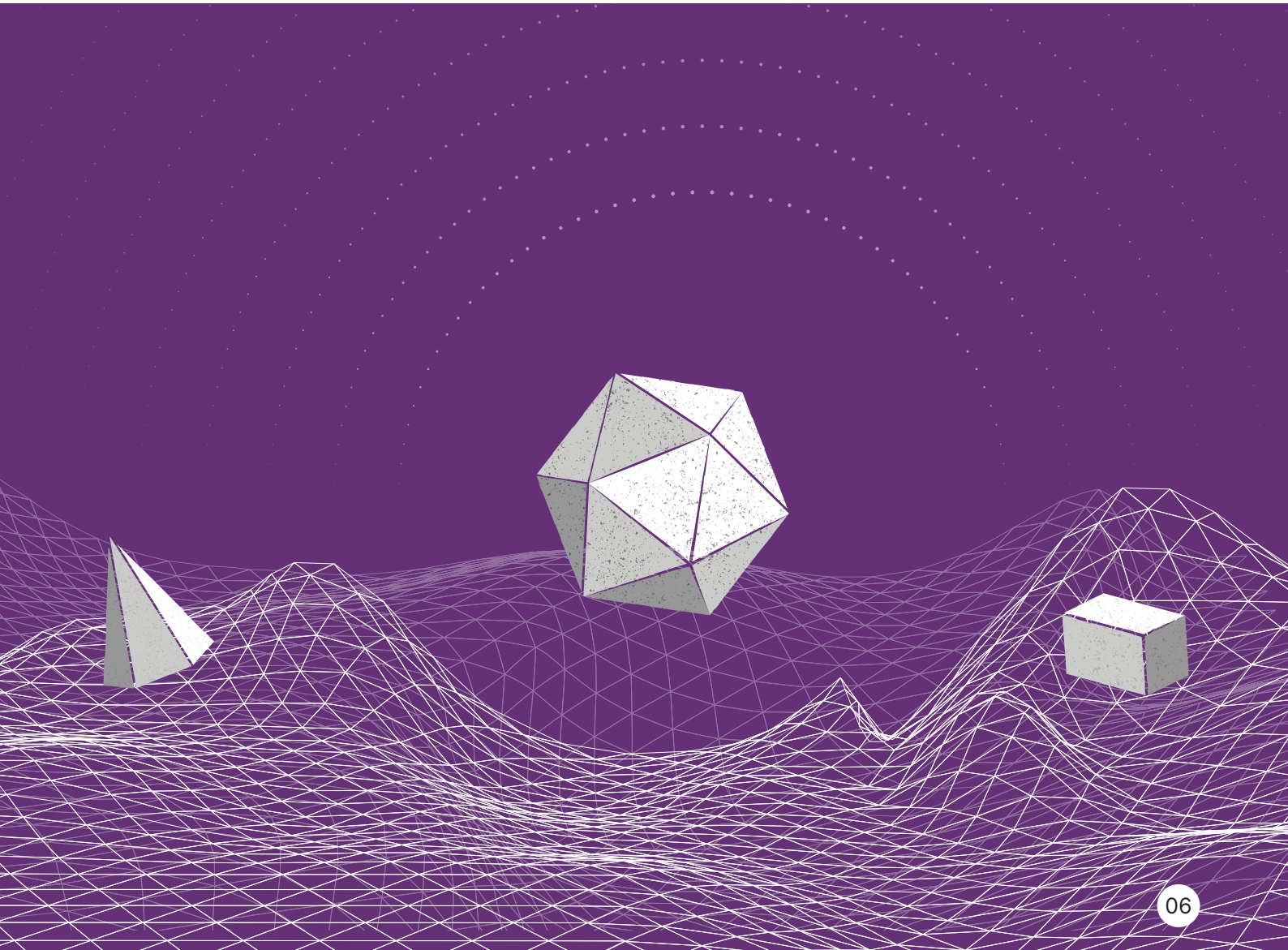
Developers pull in open-source components in the applications they build. Containers pull in base OS images and runtime utilities they need to host application code before they deploy their applications. Open-source packaged dependencies are pulled into repositories using automated tools. As applications are built, integrated, and deployed, more and more Shadow Code is added. Almost all of it is opaque to your developers and DevOps engineers (and the tools they use) who pull them in and should be a fundamental consideration in today's software security tooling selection and development planning approaches.

Quick takeaway

Your software supply chain is almost entirely made of open-source software.

Open-source code is added at all stages of the software development lifecycle and contributes two to nine times the code your developers write.

Open-source Code IS the Software Supply Chain



Open-source Code Escapes Governance and is Embedded in Your Applications

Open-source code is present in your applications and containers, pulled in by applications developers and DevOps at various stages of software development and deployment. Just like Shadow IT, which was pervasive in the last decade, this “Shadow Code” in your applications is pervasive and two to nine times the volume of code your developers write.

Developed outside your organization's boundaries, its inner workings and dependencies remain largely opaque to your developers, hindering their ability to fully assess and mitigate potential risks. In general, the security profile of open-source development is much laxer than that of enterprise software development.

> Detecting dependencies is not the same as trusting their lineage. Just like mere detection of a person’s existence is not a measure of their Trustability, their Children’s Trustability, and their Grand Children’s Trustability.

Additionally, current AppSec tools deployed by organizations to scan their applications are unable to effectively scan open-source code, leaving organizations vulnerable to hidden risks and potential non-compliance with internal security standards.

Open-source code lives in remote open-source repositories and not inside your organization’s boundaries that your AppSec tools scan. So, while they can detect these top-level dependencies in your code, they really do not scan the source code repositories and packages that your code relies on. This oversight leads to the deployment of open-source components without adequate AppSec and SCA scrutiny, exposing applications to potential vulnerabilities and risks.

Private First-party Code to Open-source Code Ratios Vary With Stage

Lineaje analysis reveals that the percentage of open-source code varies based on **application type** and **stage of development**. Open-source packages are pulled in at different stages of the Software Development Life Cycle.

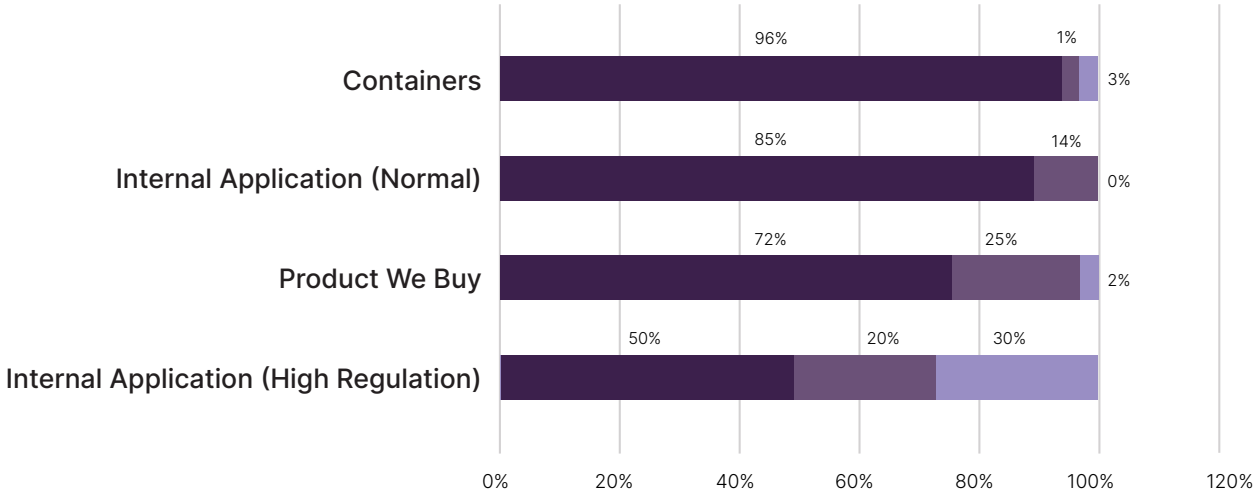
- Enterprise-built internal applications use more open-source code than those built for commercial sale. In general, ProdSec teams are more diligent about open-source selection and maintenance than AppSec teams.
- Greater private Intellectual Property (IP) is being created in applications built for sale, while internal business applications are much more commoditized and use open-source more freely.

> If your open-source security approach only involves scanning source-code repositories, like many SCA tools do, you are not assessing all the open-source your organization consumes.

- Containers, as expected, contain more open-source components than the application code they host, as much of the Operating System stack transitions to open-source Linux distributions.

A key takeaway is that if your open-source security approach involves scanning only one of source–code repositories, artifactories, or containers – you are not assessing all the open-source your organization consumes. Independent scanners for each stage of the SDLC fail to connect obvious dots, have their own limitations, and create a false sense of security for organizations.

What is in your software?



	Internal Application (High Regulation)	Product We Buy	Internal Application (Normal)	Containers
Open-source	50%	72%	85%	96%
Private	20%	25%	14%	1%
Third-party	30%	2%	0%	3%

Open-source Private Third-party

Lineaje Unified Scanner Hub can extend your AppSec tools to scan the source code and packages of ALL your open-source dependencies that existing AppSec and next-gen SCA tools cannot do on their own.

CHAPTER 2

Open-source is Deep and Diverse

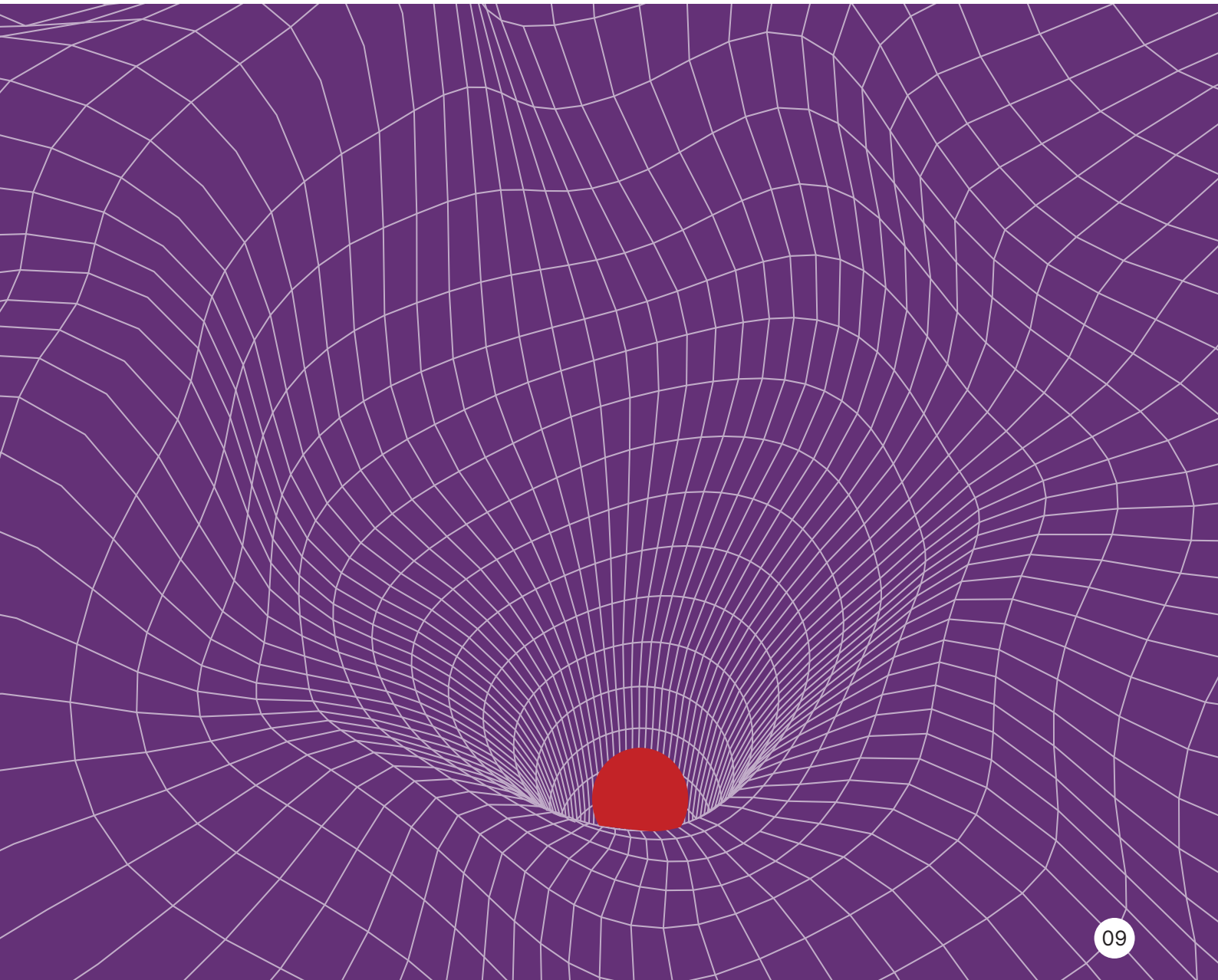
Imagine trying to build a tower using LEGO blocks, K'NEX pieces, and wooden blocks all at once - that's open-source. Your open-source developers are master builders who can pull the pieces together, but each piece comes from different toy manufacturers with different standards and fits.

Modern open-source "structures" typically stack 20-60 layers deep, mixing pieces from different manufacturers far beyond the top-level components developers directly place. These hidden connections (transitive package dependencies) remain invisible to developers, yet one incompatible piece could destabilize the entire structure – and that makes open-source maintenance particularly difficult.

Quick takeaway

Open-source Embeds More Open-source from Different Providers at Each Level

Components from hundreds of open-source organizations are fitted together in a complex LEGO-like structure to assemble a single dependency your developers include in your application.



Mixed Building Blocks Master Assembled Together

In each open-source package, your developers embed packages from other open-source developers. Often, the “supplier” in open-source changes with each transitive dependency. These components can come from large open-source projects open-sourced by players like Google, Baidu, and Facebook to smaller open-source projects staffed by a few volunteers and in many cases, no full-time volunteers.

Open-source Embeds More Open-source which Embeds More Open-source which Embeds More Open-source which...

Each of these "open-source projects" that your dependency drags in is now part of your software supply chain. There is high likelihood that code developed by very smart engineers from Google, Meta, Twitter, Baidu, IBM, Oracle, etc.

is part of your application. The reliability and security of your application directly depend on each open-source project's ability to evolve and maintain their components.

Each transitive dependency may be used in hundreds of other packages as well and is not designed to be a custom fit for this specific parent that includes it and that dragged it into your application. This creates a very complex network of co-dependent packages that are “Master-Assembled” and operate together.

Each of these nested components evolve at different rates and on an independent schedule. This means that newer fixed versions of sub-components are available at different times and may not be taken up-even when available – by the direct open-source projects you depend on.

Updating children of a direct open-source dependency is complex. While in some cases, the updated versions of nested components may be compatible to other components master-assembled together by open-source developers, they frequently are not.

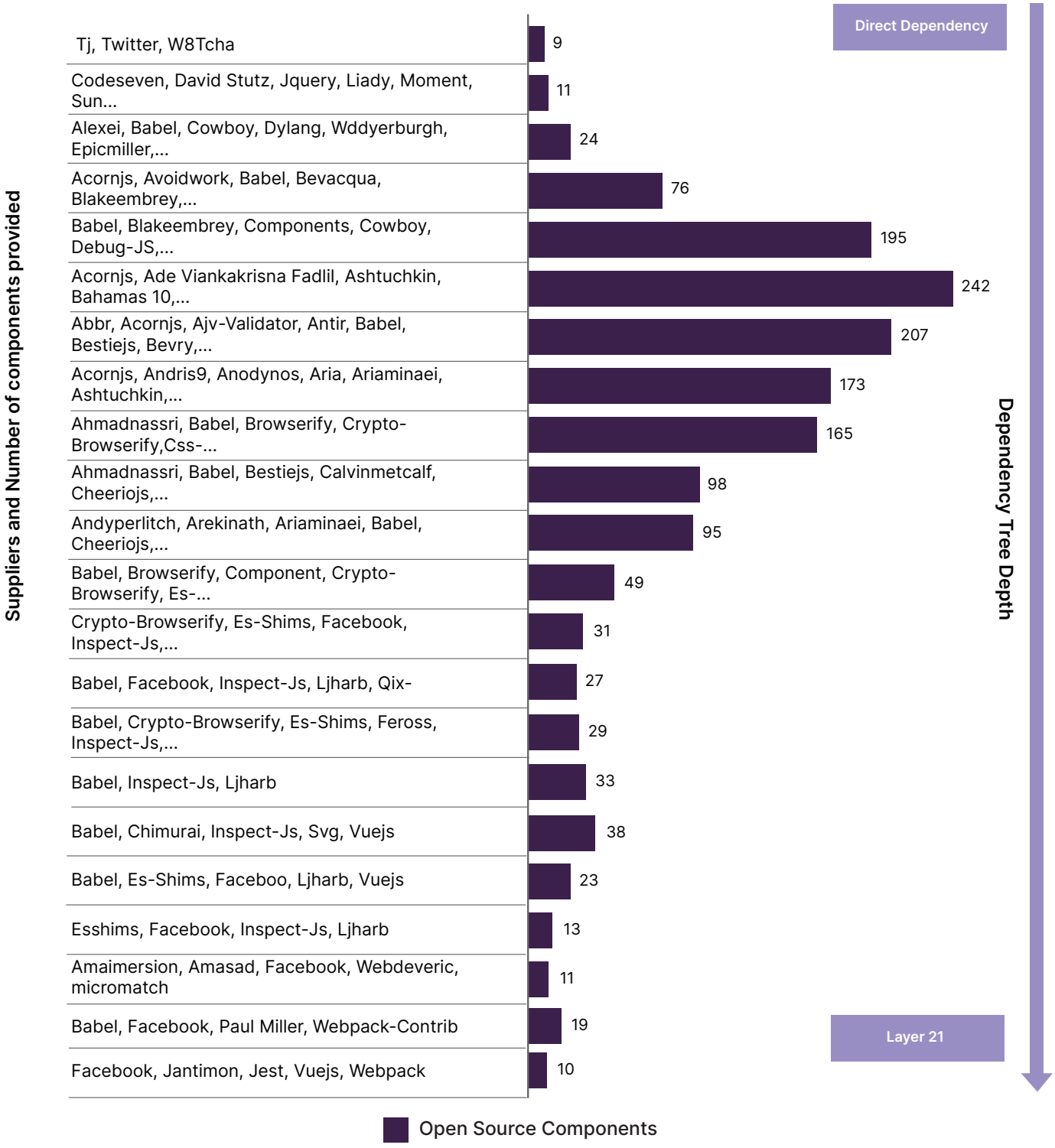
Updating Open-source to “Latest Fixed Versions” of nested components will frequently break your direct Open-source dependency, your application or your deployed container.

Updating open-source to “latest fixed versions” of these nested components will frequently break your direct dependency. Recertifying these direct dependencies that break your developers just sourced but did not build is expensive and complex. This is the single biggest reason why developers cannot easily just pull in a new version of a deeply embedded component without breaking your business application. AppSec teams and tools do not understand software structure.

Dependency Chains are Deep, Complex, and Diverse

An analysis of Apache eCharts used in 280,000 projects shows us the number of components and their main contributors up to 21 layers deep!

Open-source Embeds Other Open-source Components



Lineaje Dependency Crawler technology creates the deepest supply chain dependency tree in the industry.

Lineaje Dependency Crawler technology crawls the package and source code of each dependency to discover its dependencies. Then it crawls the package and source code of each dependency discovering their dependencies and continues to do so till there are no more dependencies. This is similar to web crawling technologies that can illustrate a complex inter-connected web by crawling through each URL to its connected URLs.

CHAPTER 3

Open-source Lacks Software Integrity Attestation

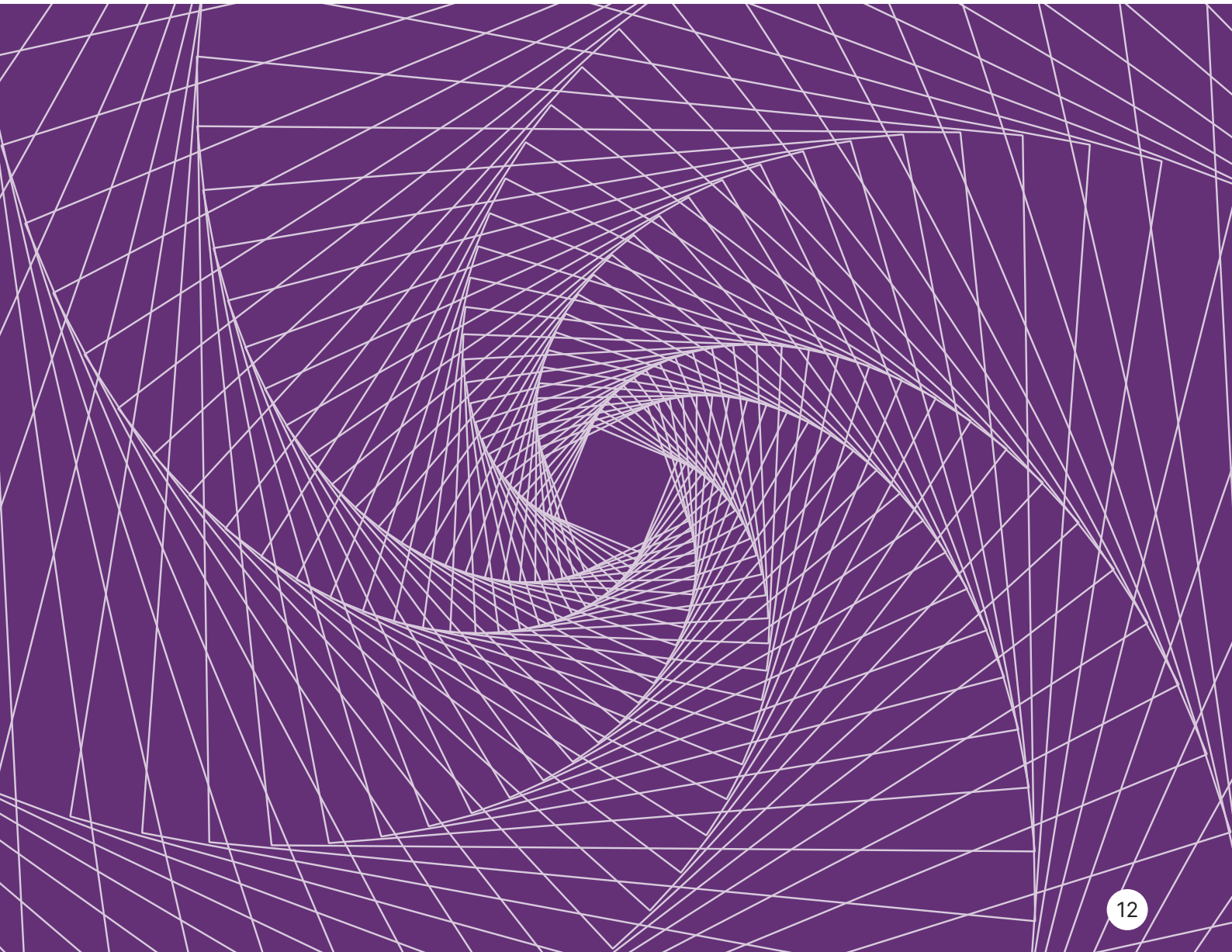
Software integrity attestation is the capability to ensure what you deploy is what you build and what you build comes from your source code. And what you sourced is identical to what was published by the open-source developers who published that package, and each nested package in that dependency is exactly identical to what was published by the open-source developers – at each of the 20+ levels in your software supply chain. And on top of it, attesting that each package is actually built from the source code it came from.

Enterprises embed open-source components without verifying if they are untampered or contain components of unknown Lineage and capabilities. Unsurprisingly, because open-source's integrity is trusted but not verified, it frequently breaks Trust. Because it is assumed transparent, but transparency is unverified, it is frequently Opaque. Because open-source developers cannot validate their dependent's code, it is frequently Unknown.

Given its open contribution model, open-source code is more easily tamperable than private code repositories. Smart state actors, malware writers, and attackers can use one tamper in open-source to attack hundreds of organizations in one effort.

Quick takeaway

5%-8% of components in open-source dependencies of any application are unknown, tampered with, or are of dubious origin.



What Is Software Supply Chain Integrity Attestation and How Is It Achieved?

The internet was built on trust. As the internet became popular and commercially important, it was exploited in thousands of ways – from stealing our identities, to ransoming our data, to shutting down water and electricity, to disrupting elections. A connected world with smart malicious attackers and state actors is dangerous. All these compromises have predominantly been "After the fact" attacks exploiting vulnerabilities in software we use, social weaknesses in people, and deeply technical malicious code.

Unsurprisingly, because open-source is assumed transparent, it is frequently opaque.

The world of open-source software powers the internet, and the applications that run on it, and our enterprises use it to massively digitally transform their operations.

Critical software manages our water, electricity, retail stores, banks, government services, our defenses – almost every aspect of our daily lives. The builders of this software use open-source extensively but do not attest to its integrity. And so, the majority of the software that our lives depend upon – is pulled into our applications, is opaque and lacks Software Integrity Attestation.

Unsurprisingly, because open-source's integrity is trusted but not verified, it frequently breaks Trust. Because it is assumed transparent but transparency is unverified, it is frequently Opaque.

Open-source developers cannot validate their dependent's code nor can your developers. Embedded open-source in all applications is unverified.

Because open-source developers cannot validate dependent's code, it is frequently Unknown. You get what you trust but Verify, open-source's Software Integrity is not attested by Enterprises and Governments that consume it to build our critical systems!

Given its open contribution model, open-source code is more easily tamperable than private code repositories. Smart state actors, malware writers, and attackers can use one tamper in open-source to attack hundreds of organizations in one effort.

Legacy Software Attestation Technologies Cannot Provide Open-source Integrity Attestation

Software integrity attestation is the capability to ensure what you deploy is what you built and what you built comes from your source code. And what you sourced is identical to what was published by the open-source developers who published that package, and each nested package in that dependency is exactly identical to what was published by the open-source developers – at each of the 20+ levels in your software supply chain. And on top of it, attesting that each package is actually built from the source code it came from.

Attesting to the integrity of all software you source is harder than that of software you build in-house. It is built outside your organization, so build-centric integrity checks cannot attest to that. Just like your AppSec tools do not assess the risk in your software supply chain, they cannot attest to the integrity of the direct components you source and their supply chains. At the same time, compromises like 3CX, XZ Utils, and others have proven that existing AppSec and SCA tools fail to detect open-source tampers.

In recent years, code signing techniques and "build-system" driven attestations (such as Google's SLSA approach) have been advocated as valid software integrity attestation solutions. Attacks like XZ demonstrate that these attestation techniques check the compliance box but fail to achieve the basic goal of attestation – detecting software tampers. In fact, they create a reverse issue – the illusion of untampered software when they certify the attestation of software they know little about. Just because an approach like SLSA attests to the code you build in your CI/CD pipeline, it does not mean that the code built outside your CI/CD pipeline is attested.

Open-source Embeds Other Open-source Whose Integrity is Unattested

Open-source embeds other open-source whose integrity is unattested by open-source developers who select and include those dependencies in components we ingest into our software. Given their diversity, open-source developers have no way to check the integrity of the packages they depend on. Enterprise Software Systems also inherently trust open-source and have not invested in attesting the integrity of open-source software they consume.

Most applications – that embed open-source – end up with integrity issues in the open-source they embed. Lineaje automatically assesses and attests the software integrity of all open-source software it scans. Analysis of thousands of enterprise applications reveals the following pattern:

> We would not buy a **$2** can of soup if **7%** of ingredients were of dubious origin. Should your critical software contain known components of dubious origin?

Percent of Components	Trustworthiness	What it means	Implication	The threat that would be detected with this capability
6.96%	Dubious Origin	The component does not exist where the open-source component your developers sourced asserts to originated from.	Stop Ship : Your software contains unknown components • Manipulation of software update/distribution mechanisms	Revival hijack SC-Attacks
Variable	Tampered	The component exists in open-source, but what your software contains is provably different.	Stop Ship : Your software contains components that are clearly “updated” • Manipulation of development tools, development environment, source code repositories (public or private), source code in open-source dependencies • Compromised/infected system images, replacement of legitimate software with modified versions This is probably an attack that would harm your users	3CX, XZ, SolarWinds
0.47%	Minimally Accepted Trust level	The component used in the parent is exactly what was published by the open-source developer who published that component	Minimally Acceptable Integrity Attestation for All Components in your software	While the component is trustworthy, its dependencies may not be
92.57%	Trustworthy	The component used in the parent is exactly what was published by the open-source developer who created it. The source code tied to the component is verified to be the source code of the published package.	Known Origin, Attested Software Integrity	Certified original. While the component is trustworthy, its dependencies may not be. May still be risky due to vulnerabilities, code quality, security posture, geo-provenance, anonymous but authorized contributors, etc.

Lineaje Attestation Engine automatically attests to the integrity of each open-source and private component in your application and alerts you on any component that is not fully trustable. It’s the only technology that can attest all open-source software you use in your applications and detect inherent tampers in open-source you consume.

CHAPTER 4

Open-source Software Is Global and Often Anonymous

A key strength of open-source software is its collaborative development by a global community of developers. However, as geopolitical tensions rise, state actors take a deeper interest in supply chain compromises, and digital dominance increasingly leads to economic and security dominance – geo-provenance of software is scrutinized deeply.

Governments and Defense departments have long required that all private software they buy be built locally. Now, as software supply chain concerns rise, compliance mandates requiring components of critical software exclude code from adversarial geographies.

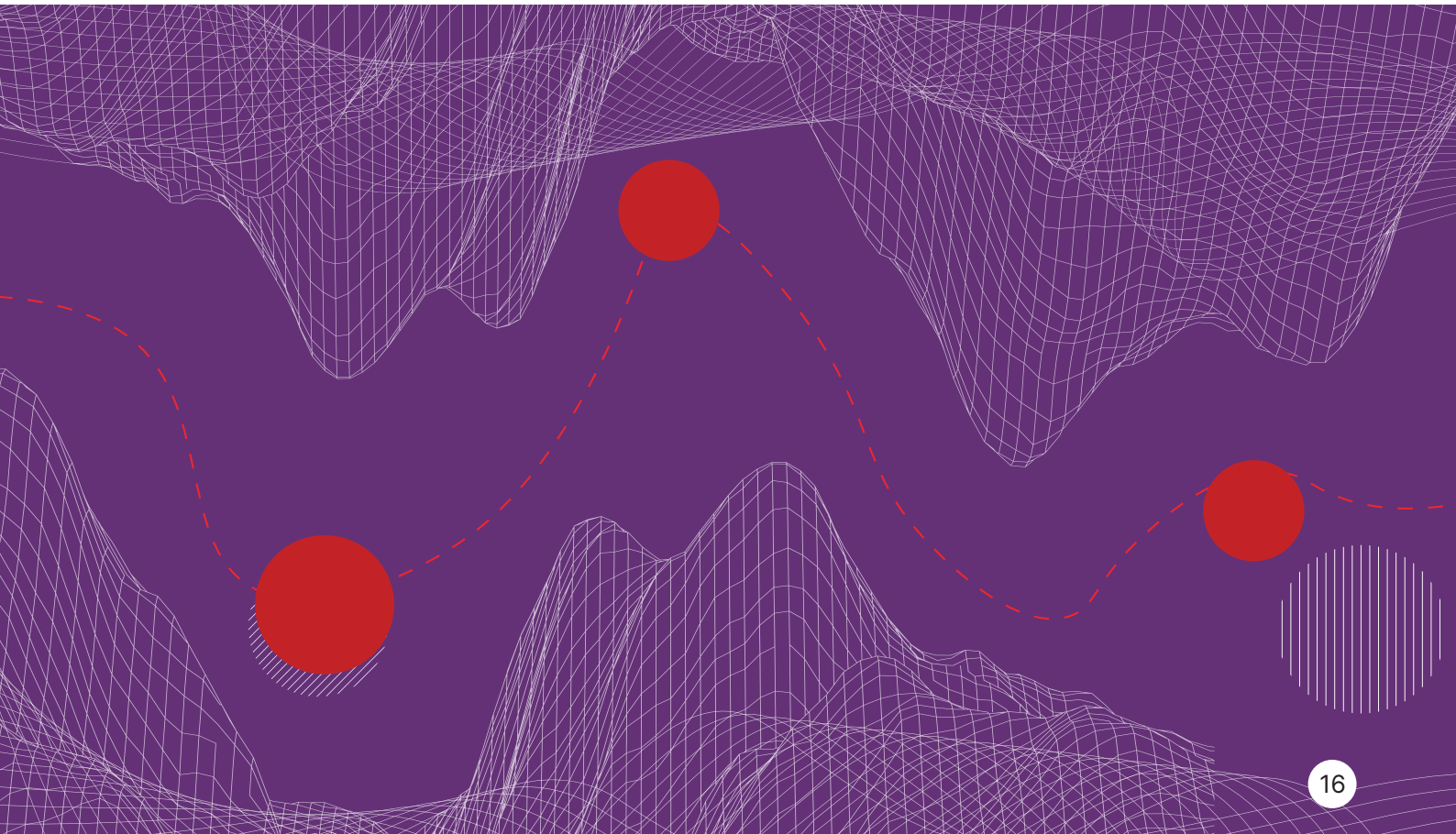
Geo-provenance constraints are particularly hard on open-source components, given the complete lack of control over geo-commits and geo-contributions. However, a pattern is emerging – the more critical the software application, more is the concern around the geo-provenance of both the open-source and private code it embeds.

A typical mid-size application can have contributions from several countries.

Quick takeaway

United States contributors commit more code to open-source projects than those from any other country, with Russia following closely. However, a notable 20% of American contributors choose to remain anonymous, twice the ratio of Russian contributors and three times that of Chinese contributors.

Country	% Commits
United States	34%
Russia	13%
Canada	9%
United Kingdom	7%
Brazil	6%
Germany	3%
China	1%
New Zealand	1%



Global Software Supply Chains Embed Geopolitical Risks

The COVID-19 pandemic served as a stark wake-up call, exposing the critical importance of software supply chain security on a global scale. As geopolitical tensions escalate and software underpins increasingly vital systems, the provenance of our code has become a matter of paramount national and economic security. This heightened awareness has spurred a wave of regulations aimed at fortifying global software security protocols.

> "Cars today have cameras, microphones, GPS tracking, and other technologies connected to the internet. It doesn't take much imagination to understand how a foreign adversary with access to this information could pose a serious risk to both our national security and the privacy of U.S. citizens. To address these national security concerns, the Commerce Department is taking targeted, proactive steps to keep PRC and Russian-manufactured technologies off American roads,"
>
> **U.S. Secretary of Commerce Gina Raimondo, Sep 23, 2024**

A prime example is the recent **Notice of Proposed Rulemaking (NPRM)** issued by the U.S. Department of Commerce's Bureau of Industry and Security (BIS) on September 23, 2024. This significant measure would prohibit the sale or import of connected vehicles containing specific hardware and software components or those components sold separately, with a sufficient nexus to the People's Republic of China (PRC) or Russia. The prohibitions on software would take effect for Model Year 2027, and the prohibitions on hardware would take effect for Model Year 2030 or January 1, 2029, for units without a model year.

In parallel, the **EU New Product Liability Directive** adopted on October 10, 2024, shifts liability to manufacturers of products included in defects even when they are compromised by components they source, putting a new focus on what software producers choose to include in their products.

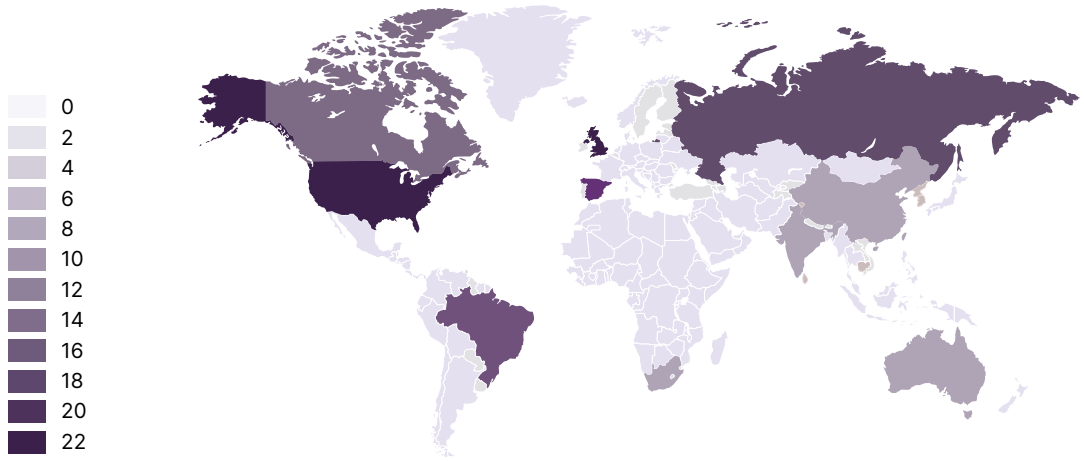
What's the Lineage of Open-source Software?

An analysis of over 15 million commits tied to enterprise applications reveals the distribution of contributions from top countries. The United States and Russia together account for nearly half of all open-source contributions, illustrating the challenge of achieving a Russia-free open-source ecosystem.

Country	Commits*	% Commits
United States	5,134,722	34%
Russia	1,910,842	13%
Canada	1,320,662	9%
United Kingdom	1,037,009	7%
Brazil	982,109	6%
Germany	408,704	3%
China	216,552	1%
New Zealand	155,551	1%

***Total Commits: 15,250,139**

Code-commit Contributions by Country



Open-source Contributors are Often Anonymous

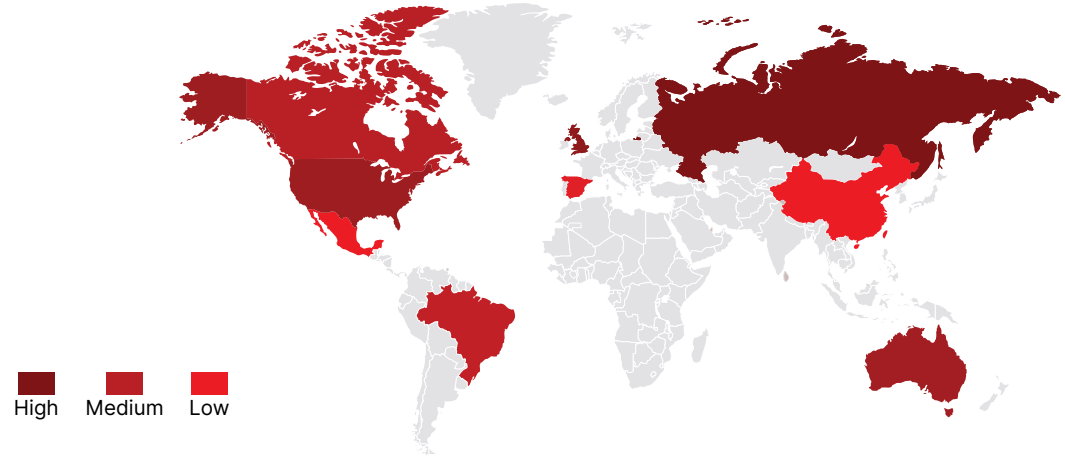
Enterprises building private software, startups creating new Intellectual Property (IP), and software contractors writing software for their customers routinely contribute software under their names. In fact all software commits in most serious enterprise software development shops can only come from verified developers committing code from secure, attested machines.

Open-source developers work from wherever they want, use personal devices, and frequently use anonymous and unverified accounts. They often choose to remain anonymous, which poses a higher risk than known, authenticated open-source contributors. The number of unknown contributors from Russia is about half that of the United States, while unknown Chinese contributors are about a third of the U.S. figure.

Top 10 Unknown and hence risky contributions come from the following countries:

Country	Known Contributors	Unknown Contributors	%Known	%Unknown
United States	15,051	3,957	79.2%	20.8%
Australia	1,139	264	81.2%	18.8%
Canada	5,329	997	84.3%	15.7%
Brazil	6,189	1,070	85.3%	14.7%
Great Britain	5,071	783	86.6%	13.4%
Spain	5,071	783	86.6%	13.4%
Germany	9,272	1,392	86.9%	13.1%
Russia	16,141	2,178	88.1%	11.9%
Japan	1,519	138	91.7%	8.3%
China	4,002	286	93.3%	6.7%

Top 10 Countries with Hidden Risks from Unknown Open-source Contributors



Lineaje Open-source Crawler:
The Lineaje Open-source Crawler autonomously gathers the geo-provenance of all open-source components used, tracking down to individual commits and authors. This enables comprehensive control of your entire software supply chain by geographic location.

CHAPTER 5

Open-source Is Not Well-maintained

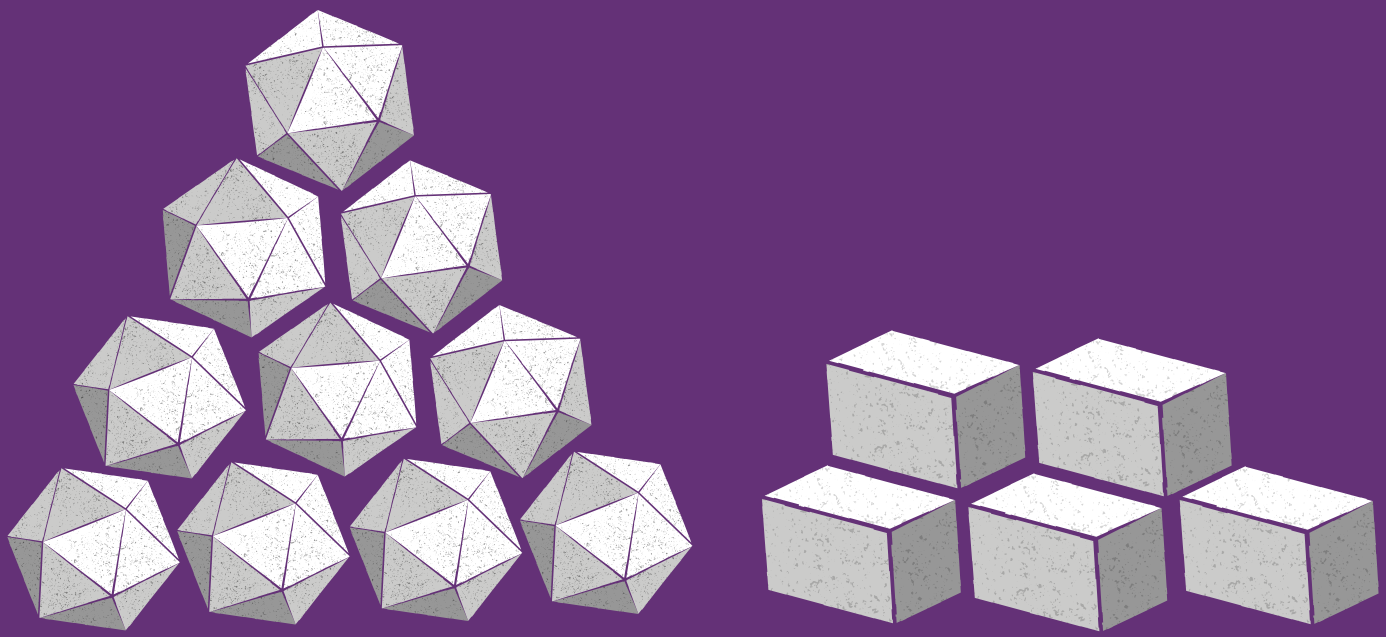
Open-source software, fueled by a global community of developers, has become a critical catalyst for accelerating software development and digital transformation in enterprises. The resulting efficiencies and cost savings are estimated to exceed a staggering **$450 billion annually**. More than **90%** of modern applications embed open-source components.

As successful open-source projects grow, they require increasing maintenance efforts. The exciting work of building new features gives way to the mundane but crucial task of fixing and maintaining existing code. Over time, fewer developers shoulder the burden of maintaining expanding codebases as others move on to new projects.

Quick takeaway

67.6% of open-source components used by organizations are not well-maintained, leaving critical software applications vulnerable to **security breaches, performance issues, and costly downtime.**

Without active open-source management, organizations are exposed to significant risks that can disrupt operations and damage their reputation.



Embracing Innovation, Ignoring Maintenance: Enterprise Reliance on Open-source Software

With over 90% of modern applications embedding open-source code, enterprises heavily rely on open-source components to drive innovation and digital transformation. This dependency results in estimated annual savings exceeding **$450 billion.**

Open-source development is often unpaid and driven by volunteers passionate about innovation. As projects grow, the maintenance burden increases, diverting focus from new features. Maintaining and fixing code becomes increasingly complex, falling on a shrinking pool of developers even as the total lines of code expand with each version. Unmaintained open-source components can deteriorate quickly, akin to milk, not wine. Critical dependencies may become obsolete as developers shift focus to new trends, leaving vital projects abandoned and risking critical enterprise applications that embed them.

Open-source Maintenance efforts are rarely prioritized and often do not match Enterprise standards.

Enterprise enhancement requests further dilute innovation, challenging the premise that dependencies accelerate innovation.

While some open-source projects successfully transition, many struggle to maintain the same level of innovation and support as required by commercial applications. The maintenance burden inevitably rises with wider adoption, further detracting from innovation.

Just as startups must evolve their products for enterprise deployment, so must successful open-source projects. However, unlike commercial products, open-source software lacks dedicated support teams or contractual obligations, relying instead on engineers with innovative ideas and the skills to bring them to life.

Measuring Maintainability of Open-source Software

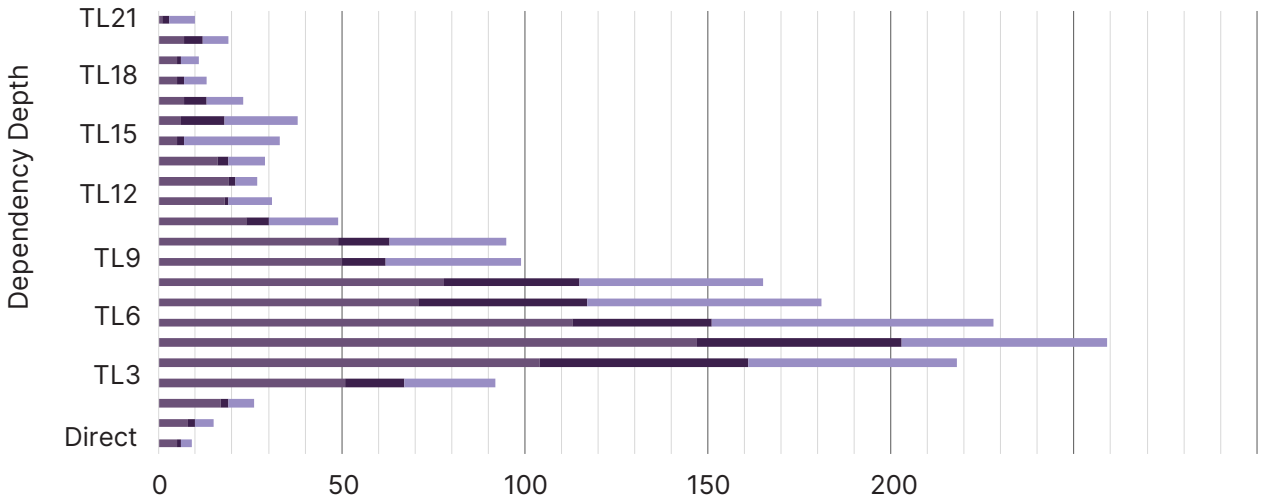
Tech debt piles up quickly in software development. Vulnerabilities are being discovered earlier and faster. To keep open-source software secure and usable by enterprises, open-source software must also be maintained. In an era where software is frequently updated hundreds of times a day, it is hard to argue that unchanging software is well-maintained.

Lineaje assesses the maintainability of applications and their dependencies automatically. By default, we apply the following definitions:

	Criteria
Unmaintained	No fix in the last 2 years
Well maintained	Fix available in last 6 months
Maintained/ Grey OSS	Fixed in last 6 months to 2 years

The distribution of a typical medium-complexity application is shown below.

OSS Component Distribution



	Direct	TL1	TL2	TL3	TL4	TL5	TL6	TL7	TL8	TL9	TL10	TL11	TL12	TL13	TL14	TL15	TL16	TL17	TL18	TL19	TL20	TL21
Unmaintained	5	8	17	51	104	147	113	71	78	50	49	24	18	19	16	5	6	7	5	5	7	1
Gray OSS(6mn-2yrs)	1	2	2	16	57	56	38	46	37	12	14	6	1	2	3	2	12	6	2	1	5	2
Well Maintained	3	5	7	25	57	56	77	64	50	37	32	19	12	6	10	26	20	10	6	5	7	7

Count of Components

	Criteria	Count	%
Unmaintained	No fix in the last 2 years	806	48.3%
Well maintained	Fix available in last 6 months	541	32.4%
Maintained/ Grey OSS	Fixed in last 6 months to 2 years	323	19.3%

Over 67.6% of open-source components used by organizations are not well-maintained, leaving critical software applications vulnerable to **security breaches, performance issues, and costly downtime.** Without active open-source management, organizations are exposed to significant risks that can disrupt operations and damage their reputation.

Lineaje Open Source Manager automatically segments your open-source into unmaintained, maintained, and well-maintained open-source depndencies and drives workflows and remediation that is appropriate for each category of open-source.

CHAPTER 6

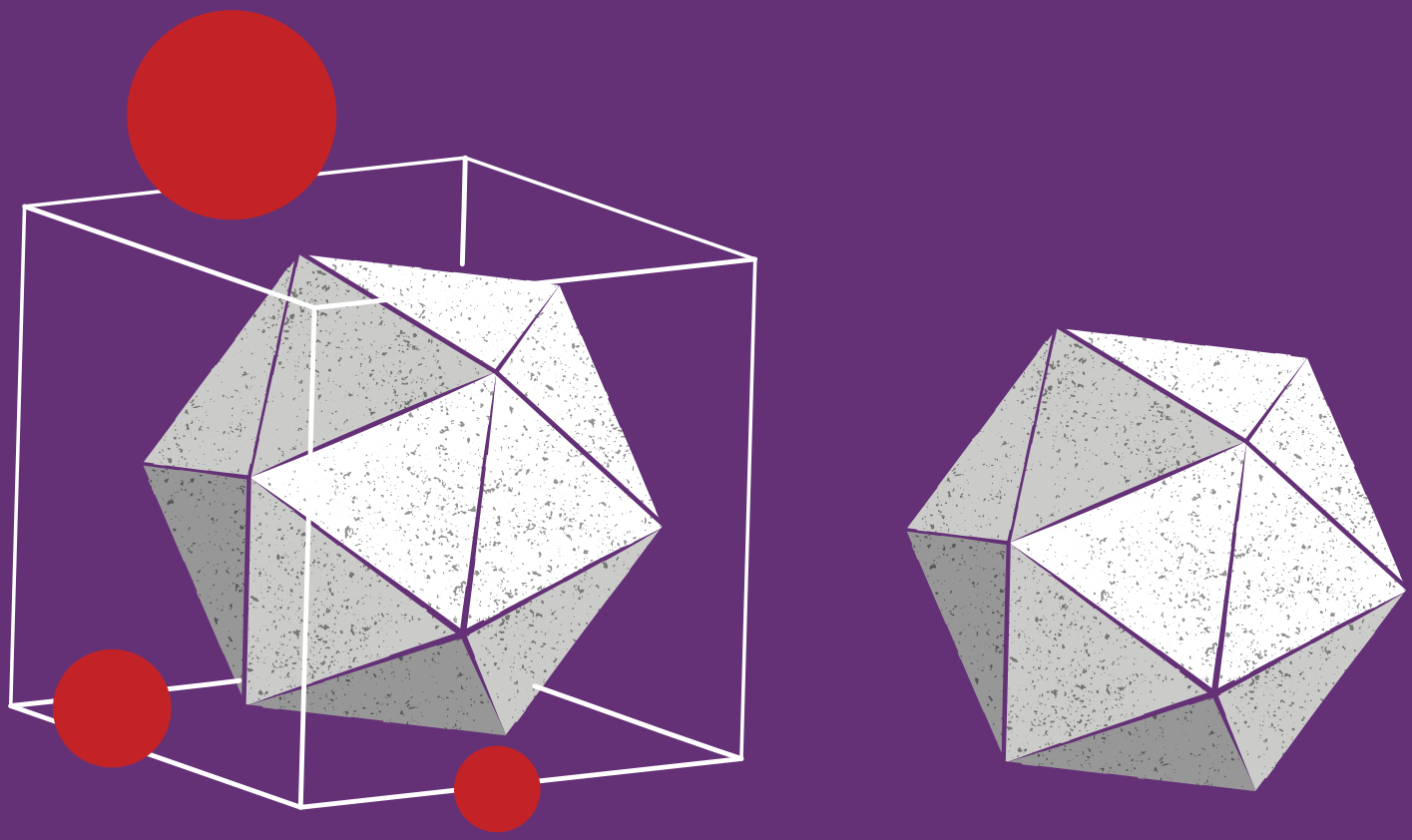
Secure Open-source Components Come from Stable Code Built by Mid-sized Teams

Rapidly changing open-source components while offering new features inherently introduces more vulnerabilities and risks. This is because each update brings the potential for new bugs, compatibility issues, and unforeseen interactions with existing code. On the other hand, unmaintained components become increasingly vulnerable over time due to the lack of security patches and bug fixes for newly discovered exploits. Hence, there is a fine balance between Innovation that enhances Risk and Maintenance that enhances Security.

Additionally, open-source projects of all sizes exist, with contributors varying from a handful to large teams. Just like enterprise teams, small, tightly-knit open-source teams deliver a lower vulnerability open-source project than those with larger teams.

Quick takeaway

Well-maintained open-source is 1.8 times more vulnerable than unmaintained open-source, and mid-sized teams represent the least risky projects.



The “Well-maintained” Paradox

Enterprise software developers know this well – software that changes a lot with every version is more buggy. The same is true for open-source software.

Lineaje measures vulnerabilities, code quality, security posture, and multiple other quality dimensions of open-source it assesses automatically. We arrive at a set of paradoxical results:

> Well-maintained open-source software is 1.8 times more vulnerable than unmaintained open-source software.

- Well-maintained components show the highest vulnerability rate, suggesting that active development and frequent updates might inadvertently introduce more security issues.
- Both maintained and Unmaintained open-source software projects have lower vulnerability rates, suggesting that open-source developers try to leave projects in a good state even as they move on.

Vulnerability Distribution by Maintenance Status

Maintenance Status	Vulnerable Components %
Well-maintained	3.7%
Moderate Updated	2.0%
Unmaintained	2.0%

Open-source is Dominated by Small Teams Creating Vulnerable Software

A related factor is the size of teams that work on an open-source package. About 2/3rd of open-source software is created by small teams of less than 10 contributors. These teams band together to deliver an innovative capability. However, this suggests that limited resources and oversight in small teams may lead to security oversights and maintainability issues over time.

> About 2/3rd of all open-source software is created by teams of less than 10 contributors and has 3 times the vulnerabilities of open-source created by mid-sized teams.

The "Goldilocks zone" of 50 contributors: About a fifth of all open-source software is created by mid-sized teams. These teams represent open-source code with the best security and quality metrics. The vulnerability rate in their open-source software is the lowest – at 1.0% reflecting an optimal balance between coordination overhead and sufficient resources for security oversight.

Diminishing returns of large teams: Projects with over 50 contributors are fewer and show an increase in vulnerability rate (1.4%) compared to mid-sized teams. Large projects are more complex, and adding more contributors doesn't improve security and leads to more vulnerabilities.

Vulnerability Distribution by Contributor Count and Supplier Type

Contributor Count	Total Packages	% of Packages	Vulnerable Components %
<10 Contributors	4,591	66.7%	3.3%
11-50 Contributors	1,348	19.6%	1.0%
>50 Contributors	931	13.5%	1.4%

Lineaje continuously monitors your open-source packages for vulnerabilities, automatically alerting you to new issues and providing a minimal-effort fix plan built by Lineaje AI.

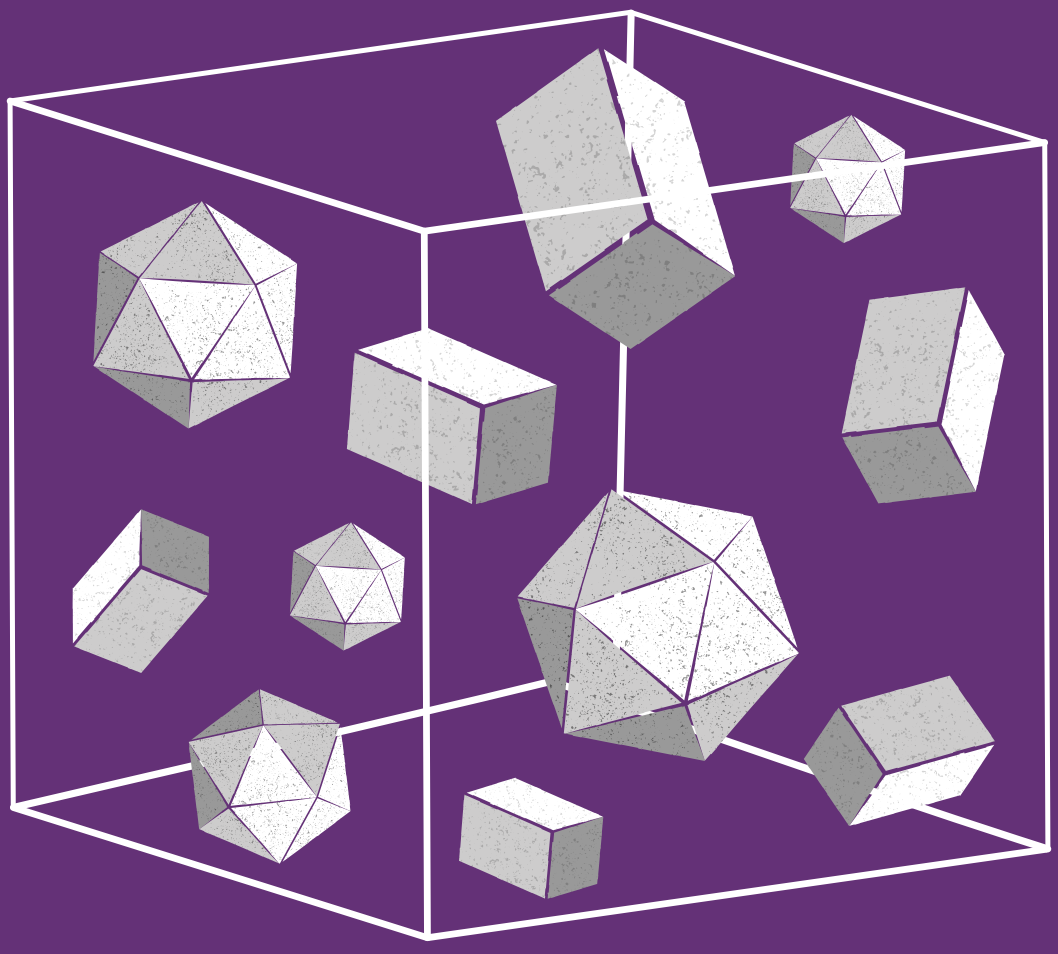
CHAPTER 7

Version Sprawl! Open-source Usage in Enterprises is Unmanaged

Once an open-source package is used within an enterprise, it is utilized multiple times by developers to accelerate development. As more and more developers use the same package, they all use the current version available to them. As they update the components they build, they may update to later versions. These changes are not synchronized, given most organizations do not manage their open-source dependencies. This creates a version sprawl – the fact that multiple versions of the same open-source package exist in one organization's business applications complicating reachability, vulnerability prioritization, and remediation.

Quick takeaway

Version Sprawl! More than **15%** of components have multiple versions in a single application.



Multiple Versions of the Same Open-source Component Exist in the Same Application

Business applications and open-source packages are built modularly. Different developers work on each module. They may use the same components in their code, but they may not update and upgrade these versions simultaneously. So, multiple versions of the same component exist simultaneously in the same package at different depths of the transitive dependency chain.

Version Sprawl complicates software updates, reachability analysis, and compatibility testing, increasing software complexity dramatically.

Each version must be upgraded separately, and each version must be considered independently. Each version potentially has a different compatibility matrix, creating a cascading set of multiple versions of other components.

Once used in an application multiple times, developers rarely go back and synchronously upgrade to "Approved" versions. The primary reason is a complete lack on investment in active management of open-source software used in an enterprise beyond some basic Vulnerability Management.

Version Sprawl Impact

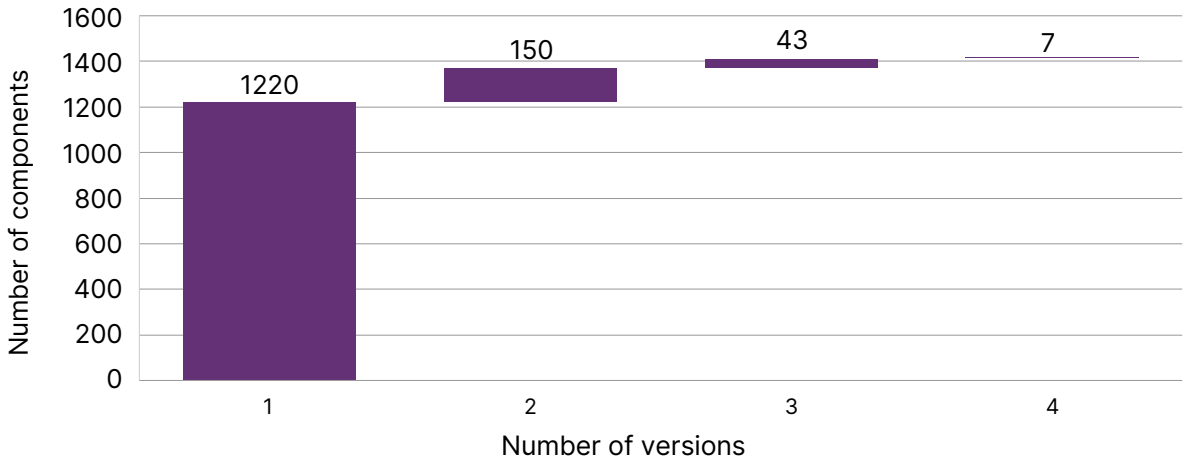
Analysis of applications deployed in enterprises showcases a pattern. About 14% of all open-source components are used more than once in applications, and version sprawl increases exponentially across multiple applications.

For example, enterprises have had a hard time getting past the Log4jShell vulnerability. A key reason is the lack of management control over open-source used by developers. Version sprawl is a leading indicator of how badly maintained any application or its dependency is.

The bigger the version sprawl in your dependencies, the more costly they are to maintain, and the more insecure your application.

The bigger the version sprawl in your dependencies, the more costly they are to maintain and the more insecure your application. The version sprawl of a medium-complexity application is shown below.

Multiple Versions of Same Component



Lineaje Assessment Engine automatically detects version sprawl in private, third-party, and open-source dependencies in you application and recommends the most compatible and secure application and version with a click.

CHAPTER 8

Open-source Promotes Unconstrained Polyglotism

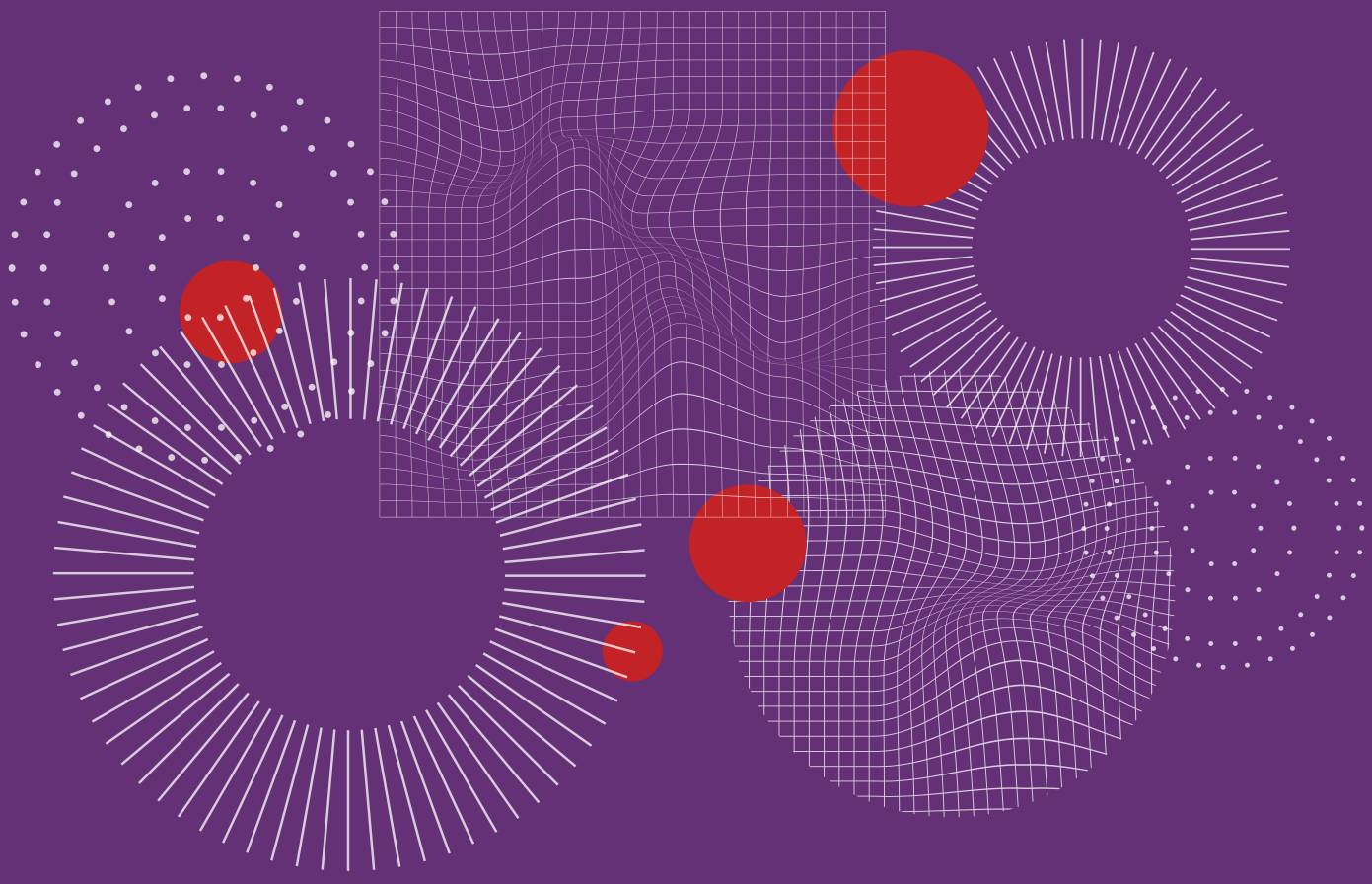
Most software engineering organizations write private/first-party code in less than 10 languages. Expanding the list of languages they support requires investment in incremental resources and tools and is a careful decision.

However, when open-source dependencies are included inside private code, enterprises do not seem to consider the languages of their third-party and open-source dependencies and the implications of adding memory-unsafe languages – making their applications more insecure.

Developers cannot fix what they don't know. This language proliferation through open-source dependencies tests the limits of engineering teams. Their ability to find and fix issues in their open-source dependencies is limited to languages they know. Even their current tools, like SCA, provide support for a handful of first-party languages, and hence their "discovery" and "reachability" assertions only apply to a small subset of dependencies.

Quick takeaway

A mid-sized application, on average, pulls in **1.4 million** lines of code in **139** languages and drags in more risky memory-unsafe languages.

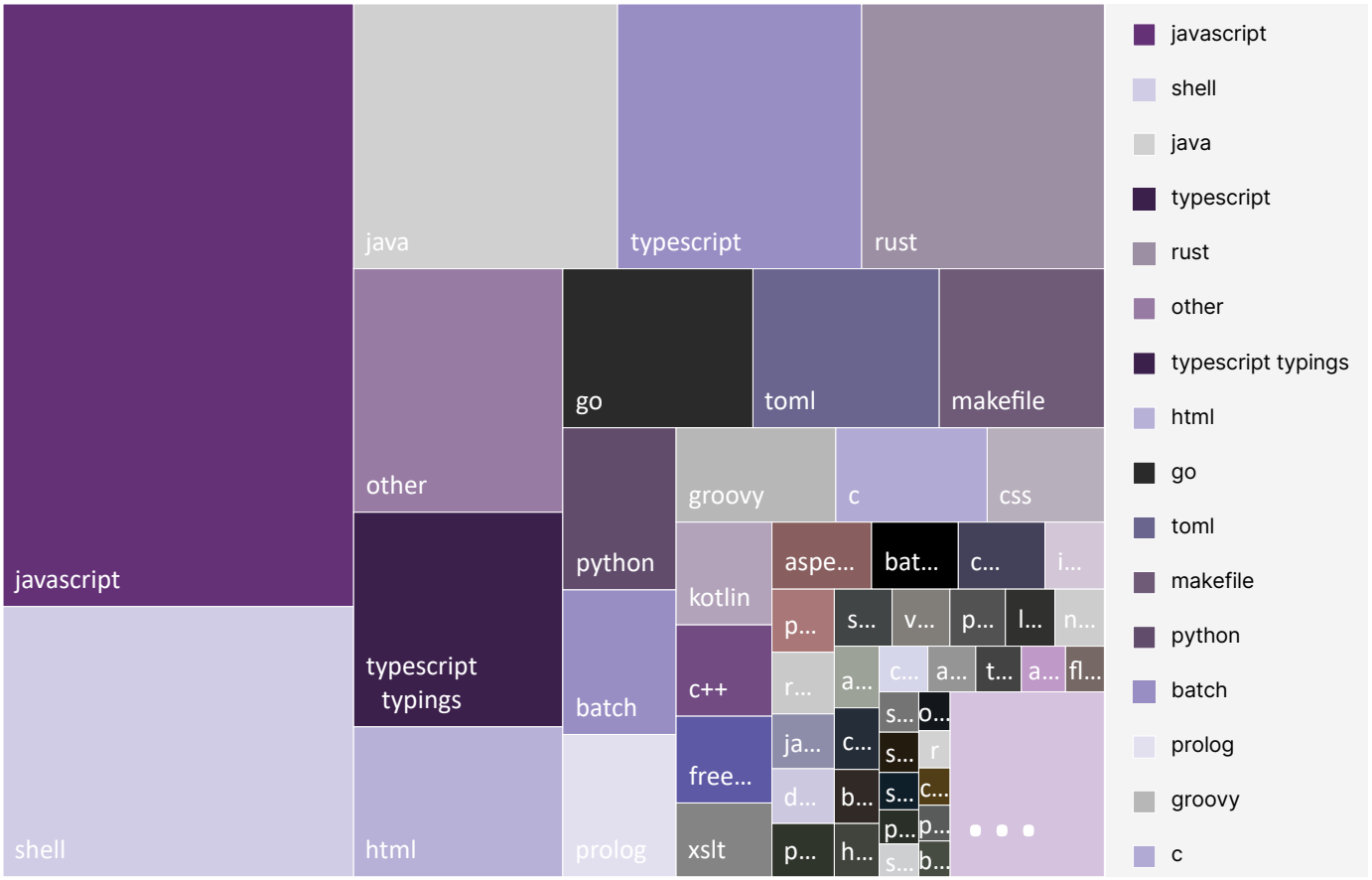


Unmanaged Language Proliferation is Insecure by Design

A typical mid-sized application analyzed by Lineaje has **139** languages in its open-source dependencies. These dependencies pull in **1.4 million** lines of code! The cost of delivering these dependencies by private developers with a productivity of 50 lines of code a day would be more than 100 person-years.

Choosing to be More Secure: Using Secure-by-Design Memory-safe Dependencies

Languages Used to Build Your OSS



Beyond developer productivity, language selection plays a key role in software security. Memory-safe languages include Rust, Go, C#, Java, Swift, Python, and JavaScript. Languages that are not memory-safe include C, C++, and assembly.

How big an issue is memory safety? In April 2023, the National Security Agency (NSA) joined the Cybersecurity and Infrastructure Security Agency (CISA) and U.S. and international partners to publish a report, **The Case for Memory-safe Roadmaps.** They presented the following statistics arguing that enterprises actively move to memory-safe languages:

- About 70% of Microsoft's Common Vulnerabilities and Exposures (CVEs) are memory safety vulnerabilities (based on 2006-2018 CVEs).
- About 70% of vulnerabilities identified in Google's Chromium project are memory safety vulnerabilities.
- In an analysis of Mozilla vulnerabilities, 32 of 34 critical/high bugs were memory safety vulnerabilities.
- Based on analysis by Google's Project Zero team, 67% of zero-day vulnerabilities in 2021 were memory safety vulnerabilities.

A key recommendation of the report is for enterprises to phase out memory-unsafe languages from their applications.

Compiled vs Interpreted Languages Change Your Security Posture

Compiled languages are generally considered more secure than interpreted languages because they translate code into machine code before execution. This allows for stricter checks and optimizations during compilation, resulting in faster performance and better control over memory management. Interpreted languages execute code line-by-line at runtime, making them more vulnerable to security exploits like code injection attacks due to the exposure of source code during execution.

- **Examples of compiled languages include** C, C++, C#, Rust, Erlang, Go, etc.
- **Examples of Interpreted languages include** Python, JavaScript, PHP, Ruby, etc.

A bias towards using compiled languages creates less exploitable applications.

Lineaje AI Plan can discover all languages used in your dependencies and in your code and make recommendations on language “substitution” to create more secure applications.

CHAPTER 9

Open-source Vulnerability Fixing is Complicated & Broken

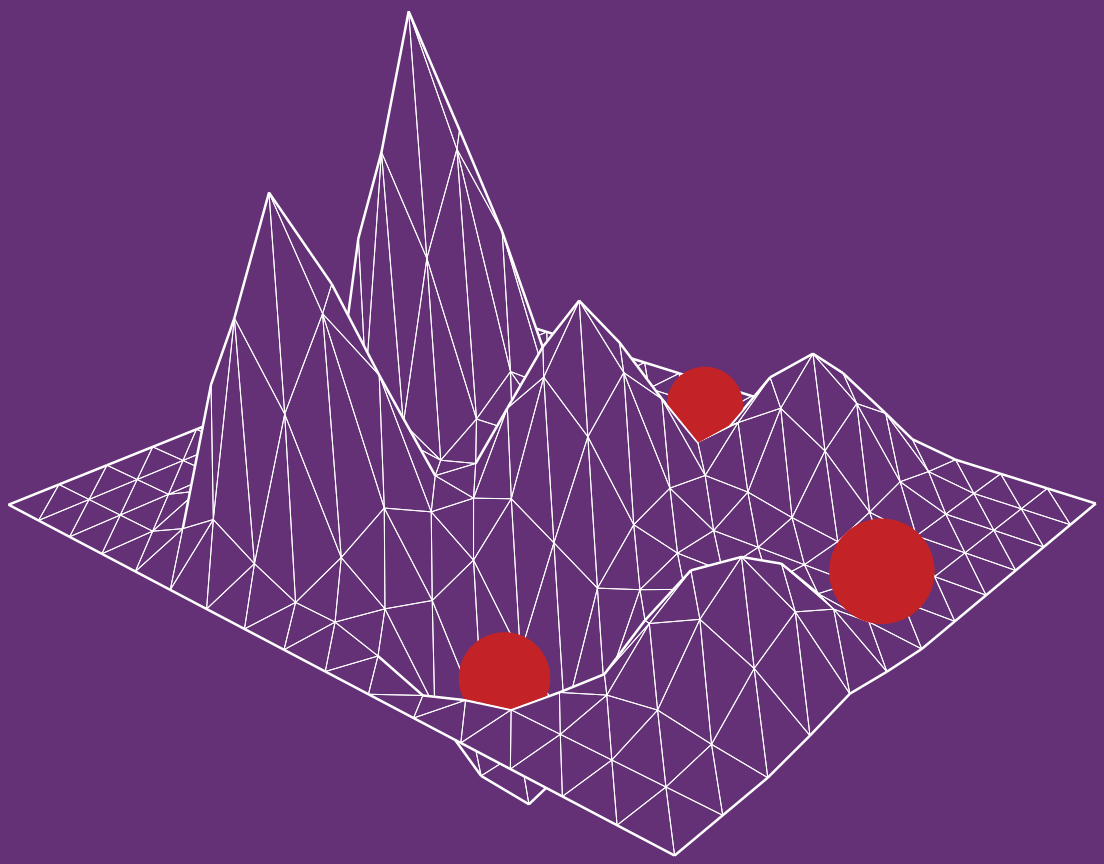
Software Developers are master assemblers of software. AppSec tools live in a DIY world and make terrible assumptions. Just like it is foolhardy to expect a great DIY carpenter to fix a complicated smart TV, expecting your developers to fix every security flaw in software they didn't build is foolhardy. Even with the best tools, the carpenter cannot open up and take a chip out of the motherboard and replace it. The carpenter can certainly fix the TV stand! Today's software is more complicated than a smart TV.

As applications embed more open-source code, the potential attack surface expands dramatically. AI-powered tools like Google's Big Sleep are revolutionizing vulnerability discovery, uncovering flaws previously hidden from traditional scanners. This means organizations will face an unprecedented surge in the number of vulnerabilities requiring immediate attention.

Current approaches to vulnerability prioritization do not consider developer remediation challenges like software structure, dependency graphs, the complexity of direct dependency upgrades versus transitive dependency upgrades, component interaction intensity, rate of code changes and their reachability changes, or container rebuild rate, making it impossible for organizations to get on top of their vulnerability backlog and overwhelming developers with sub-optimal AppSec priorities.

Quick takeaway

95% of all vulnerabilities come from your open-source dependencies. Knowing which your developers can fix, and which they should not, eliminates at least **50%** of vulnerability fix effort.



Vulnerability Prioritization is Mismatched with Vulnerability Remediation

Various AppSec scanners, including deep vulnerability scanners, find vulnerabilities and weaknesses in the components they scan. These vulnerabilities are prioritized by severity, exploitability, and reachability and then assigned to developers to fix.

Our software changes continuously. Modern software development runs through a CI/CD pipeline – **Continuous** Integration**/Continuous** Deployment.

Lineaje has customers that make a million updates a day to production software. These changes drag in new components, update existing ones, and create new code paths – creating a need for reprioritization of vulnerabilities.

> "We cannot achieve business goals if we are expected to meet AppSec dmands on their timelines" - **Big Financial AppSec Leader**

> "My desired outcome is to automate the process of aggregating, consolidating, contextualizing, attributing, and monitoring vulnerabilities" - **Cutting-edge AppSec Leader**

AppDev leaders focused on digital innovation are incurring significant overhead from vulnerability remediation efforts impacting their innovation and business goals. Reactive scanning and prioritization by AppSec is disruptive, and insistence on fixing by priority is disruptive and ignores the size of the fix effort. Prioritizing large, complex fixes creates a bottleneck in enterprise resources that deprioritizes simpler, highly impactful security improvements.

Even Simple Applications Pull in Complex Open-source Transitive Dependencies

Lineaje's analysis routinely shows that software supply chain trees are up to 60 levels deep due to the simple fact that open-source components often embed other open-source components, creating deeply nested dependencies. However, even updating simple applications is complex. To illustrate this concept, we use a simple application in a widely distributed product.

> Even simple open-source direct dependencies pull in an additional 600% of opaque transitive dependencies that developers and AppSec cannot see.

Number of components by Dependency Level

Project name	Depth level	Open-source Components	Open-source Well Maintained	Open-source Maintained	Open-source Unmaintained	Third Party	Private
tc_service	0	0	0	0	0	0	2
Direct Dependecies	1	36	18	7	9	0	0
Transitive Dependencies	2	85	41	17	27	0	0
Transitive Dependencies	3	73	31	15	26	0	0
Transitive Dependencies	4	47	17	17	13	0	0
Transitive Dependencies	5	4	2	0	2	0	0
Transitive Dependencies	6	1	0	0	1	0	0
	Total	246	44%	23%	32%	0	1%

Tc_service has 2 private components that use 36 direct open dependencies. Application developers selected and included these in their private components. However, these 36 components pull in 208 additional open-source components. These 208 components are typically not from developers who built the direct dependencies and are opaque to the developers who included them in their code.

Software Dependency Structure Impacts Vulnerability Fix Complexity

Tc_service has 73 vulnerabilities. 13 of them are in direct components and 60 in transitive components.

Upgrading transitive dependencies will frequently break the direct dependency. Developers then have to perform the complex task of fixing a dependency they did not build.

Open-source Vulnerabilities

Project name	Depth level	Open-source Components	Critical Vulnerabilities	High Vulnerabilities	Medium Vulnerabilities	Low Vulnera-bilities	Unkown Vulnera-bilities
tc_service	0	0	0	0	0	0	0
Direct Dependecies	1	13	2	9	1	0	1
Transitive Dependencies	2	16	3	7	6	0	0
Transitive Dependencies	3	36	1	13	21	1	0
Transitive Dependencies	4	7	0	3	3	0	1
Transitive Dependencies	5	1	0	1	0	0	0
Transitive Dependencies	6	0	0	0	0	0	0
	Total	73	6	33	31	1	2

When we use a new version of a direct dependency, it will pull in transitive dependencies it is certified and tested against. Upgrading transitive dependencies will frequently break the direct dependency. Developers then have to perform the complex task of fixing a dependency they did not build.

Not All Vulnerability Fixes Can be Applied Easily

Upgrading transitive dependencies-like many AppSec tools do-will frequently break the direct dependency needlessly increasing developer effort.

Open-source developers do provide fixes for these vulnerabilities. Lineaje tools automatically discover the fixes revealing that 71 of the 73 have versions that fix these vulnerabilities. Two medium vulnerabilities do not have fixes.

Number of Fixed vulnerabilities in later versions

Project name	Depth level	Open-source Vulnerability Fixes	Critical Vulnerabilities	High Vulnerabilities	Medium Vulnerabilities	Low Vulnera-bilities	Unkown Vulnera-bilities
tc_service	0	0	0	0	0	0	0
Direct Dependecies	1	13	2	9	1	0	1
Transitive Dependencies	2	14	3	7	4	0	0
Transitive Dependencies	3	36	1	13	21	1	0
Transitive Dependencies	4	7	0	3	3	0	1
Transitive Dependencies	5	1	0	1	0	0	0
Transitive Dependencies	6	0	0	0	0	0	0
	Total	71	6	33	29	1	2

These transitive dependencies with fixes can potentially break the direct open-source dependencies they are embedded in. Enterprise developers cannot recertify their direct dependencies with new transitive dependency versions, nor do their existing tools give them clear dependency information.

Software Structure-based Vulnerability Fixes Are “Almost” Free

A smarter approach is to upgrade direct open-source dependencies to compatible but fixed versions. This is a complex analysis but modern tools like Lineaje can do that using Lineaje AI.

Compatible Direct Dependency Patching

Number of Fixed vulnerabilities

Project name	Depth level	Open-source Vulnerability Fixes	Critical Vulnerabilities	High Vulnerabilities	Medium Vulnerabilities	Low Vulnera-bilities	Unkown Vulnera-bilities
tc_service	0	0	0	0	0	0	0
Direct Dependecies	1	10	2	6	1	0	1
Transitive Dependencies	2	13	2	6	4	0	1
Transitive Dependencies	3	27	0	9	17	1	0
Transitive Dependencies	4	5	0	3	2	0	0
Transitive Dependencies	5	2	0	2	0	0	0
Transitive Dependencies	6	0	0	0	0	0	0
	Total	57	4	26	24	1	2

20% of all vulnerabilities were eliminated by simple compatible upgrades of direct open-source dependencies without complex prioritization and dev effort.

The benefit of this approach is that 14 of the vulnerabilities (2 critical, 7 high) can be eliminated with a simple PR request with minimal effort from developers. This is about a 20% drop in vulnerabilities making the application more secure.

Incompatible Direct Dependency Patching Delivers Large Vulnerability Reduction

Multiple direct dependency changes are incompatible with each other and with private code that embeds them. Grouping all incompatible direct dependencies and retesting the entire set once is much more efficient.

Number of unfixed vulnerabilities

Project name	Depth level	Open-source Vulnerability Fixes	Critical Vulnerabilities	High Vulnerabilities	Medium Vulnerabilities	Low Vulnera-bilities	Unkown Vulnera-bilities
tc_service	0	0	0	0	0	0	0
Direct Dependecies	1	3	0	2	0	0	1
Transitive Dependencies	2	0	0	0	0	0	0
Transitive Dependencies	3	4	0	1	2	1	0
Transitive Dependencies	4	5	0	2	3	0	0
Transitive Dependencies	5	1	0	0	0	0	1
Transitive Dependencies	6	0	0	0	0	0	0
	Total	13	0	5	5	1	2

Applying 12 patches for 13 new “most secure” versions of the direct dependencies fixes 60 of the 73 vulnerabilities. This leaves 13 transitive vulnerabilities that are not fixed. These are “High Risk” fixes for vulnerabilities. Even though available, they have not been picked up by the direct open-source dependencies.

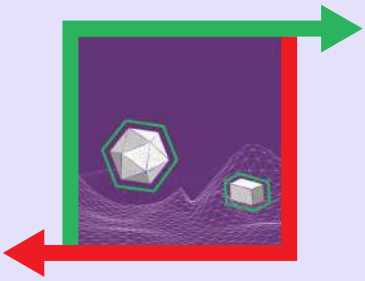
Enterprise developers picking these up risk expensive direct dependency recertification efforts. These should be taken up very carefully. A better approach is to backport these fixes to compatible versions of these transitive dependencies and use those backported versions.

Lineaje AI Remediate not only highlights compatible and incompatible versions of fixed components but also shows which incompatible, transitive upgrades your developers should not fix but backport.

Critical Facts About Lineaje You Can't Afford to Ignore

Pervasive

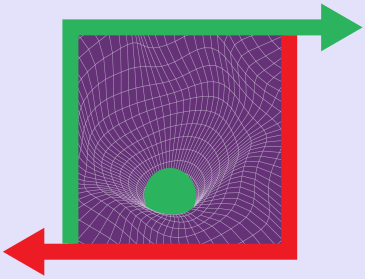
Open-source contributes two to nine times the code your developers write.



Lineaje Unified Scanner Hub can extend your AppSec tools to scan the source code and packages of ALL your open-source dependencies that existing AppSec and next-gen SCA tools cannot do on their own.

Deep

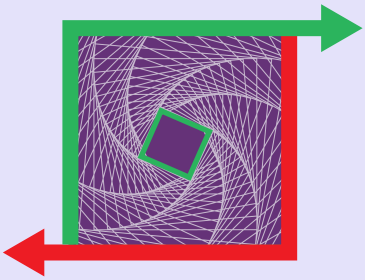
Open-source embeds **20-60** layers of components from dozens of open-source organizations assembled in a complex LEGO-like structure in a single dependency your developers include in your application.



Lineaje Dependency Crawler technology creates the deepest supply chain dependency tree in the industry.

Unattested

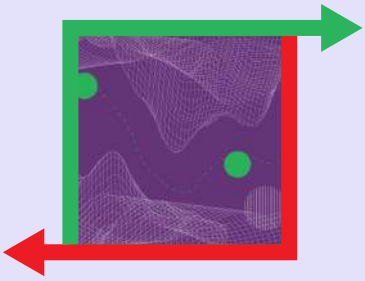
5%-8% of components in open-source dependencies of any application are unknown, tampered with or are of dubious-origin.



Lineaje Attestation Engine automatically attests to the integrity of each Open-source and private component in your application and alerts you on any component that is not fully trustable.

Global

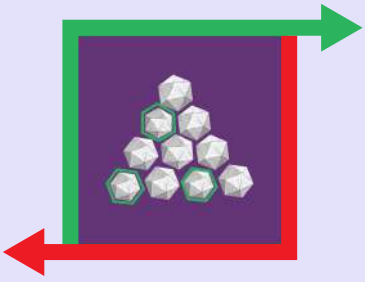
United States contributors commit more code to open-source projects than those from any other country, with Russia following closely.



Lineaje Open-source Crawler:
The Lineaje Open-source Crawler autonomously gathers the geo-provenance of all open-source components used, tracking down to individual commits and authors. This enables comprehensive control of your entire software supply chain by geographic location.

Anonymous

20% of American contributors choose to remain anonymous, twice the ratio of Russian contributors and three times that of Chinese contributors.

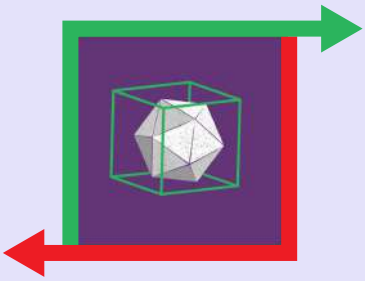


Lineaje AI Search:
A simple query in Lineaje AI Search can discover components with the most anonymous code written after a geopolitical event from a country involved in that geopolitical event.

Critical Facts About Lineaje You Can't Afford to Ignore

Unmaintained

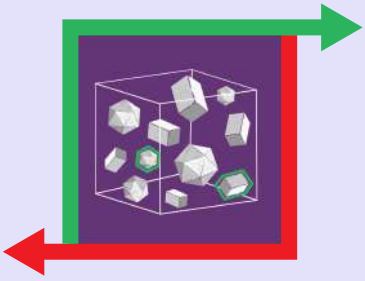
Shockingly, unmaintained open-source is less vulnerable than well-maintained open-source which is **1.8** times more vulnerable, and mid-sized teams represent the least risky projects.



Lineaje Open Source Manager automatically segments your open-source into unmaintained, maintained, and well-maintained open-source depndencies and drives workflows and remediation that is appropriate for each category of open-source.

Version Sprawl

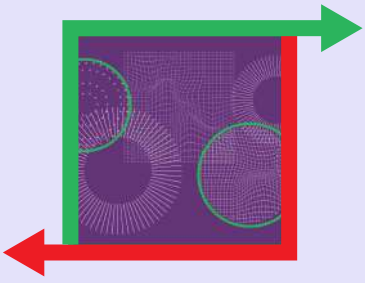
More than **15%** of components have multiple versions in a single application.



Lineaje Assessment Engine automatically detects version sprawl in private, third-party and open-source dependencies in your application and recommends the most compatible and secure version with a click.

Polyglot

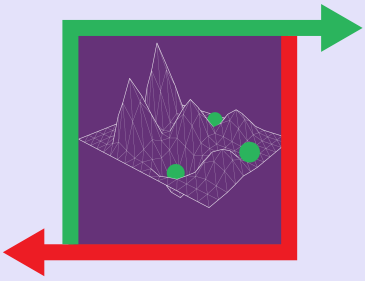
A mid-sized application, on average, pulls in **1.4 million** lines of code in **139** languages and drags in more risky memory-unsafe languages.



Lineaje AI Plan can discover all languages used in your dependencies and in your code and make recommendations on language "substitution" to create a more secure applications.

Vulnerable

95% of all vulnerabilities come from your open-source dependencies. Knowing which your developers can fix, and which they should not, eliminates at least **50%** of vulnerability fix effort.



Lineaje AI Remediate not only highlights compatible and incompatible versions of fixed components but also shows which incompatible, transitive upgrades your developers should not fix but backport.

Learn more about Lineaje

View another report, "What's in Your Open Source Software?"
An Approach to Enhance Software Supply Chain Security demonstrated by a deep analysis of the Apache Software Foundation

Download Today

Explore Lineaje's blog for valuble insights on software supply chain security, open-source management, and the latest industry developments.

Read More

See a demo to discover how Lineaje can help enhance your organization's security posture.

Schedule Demo

LINEAJE