LINEAJE

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

A LINEAJE DATA LABS REPORT

Table of Contents

Is Your Company the Next Troy?

Throughout history, from ancient Greek armies to modern warfare, investing in a secure supply chain has been a critical factor in military operations. The same is true for nearly every industry, as securing the supply chain is essential for operational success. The pandemic highlighted vulnerabilities in the United States physical supply chain, which led to significant disruptions. **Have companies recognized the problems lurking in their software supply chain? The software pandemic that showcases a vulnerable software supply chain is already upon us.**

With more software being assembled rather than built by developers, a reliable and secure software supply chain is now critical for our digital future. Open source software is now an intrinsic part of the software supply chain. Lineaje recognizes commercial applications where 99% of the software is open source.

Unfortunately, software producers have paid little attention to the robustness of their software supply chain. **It's not surprising to learn that most organizations lack a complete understanding of their software supply chains.**

In the Greek myth of the Trojan War, the soldiers defending the city of Troy brought what appeared to be a gift into their city walls. This became known as the Trojan horse, which ultimately led to the city's destruction. Today, developers make decisions on the dependencies they bring in. These decisions – what to source and from whom to source – are typically based on the utility of the dependency and its popularity. Without deep analysis, developers are essentially bringing in Trojan horses. Like the soldiers of Troy, your developers can become enamored by the beauty of the dependencies they bring in. What at first seems to be a gift may conceal harmful ingredients.

Do not blame your developers for dragging these Trojan horses in. Tools to analyze the deep and dynamic software supply chain have only recently been created. We created Lineaje Data Labs to provide that needed deep analysis tool for open source software supply chains. SBOM360 by Lineaje is the industry's first comprehensive software supply chain manager. The Apache Software Foundation (ASF), one of the most popular open source providers, represents the gold standard in open source software. This report assesses ASF's software and extends the findings to the broader open source world.

Another thing to keep in mind is that if Lineaje can assess the software supply chain of a globally distributed ASF, we can do it for your company as well – irrespective of whether you are a software producer or consumer. Using SBOM360, Lineaje Data Labs completed this analysis in about four weeks. We have introduced a fast, accurate, and crucial tool to the cybersecurity sphere. If you want your software portfolio assessed, contact us.

Happy reading!

Javed Hasan
Co-founder & CEO
Lineaje Inc.

Assessing Your Software Supply Chain

Over the last decade, the modern software supply chain has become more dynamic, complex, innovative, free, and global. Yet, it's been left entirely unmanaged.

Today, software involves tens of thousands of components, is built in thousands of independent build systems, and may be shipped or deployed in a hundred different ways. This creates significant challenges in understanding two fundamental questions about any product we buy: "What's in your Software?" and "How good is it?".

Software producers have embraced open source software much more enthusiastically over the last decade. Lineaje's analysis indicates that, on average, 70 to 90% of software comprises open source software. In some cases, over 99% of corporate software comprises open source software. However, this software is not adequately tracked, maintained, updated, or inventoried.

The lack of visibility into the software supply chain creates an unsustainable cycle of discovering vulnerabilities and weaknesses in software and IT systems, overwhelming organizations. The opacity of the software supply chain is exacerbating the problem.

Only software that is built securely can run securely. Current runtime security controls operated by software consumers fail to secure digital infrastructures. This is evidenced by continuous breaches and attacks even as the world quickly becomes more digital. We need a new approach – a focus on empowering and enabling software producers to build better software.

Only software that is built securely can run securely.

Like the Trojan Horse, the supply chain is created outside city limits.

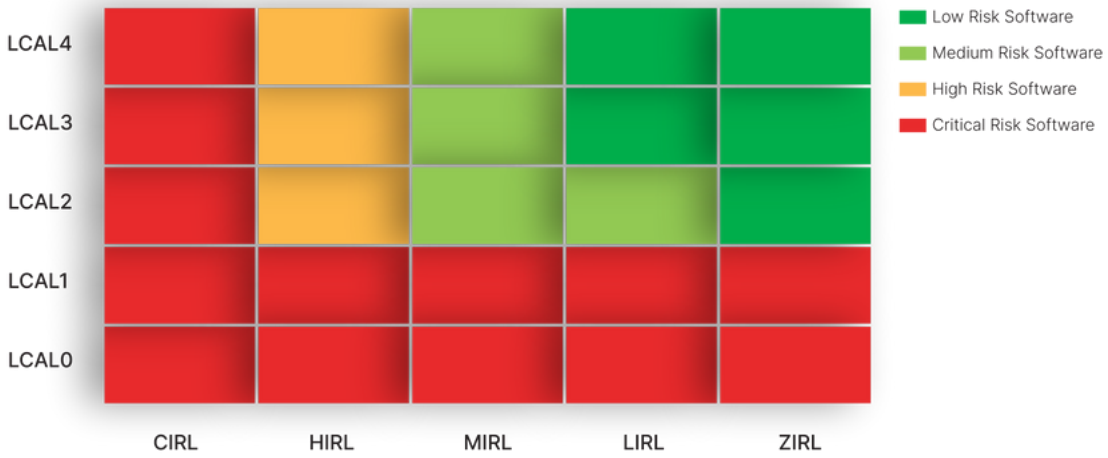
The software supply chain is not built in our CI/CD pipelines but is created to their left. The focus must be "left of shift-left". Organizations need tools like **SBOM360** to assess the integrity and risk of software sourced from the left of shift-left. This east-west risk vector runs straight into you and your customers' production systems, with the CI/CD security tools having no ability to assess either integrity or risk despite what your existing vendors may claim.

Lineaje's approach focuses on your software's DNA and produces a fully transparent, trusted software supply chain for your organization in four steps:

Organizations need tools like SBOM360 to assess the integrity and risk of software sourced from the left of shift-left. This east-west risk vector runs straight into you and your customers' production systems, with the CI/CD security tools having no ability to assess either integrity or risk despite what your existing vendors may claim.

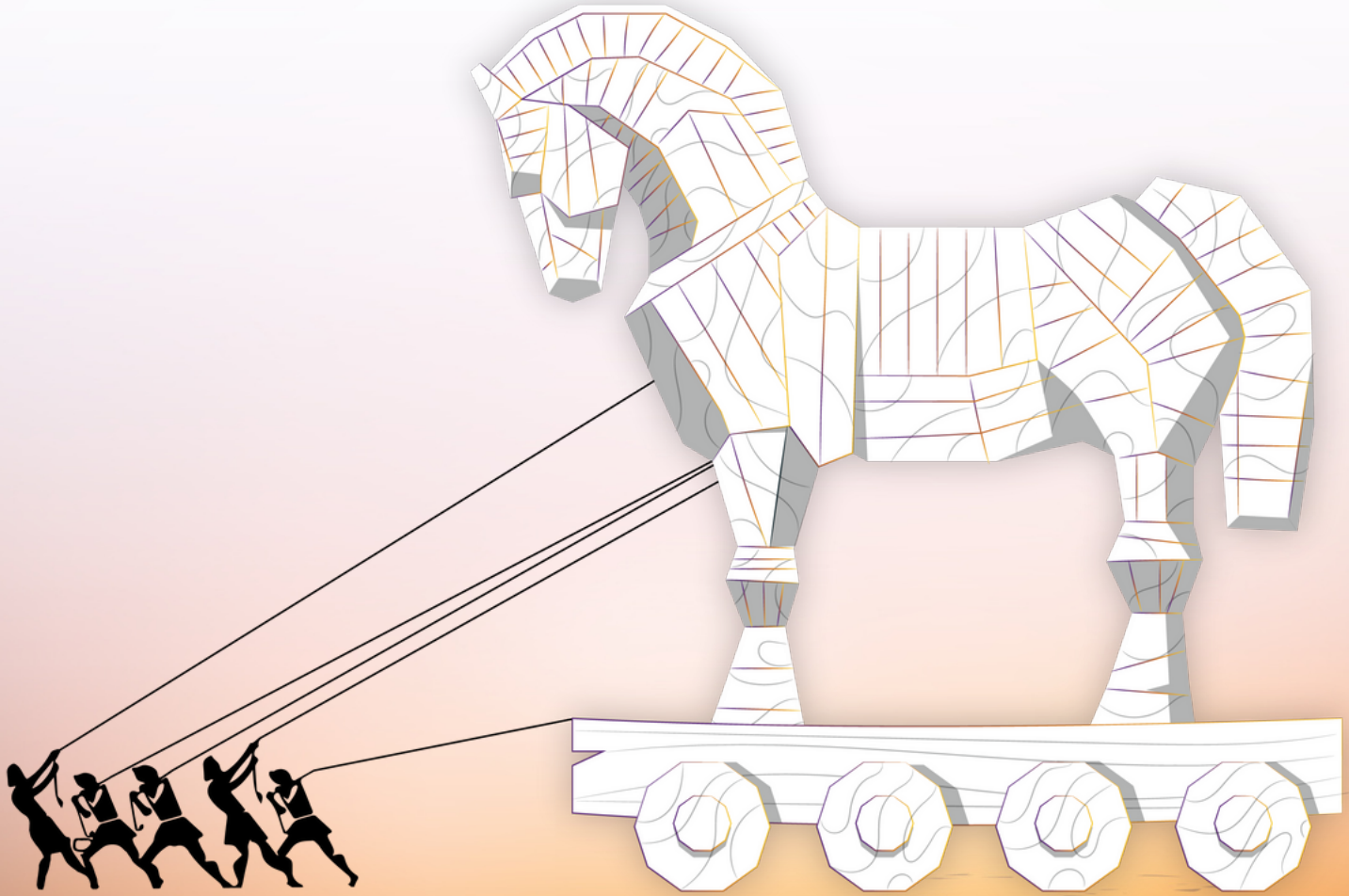
- Discovering your software components and creating their entire genealogy – including all transitive dependencies. This is called **deep provenance.**
- Establishing Integrity: Our unique deep DNA fingerprinting technology establishes integrity throughout the supply chain without relying on external tooling and their assertions.
- Evaluating Inherent Risk: Determining how risky a component is.
- Building a great Software Supply Chain "Lineaje" by remediating integrity and inherent risks.

Our Software Supply Chain Lineaje Model enables us to rate all software we assess on an inherent risk vs. software integrity graph. This makes it easy to compare versions of the same software and track improvements over time. Additionally, you can evaluate your product portfolio and remediate your weaknesses. Procurement managers & CISOs can compare various vendors' products to determine the best vendor for them.



An in-depth discussion of this model can be found in the chapter "A Formal Model for Software Supply Chain Security". For a quicker solution, you can find a cheat sheet on the last page of this report.

As previously mentioned, we performed our genealogy analysis on one of the world's most widely used open source software producers – The Apache Software Foundation – to demonstrate the effectiveness of this approach. However, most dependencies in ASF projects are from non-ASF sources enabling a rich analysis of a wide spectrum of open source packages and driving some conclusions about the open source ecosystem.



Trojan Horses

What's Inside an Open Source Trojan Horse?

Lineaje Data Labs decomposed and assessed tens of thousands of open source projects – enabling us to not only analyze the top ASF projects but also get a statistically significant insight into open source projects. This chapter focuses on key findings and takeaways for open source software.

10%
Visible Direct Dependencies

90%
Transitive, Invisible, Dependencies

2.7
Mean times a package is reused in a Project

3%
Unknown Components

5.3%
Components of dubious origin

82%
Components with inherent Risk

90%
Transitive vulnerabilities your dev cannot patch

2.7
Average number of repeated patches for every vulnerability

41%
Vulnerabilities introduced by dependent Projects

>50%
Critical & high vulnerabilities in Transitive Dependencies

What Troy Missed, But You Should Not!

The key takeaways from our analysis are interesting

Every Software is Like an Iceberg

Icebergs are deceptive, appearing smaller and more benign than they truly are. The same holds for open source software. More than four-fifths of open source software is invisible to - tools focusing on direct dependencies and developers including those packages.

The Soldiers of Troy: Developer-led Patching is of Limited Defensive Value

Open-source software, like commercial software, contains direct and transitive dependencies. Vulnerabilities exist in both. However, not all vulnerabilities have solutions. In fact, most vulnerabilities have no available fixes. **Your most significant risk is not the vulnerabilities you did not patch but those you do not have patches for!** They stay with you whether you patch the rest or not.

Integrity Check: True Open Source Dependency or Trojan Horse?

While discovering a software's dependency chain is interesting, establishing its integrity is critical. Without specifying the integrity of both embedded packages and their linked source code, you cannot know which dependencies could be a Trojan horse that, in turn, drag in other unsavory dependencies. Our research shows that 82% of all open-source software components are inherently risky.

Investigate the Horse: Manage the Software Supply Chain Lineaje

In our model, the popularity of a project has little to do with its overall risk position, but developers predominantly select based on popularity. Apache eCharts, the most popular ASF project, is firmly bright red – but is embedded in approximately 210,751 other open-source projects. This may include some you already use!

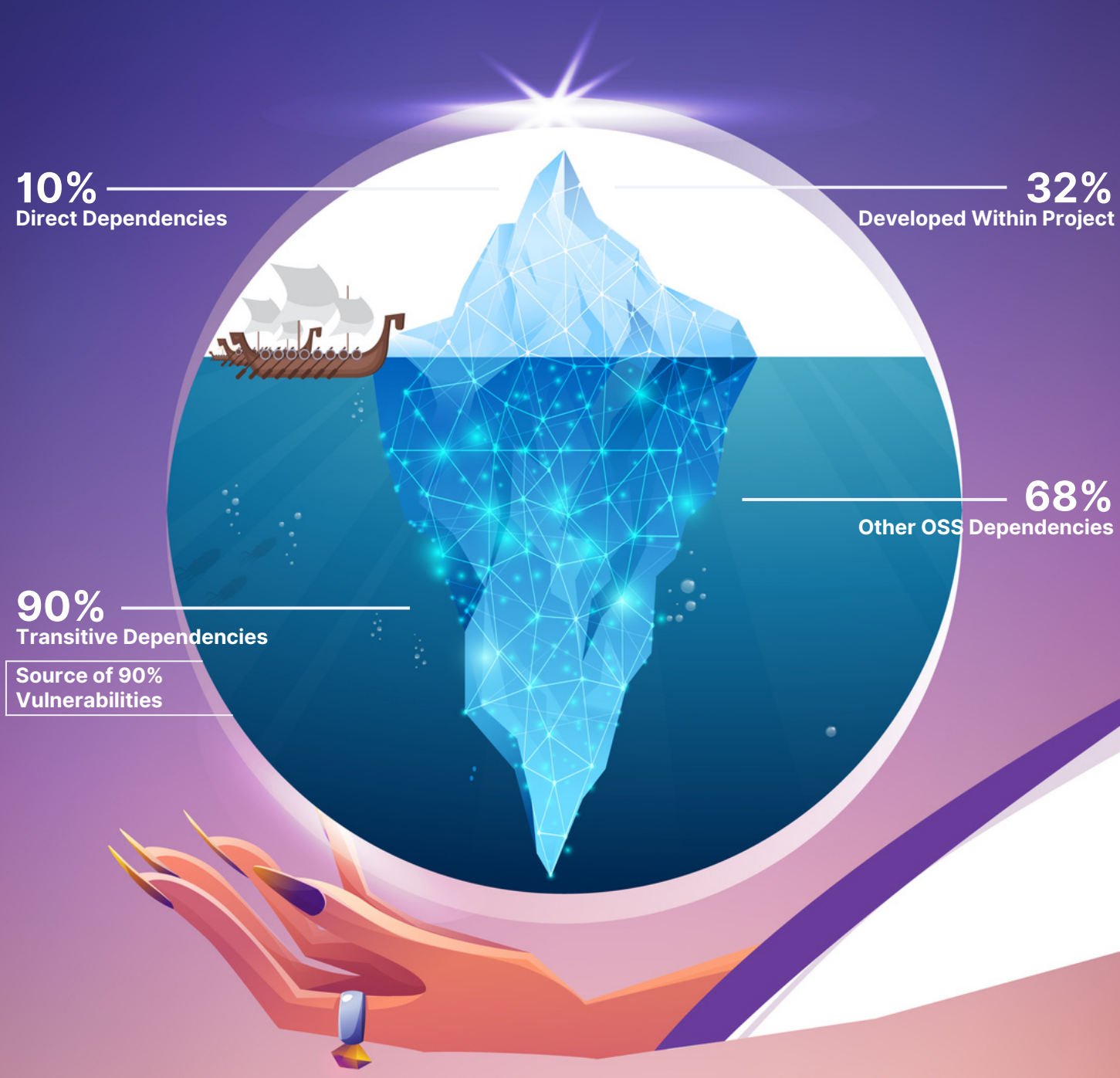
Every Software is Like an Iceberg

Discover Software Genealogy

Icebergs are deceiving. These seem smaller and more benign than they are. The same is true for open source software. More than 80% of open source software is invisible to - tools focusing on direct dependencies and developers including those packages.

Open source software also embeds other open source software. Our analysis reveals that open source software developers freely pull in other open source packages which are not part of their project.

Each dependency they pull in brings in more dependencies to their project. Sometimes they pull in the same package! On average, a package is used 2.7 times in any piece of software your organization pulls in.



Integrity Check: True Open Source Dependency or Trojan Horse?

Measure Software Integrity

Discovering your dependency chain is interesting, but establishing its integrity is critical. Without establishing the integrity of both embedded packages and their linked source code, you cannot know which dependencies could be a Trojan horse that, in turn, drag in other unsavory dependencies. Our sample shows that 82% of all open source software components are inherently risky. (For a deeper discussion, please read the chapter "Establishing a formal model of Software Supply Chain Security")

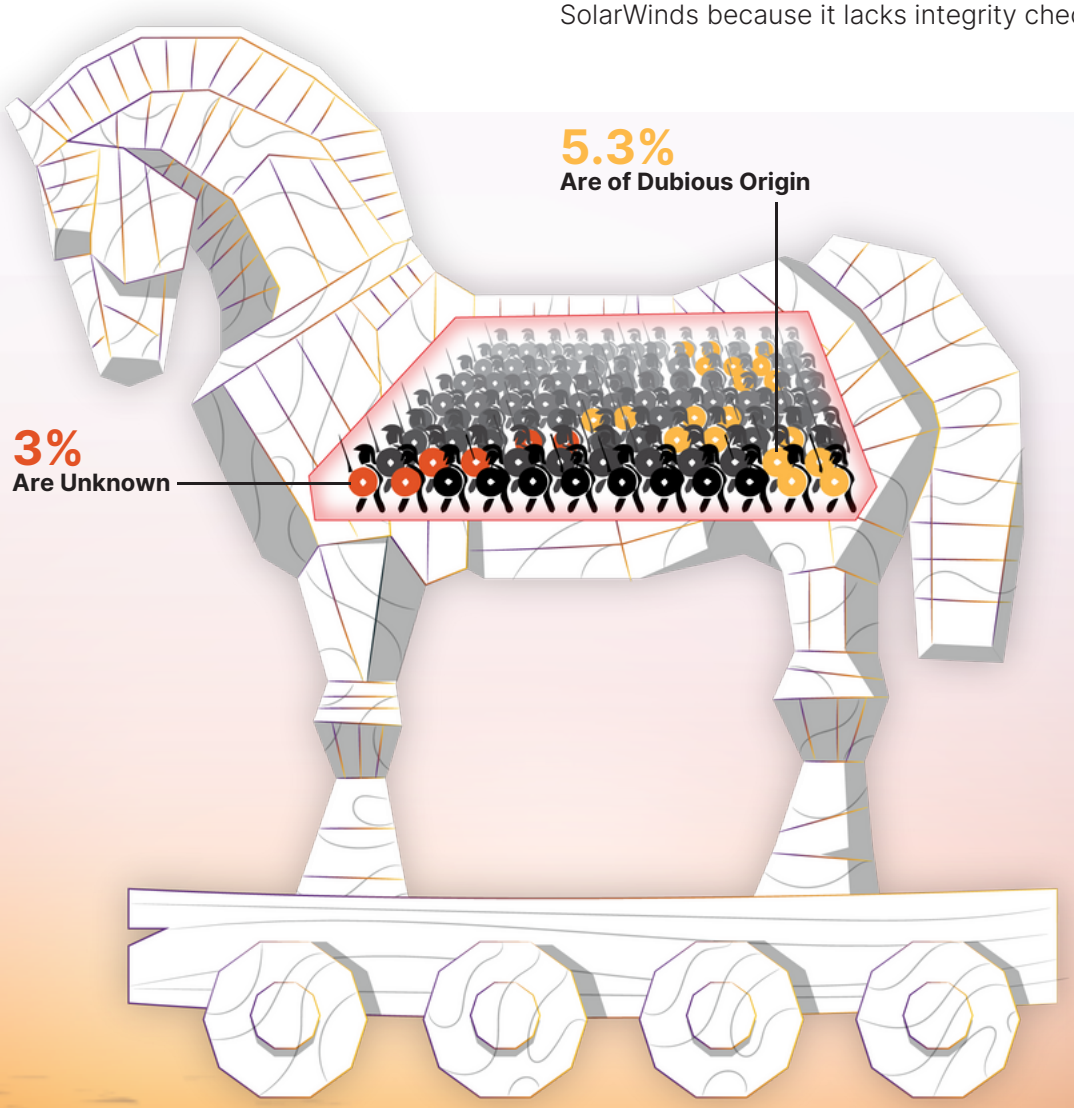
3% of all components analyzed are unknown! These unknowns are potentially more lethal than any vulnerability in your software. Deep integrity analysis shows we cannot determine any known source for them – LCAL 0 is their integrity level.

However, they are included in the package you depend on and ship as part of your software.

Your SCA tools and dependency checkers will find nothing wrong with these unknowns because they don't establish the integrity of their reports! You are distributing software with low integrity that is easily tamperable and unable to detect tampering.

Your SCA tools and dependency checkers will find nothing wrong with these unknowns because they don't establish the integrity of their reports!

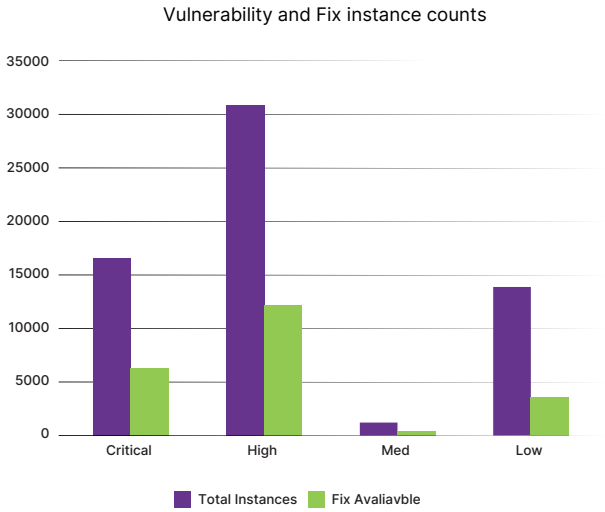
Another 5.3% are of dubious character – LCAL 1 level. They lie about who they are! Integrity checks reveal that the PURL package and/or its linked source do not match the package you are shipping with. If a dependency pulls them, they come pre-tampered. Your dependency checkers will identify them as "good to go"! Just like your SCA tool would never detect a SolarWinds because it lacks integrity checks.



The Soldiers of Troy: Developer-led Patching is of Limited Defensive Value

Understand Inherent Risk

Like commercial software, open source software contains direct and transitive dependencies. Vulnerabilities exist in both. However, not all vulnerabilities have fixes. In fact, the majority of vulnerabilities have no available fix. Your most significant risk is not the vulnerabilities you did not patch but the ones you do not have patches for! They stay with you whether you patch the rest or not.

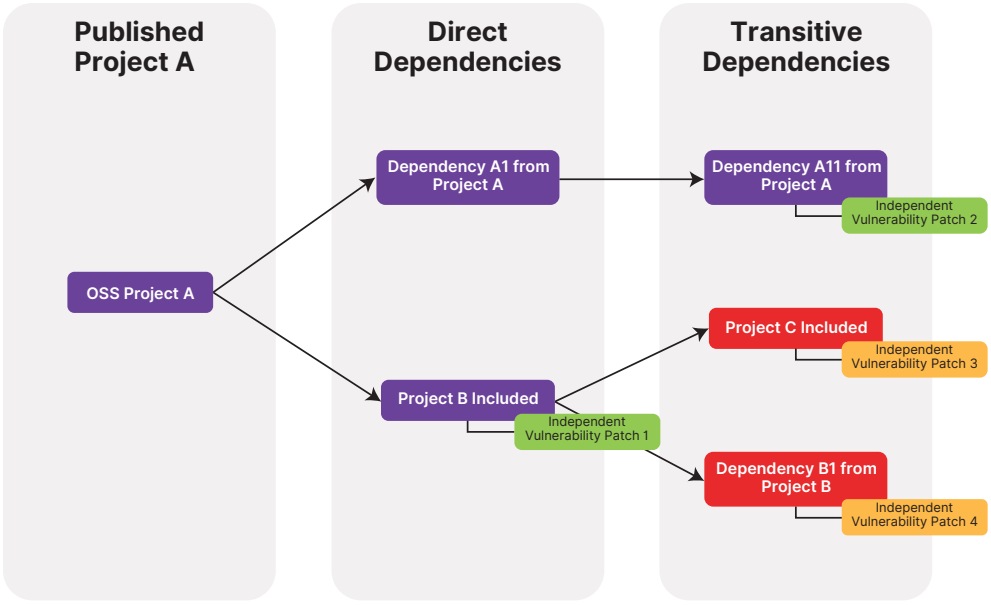


Your most significant risk is not the vulnerabilities you did not patch but the ones you do not have patches for!

We see the following distribution for all the vulnerabilities you have fixes for:

Component Type	Vulnerability Distribution
Vulnerabilities in direct dependencies	6%
Vulnerabilities in other transitively dependent open source projects	94%

Vulnerabilities that are part of transitive dependencies and are external to your project cannot be patched by your developers. Though independent patches exist for those vulnerabilities, Project B has to uptake those patches. If your developers do that, they also have to re-certify Project B with those updates - this is impossible for them!



This means that its code is exercised a few times in different contexts. Reachability analysis prioritizing vulnerabilities based on what an organization exercises misses this critical insight.

Additionally, our analysis reveals that a specific component, on average, exists 2.7 times. This means that its code is exercised a few times in different contexts. Reachability analysis prioritizing vulnerabilities based on what an organization exercises misses this critical insight. Vendors pitching “reachability analysis” need to do a reachability analysis for that component in each dependency instance before advising you to ignore that patch.

Investigate the Horse: Manage Your Software Supply Chain

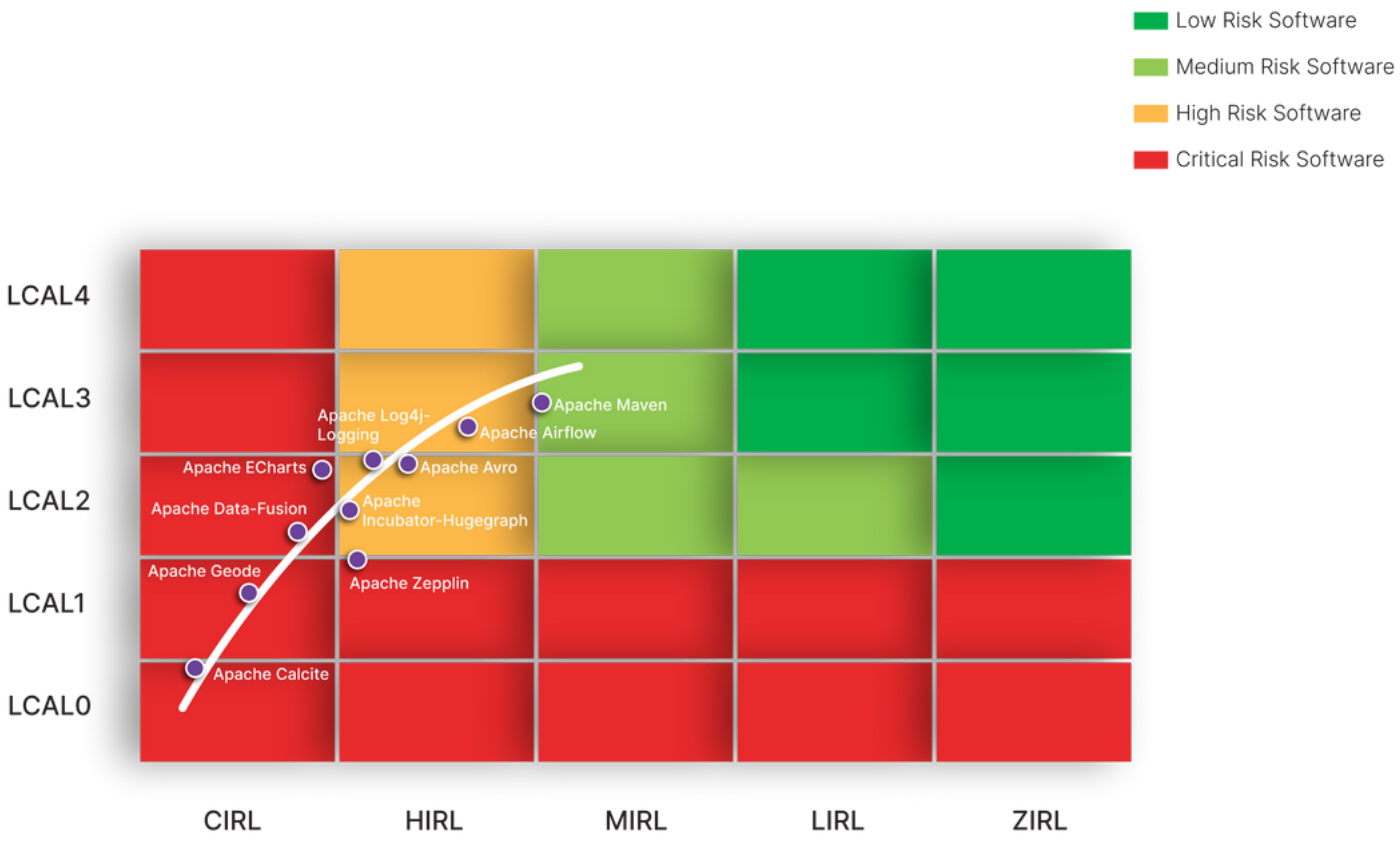
Are you shipping a Trojan Horse?

What's easy to understand is – the goal - a high-integrity software supply chain with low inherent risk. However, the soldiers of Troy could establish neither - they didn't have the tools. Have your developers established that, or are you shipping a Trojan horse?

All the ASF projects do not get a passing grade in our analysis. Though some come close, they could get there if they fixed key vulnerabilities and up took some patches. This underlines the point - if you are not deeply analyzing your dependencies from open source software, your products are very unlikely to pass an in-depth security assessment of your SBOM.

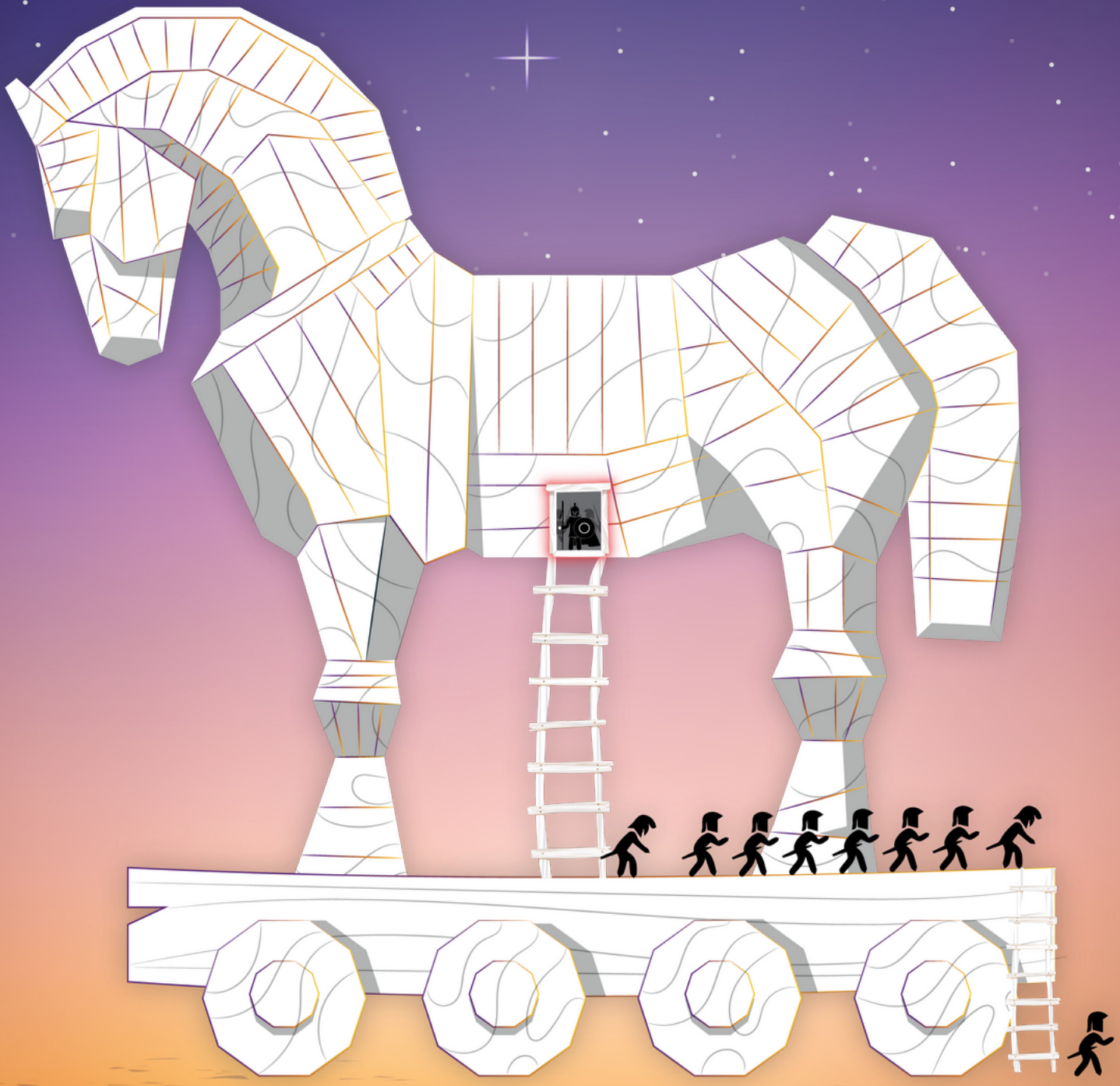
Open source dependencies drag in significant security risks. Each is a potential Trojan horse that can bring in low-integrity software, putting your organizations and your customers at risk. Many have significant vulnerabilities that your developers cannot patch.

In our model, the popularity of a project has little to do with its overall risk position, but developers predominantly select based on popularity. Apache eCharts, the most popular ASF project, is firmly bright red – but is embedded in 210,751 other open source projects - some that you may be already using.



A dependency with high integrity (LCAL 2+) implies the project is well maintained. Those developers are more likely to update their software regularly and respond to risky issues faster. In our Apache Foundation Software analysis, we see this playout. Apache Airflow and Apache Maven are high integrity and come close to our green zones. If the authors of those projects upgraded components to the latest versions, they would firmly be in the green.

We recommend you take the first step - connect with us, and we will assess your software supply chain. And if you consume software, insist on an SBOM so that you know how well-managed a product's software supply chain is. SBOM360 can also tell you that!



Assessing the Apache Software Foundation: Software for the Good of Troy

Key Statistics: Is “Software for the Common Good” good for Troy?

1190
Organizations contributing in top 44 Projects

67
Countries with ASF Contributors

68%
Non Apache Components in Apache Software

8.32%
Unknown, Unverifiable Components

>40%
Critically Risky Components

40%
Fortune 500 Companies using Apache Kafka

32%
Websites use Apache HTTP Server

16,489
Critical Vulnerability instances with no fixes available in 44 Projects

47,343
Critical & High Vulnerability instances in 44 Projects

90%
Vulnerability with fixes that cannot be patched by users of ASF Projects

$22B
Estimated Annual Value of Software Used

210,751
OSS Projects Using Apache eChar ts

1191
Unique Vulnerabilities in 14912 Unique Components

113
Components in eCharts with unknown provenance

36
Components in eCharts which originated from Meta

Top Takeaways for the ‘Software for the Common Good’

Who’s Writing Apache Software? - Discovering ASF Genealogy

Around a third (13,322) of the components in ASF software analyzed were written by ASF contributors; however, two-thirds (28,667) were dependencies included from other open source projects. This has significant implications for the maintainability and vulnerability management of ASF projects.

Older is Wiser: Later ASF Versions are Less Risky

Lineaje’s analysis of top ASF projects is not limited to the latest version. We went back in time and inspected earlier releases as well. Crawling previous releases helped develop a history of Inherent Risk Level (IRL) for each component in that release. Later versions have lower vulnerabilities, better integrity, and lower risk than earlier versions. Organizations should promptly move to the latest versions of ASF projects.

ASF Projects: Beautiful Horses, But What Do They Hide?

Analysis of the top 44 projects shows that projects whose dependencies are authentic and whose packages could be mapped to their source have high integrity and are less risky. Their authors also better address their vulnerability, code quality, and security posture.

Should You Bring the Gift In? Assessing Apache Projects

Enterprises should carefully evaluate every ASF dependency or project they consider using a tool like SBOM360. Some, like Apache Maven, would match COTS products, while others, like Apache Calcite, do not meet basic software integrity expectations.

The Apache Software Foundation

Apache Software Foundation is a non-profit organization that manages and supports over 320 open source active software projects, including Apache HTTP Server, Apache Hadoop, Apache Tomcat, and more. Millions of people use these projects across the globe, with the Apache HTTP Server alone powering over 40% of all websites. Additionally, the foundation has over 750 individual members and 8,400 committers who contribute to developing and maintaining these projects. In 2021, ASF estimated that its contributors updated or added more than half a billion lines of code. The foundation estimated that the software delivered by it to companies would be worth roughly $22 Billion.

The foundation estimated that the software delivered by it to companies would be worth roughly $22 Billion.

ASF's contribution to the world has been immense, with its projects helping shape the internet and how the industry uses it today. For instance, in addition to the popular free and open source solution for hosting websites, Apache HTTP Server and Apache Hadoop provide a distributed computing framework for processing large datasets, enabling businesses and organizations to derive insights from big data. These projects have become critical technology infrastructure components, with millions relying on them daily.

ASF's popularity is rising, with its projects' adoption levels spiraling up in various industries. For example, Apache Hadoop's market share grew from 26% in 2017 to 44% in 2021. Additionally, in 2020, Apache Kafka, an open source distributed streaming platform, was used by over 40% of Fortune 500 companies. Overall, Apache Foundation's contributions to open source software development have been invaluable. To our understanding, ASF represents the gold standard in open source software — a well-funded, successful, open organization.

Additionally, in 2020, Apache Kafka, an open source distributed streaming platform, was used by over 40% of Fortune 500 companies.

Despite the many positive contributions of ASF's projects, vulnerabilities have been discovered in their software that has affected many organizations. For example, the Log4j vulnerability discovered in December 2021 exposed users to attacks that could execute arbitrary code on affected systems. This vulnerability and other critical vulnerabilities – like the Apache Struts vulnerability that was exploited in the Equifax data breach – illustrate how even the best-run software projects have implications for companies that use them.

The Log4j vulnerability discovered in December 2021 exposed users to attacks that could execute arbitrary code on affected systems.

This prompted Lineaje to ask the questions - "What's in ASF software?" and "How good is it?"

The Apache Software Foundation Assessment

Lineaje Data Labs completed a comprehensive analysis of 44 projects from ASF. To begin, there are some high-level key findings.

These 44 projects include 41,989 components. Across each project, there are a total of 37,860 unique transitive dependencies and 14,912 unique components. The total lines of code analyzed are well over 26 million. Below is a summary of the key findings from our analysis of ASF packages:

Analytic Findings of ASF:

- 41,989 total components and 26 million lines of code
- 68% non-ASF components
- 10% of direct components, 90% of indirect (transitive) components
- 50% of critical /high vulnerabilities found in indirect (transitive) components.
- 8.3% of components have unknown/dubious integrity of supplier origins
- >40% Inherent Risk Level (IRL) assessment at “Critical Risk” >>Highest

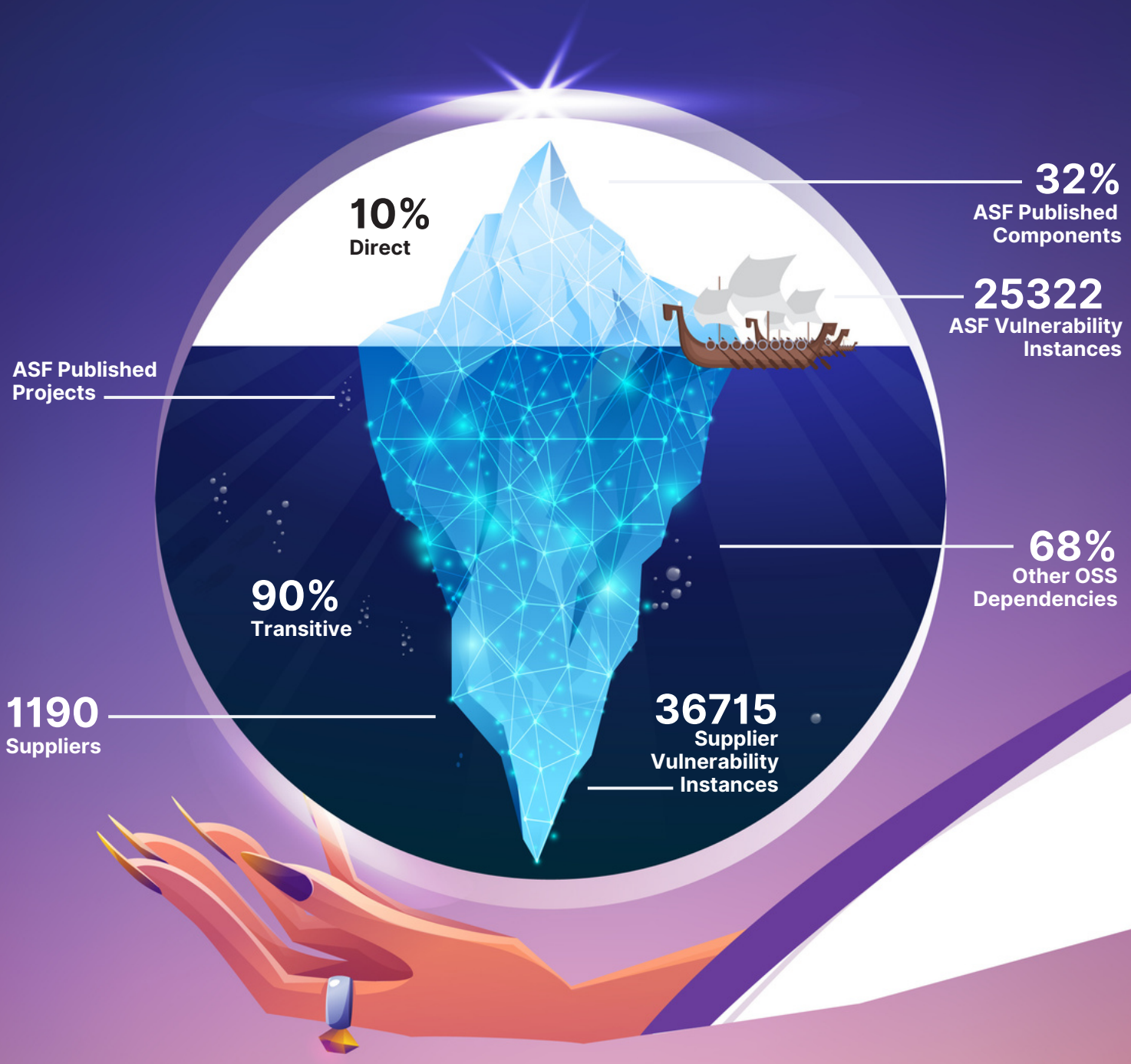
Significance of Finding:

- ASF cannot patch most of the vulnerabilities.
- Approaches focused on patching and upgrading by ASF consumers will not be effective.
- Fixes for vulnerabilities must be applied at the root component and then propagate to the “Top Level Package.”
- Validity of the origins of components is correlated with more security software.

1. Who's Writing Apache Software?

Discovering ASF Genealogy

Around a third (13,322) of the components in ASF software analyzed were written by ASF contributors; however, two-thirds (28,667) were dependencies included from other open source projects. This has significant implications for the maintainability and vulnerability management of ASF projects.



Most (68%) of the components are from non-ASF suppliers, including Oracle, Google, and Meta. This has significant ramifications for effectively identifying and fixing vulnerabilities and issues within ASF packages.

In the 44 ASF packages we analyzed, 1,190 different open source suppliers were discovered, many contributing less than 1% each. Thus, the largest number of components are from smaller, lesser-known, and harder-to-trace suppliers. This raises concerns about ASF's ability to manage the quality and security of the components in their packages from other suppliers.

2. ASF Projects: Beautiful Horses, But What Do They Hide?

Analysis of the top 44 projects shows that projects whose dependencies are authentic and whose packages could be mapped to their source are less risky. Their contributors also better address their vulnerability, code quality, and security posture.

Measuring ASF Software Integrity: High Integrity Components

Lineaje Component Attestation Level	Number of Components	% of Total
LCAL0	1255	2.99%
LCAL1	2239	5.33%
LCAL2	9649	22.98%
LCAL3	28846	68.70%

Using the Lineaje Component Attestation Level (LCAL) analysis of ASF, we determined that the "attestability" is relatively high for over 90% of dependencies.

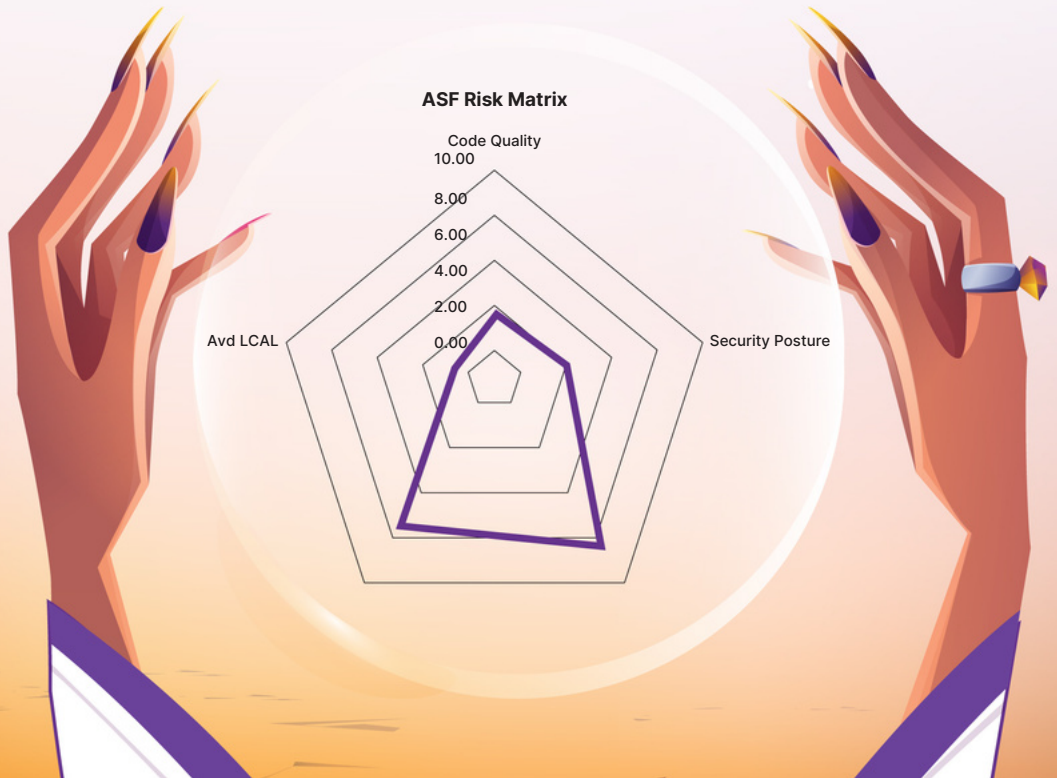
However, all components with attestation levels of LCAL 0 and LCAL 1 are completely tamperable by a threat actor. These create high tamperability potential. Given that most are transitive dependencies, these tampers will be undetectable by current CI/CD tools and build-system-centric attestation methodologies like SLSA. If your build tool did not build the dependency – which is essentially impossible for transitive dependencies - they will not raise a flag. Any supply chain is only as good as its weakest link, which is also true for the software supply chain. These low LCAL components are at high risk for tampering.

Even High Integrity Components Can Pose Risks

Even though we saw high integrity levels in over 90% of dependencies, more than two-thirds of components pose critical or high risks. The inherent risk encapsulates risk from vulnerabilities, code quality, security posture, attestability, and exposed secrets in code. An interesting aspect that SBOM360 measures is the 'Age' of the component. If a component has not been updated for an extended period of time, it is not maintained.

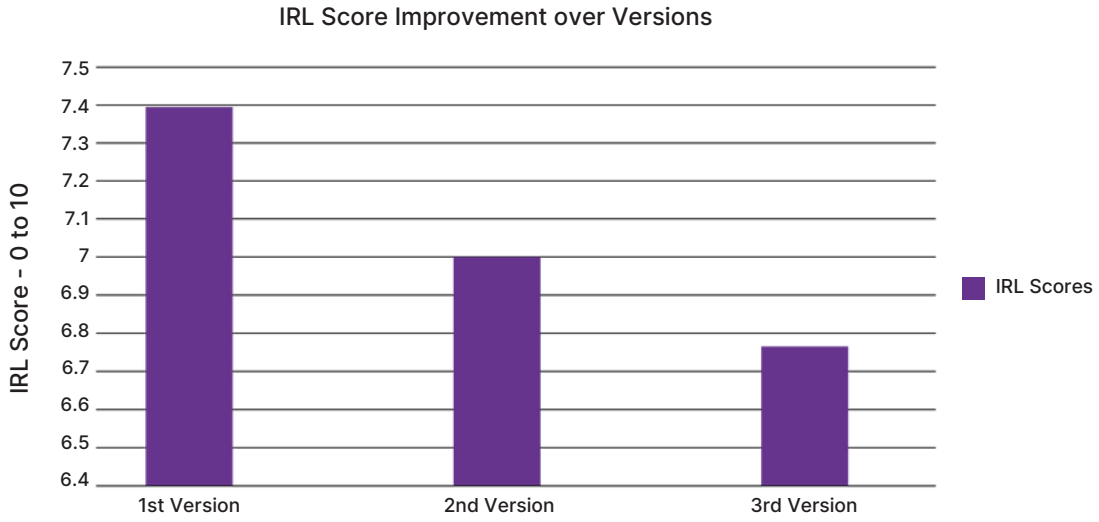
Inherent Risk Level	Count	%
Critical	22455	55%
High	5810	14%
Med	5107	13%
Low	7204	18%

Most components have risk levels that should be unacceptable to security-conscious organizations. Much effort would be needed to make software components with 'Critical' or 'High' inherent risk levels "safe". Therefore, part of the strategy for securing software includes identifying the components that explicitly drive up the risk and either removing them or finding alternatives.

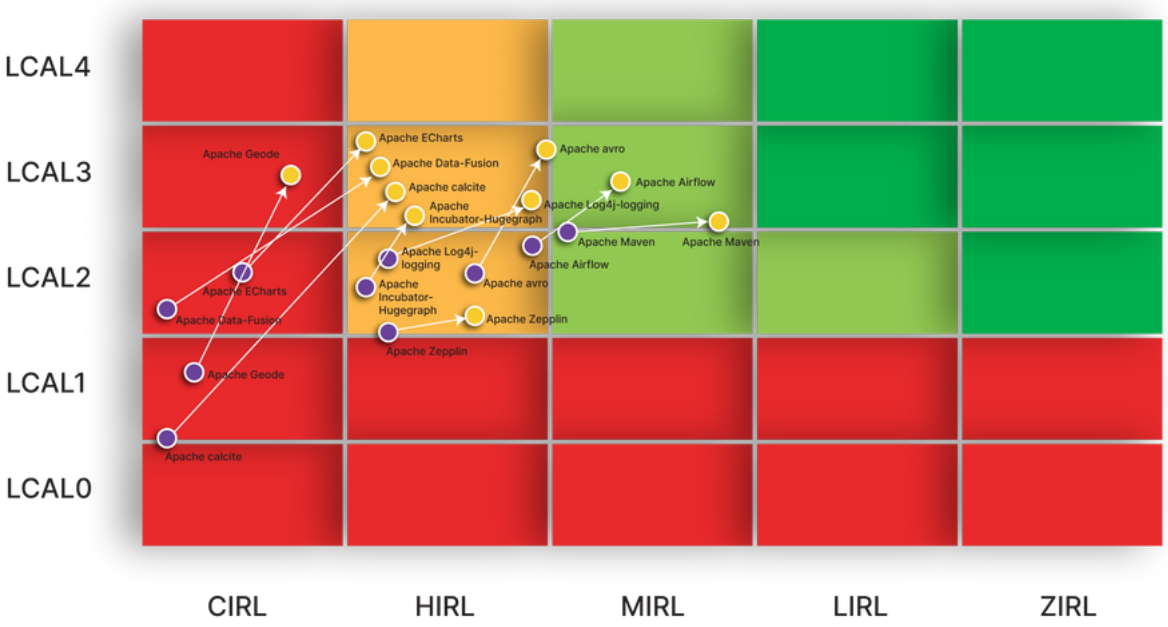


3. Older is Wiser: Later ASF Versions are Less Risky

Lineaje's analysis of top ASF projects is not limited to the latest version. We went back in time and inspected earlier releases as well. Crawling previous releases helped develop a history of Inherent Risk Level (IRL) for each component in that release. Later versions have lower vulnerabilities, better integrity, and lower risk than earlier versions. Organizations should promptly move to the latest versions of ASF projects.



ASF projects should also follow this advice. Upgrading ASF written components in ASF projects and moving to later versions of Fasterxml components will accrue 81% of all improvements that a comprehensive upgrade of all components would have accrued. This dramatically improves integrity and risk ratings for ASF Projects, moving many into acceptable risk levels.



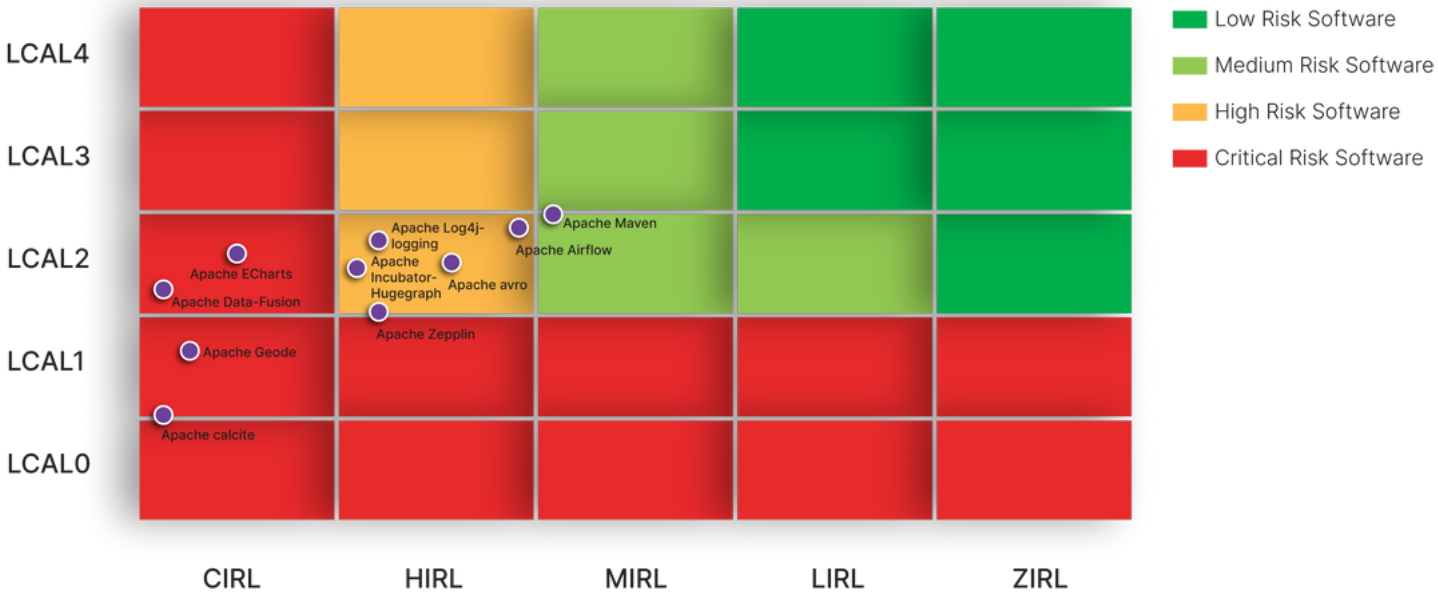
4. Should You Bring the Gift In? Assessing Apache Projects

Enterprises should carefully evaluate every ASF dependency or project they consider using a tool like SBOM360. Some, like Apache Maven, would match COTS products, while others, like Apache Calcite, do not meet basic software integrity expectations.

Managing a Great Software Supply Chain Lineaje Model

Given that ASF projects have significant dependencies recommended that ASF evaluates its dependencies as on non-ASF projects, it is apparent that the risk of carefully to improve its ratings across both integrity those projects roll into ASF projects. Just like we and inherent risk dimensions. In fact, ASF, like many recommend that organizations consuming ASF software companies, needs an open source office to software evaluate each dependency carefully, it is ensure that its projects' components meet their customers' expectations!

ASF, like many software companies, needs an open source office to ensure that its projects' components meet their customers' expectations!

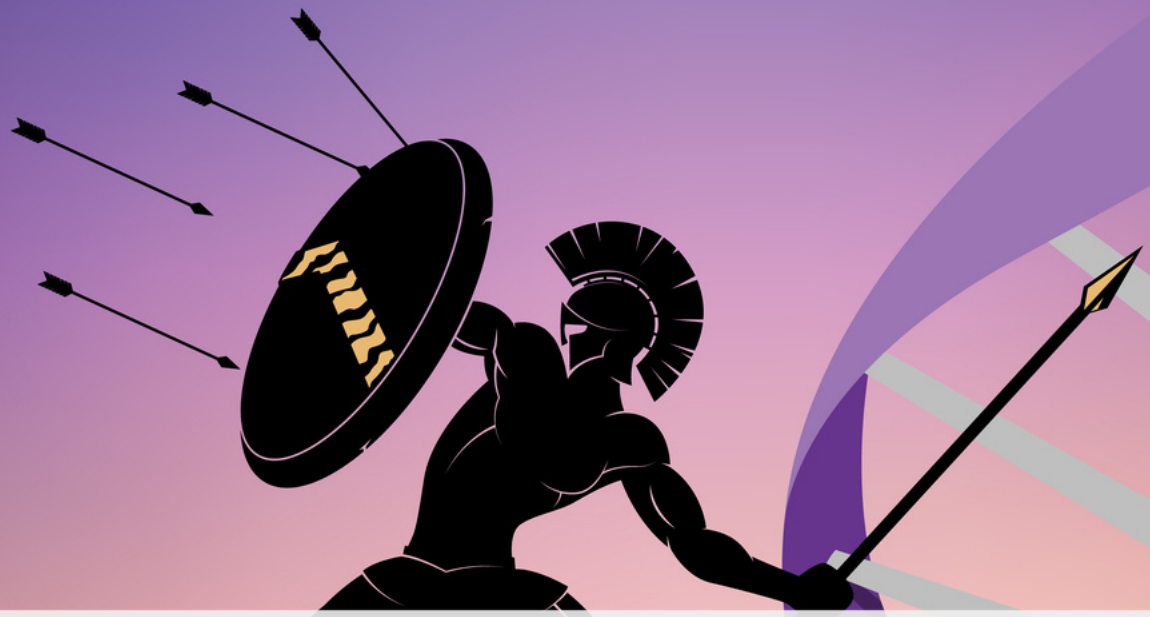


A Formal Model for Software Supply Chain Security

Lineaje's genetic software analysis is a detailed model to measure the four critical aspects of understanding and fixing software at the source, as illustrated in the diagram below.

Lineaje's Software Supply Chain Security Approach - Manage Software at the DNA Level

1. **Understand the DNA and Lineaje of your Software**
   Complete software inventory achieved when you correctly DECOMPOSE your software
2. **Determine the Intergrity of your Software**
   Can you ATTEST that your software DNA is actually what you believe it to be?
3. **Measure Inherent Risk in your software**
   ASSESS the risk of each DNA component, dependencies, and sub-dependencies down to the "leaf"
4. **Continuously Improve & Fix Your Software**
   Continuously improve software integrity and reduce inherent risk with targeted DNA level fixes

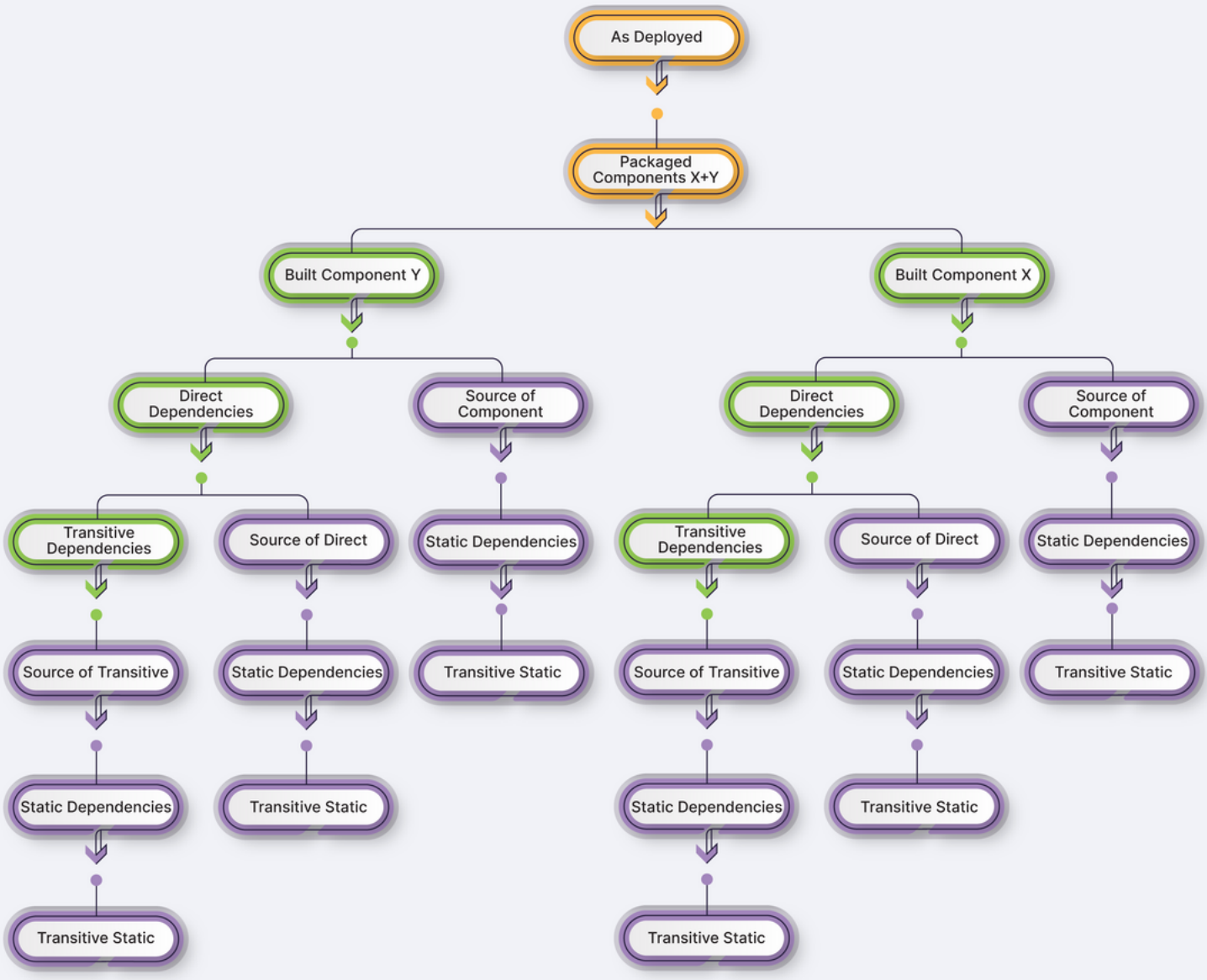


These key aspects are:

- **Discover your software's genealogy tree:** Complete software inventory achieved when you correctly DECOMPOSE your software.
- **Measure your software's integrity:** Can you ATTEST that your software DNA is what you believe it to be?
- **Understand inherent risk embedded in your software:** Software developers & consumers must ASSESS the risk of each component, dependency, and sub-dependencies - down to the last leaf.
- **Manage software supply chain Lineaje:** New version planning can now include ways to improve software integrity, reduce inherent risk, and quantify how new versions minimize the risk while continuing to highlight innovations.

Discover Your Software's Genealogy – Decomposing Software

Distributable, package, and source decompositions are required to understand your software's genealogy tree. At Lineaje, software decomposition aims to understand the entire software genealogy, regardless of the starting point or depth. To understand the most comprehensive view of the software, Lineaje crawler looks at the distributable, identifying all packaged components. It further decomposes the package, identifying components and their direct and transitive dependencies. Each component is mapped to its source origins, and a static direct and transitive dependency is prepared. This is illustrated in the diagram below.



It is important to note that regardless of the starting points – Distributable Image, Built-Component, or Source Repo – a comprehensive origins mapping is required to build the entire dependency tree.

For ASF projects, our analysis started from the source repo of the project. Lineaje Crawler built the direct and transitive dependency tree of the project. Then for each component in the dependency tree, source-build mapping yields component source repos helping us pull the static and transitive static dependency list for those components. Obtaining the entire genealogy of the open source repo allows us to derive the project's most comprehensive 'Attestation' and 'Inherent Risk Level'.

Measuring Software's Integrity: Attesting Software

There are multiple proposed standards for determining your software's integrity. NIST800-218 and SLSA (Supply Chain Levels for Software Artifacts) take a developer-centric view of component-level attestation and places the onus of creating a secure supply chain on people, tools, and processes. Additionally, it focuses on compliance but does not deliver supply chain security. Build tool dependencies in software attestation are problematic, and we shied away from that. Why? In supply chain attacks like SolarWinds, the build system itself was compromised - there are inherent weaknesses in build-tool-based attestation approaches.

In supply chain attacks like SolarWinds, the build system itself was compromised - there are inherent weaknesses in build-tool-based attestation approaches.

The second challenge is that with millions of open source projects, third parties and software development companies must comply with NIST and SLSA guidelines to create a trusted software supply chain. That is not going to happen anytime soon!

The third challenge is that compliance is not security. Almost all breached companies are compliant companies. People following processes and relying on compromisable tools are typically the weakest links in security.

Compliance is not security. Almost all breached companies are compliant companies. People following processes and relying on compromisable tools are typically the weakest links in security.

In today's world, components are created all over in multiple build systems. So, SLSA and NIST approaches are a non-starter unless adopted globally. For Google or NIST, that's probably more attainable. However, we are Lineaje Inc. and have a simple, practical, and attestation approach that has set a standard for future software guidelines.

We are Lineaje Inc. and have a simple, practical, and attestation approach that has set a standard for future software guidelines.

Protecting Troy: Introducing Lineaje Component Attestation Levels (LCAL)

Lineaje bases the supply chain integrity attestation on our proprietary and amazing fingerprinting technology. This enables us to create a software supply chain of trust, with each link having its own integrity rating from a shipping product down to its last leaf component with no further dependencies. Our fingerprinting-centric LCAL methodology has a significant advantage over build-system-centric attestation approaches. Lineaje uses what is accessible – package and source code. For example, if you are using open source software, those are accessible to you. So why not use them for attestation? If you modify those dependencies, you can still access both, enabling deep fingerprinting-based attestation.

Lineaje enables component attestations without dependency on any tools. This attestation is at a component level. A key thing to understand is that each attestation level is for the component itself, not its dependencies. A highest-level component attestation does not mean that its dependencies have as high an attestation as the parent. This is also true for SLSA and NIST and is an inherent weakness of component-level attestation. This understanding is critical for supply chain security.

A highest-level component attestation does not mean that its dependencies have as high an attestation as the parent. This is also true for SLSA and NIST and is an inherent weakness of component-level attestation.

Last but not least, LCAL levels identify the weakest links in the software supply chain. With chained LCAL levels, Lineaje can attest that what you shipped is what you built, what you built is what you sourced, and what you sourced is what was published by your dependency, what was published is what was sourced by them, and so on all the way upstream to the leaf transitive dependency – a provable end-to-end attested software supply chain.

With chained LCAL levels, Lineaje can attest that what you shipped is what you built, what you built is what you sourced, and what you sourced is what was published by your dependency, what was published is what was sourced by them, and so on all the way upstream to the leaf transitive dependency – a provable end-to-end attested software supply chain.

Lineaje Component Attestation Level (LCAL)	Attestation Description starting with component package	Mapping to SLSA Levels	Recommendation
LCAL 0	**Unknown component:** The component package cannot be resolved to any known package.	SLSA 0: No guarantees	Block ship without manual attestation by SecOps
LCAL 1	**Known component:** The component is deemed as Known if the component name, PURL are traceable and attestable. The package may be available at the PURL location, but the fingerprints do not match.	SLSA 1: Unsigned provenance	Block ship without manual attestation by SecOps
LCAL 2	**Attested component:** The component is deemed as Attested if the component name, PURL are traceable and attestable. The package is available at PURL location and the fingerprints match.	SLSA 2: Hosted source/build, signed provenance SLSA 3: Security controls on host, non-falsifiable provenance SLSA 4: Requires two-person review of all changes and a hermetic, reproducible build process	Minimum level for every component in your supply chain
LCAL 3	**Attested Build & Source:** A component package is Attested, and its source exists, and the package is Attested to be built from that source.	Undeterminable by SLSA	Lineaje recommended minimum attestation level for Software Supply Chain Security
LCAL 4	**Fully Attested:** Package & source attested to be untampered and malware free. Attestation certifies that there is no malicious code or tamper that was introduced in the original source or between the source and built output. LCAL4 attestation is compatible with builds produced in the past.	Undeterminable by SLSA	Lineaje recommended attestation level for your Software Supply Chain Security to remain untampered & malware free

Understanding Inherent Risk for Each Component of Software

Consumers and producers of software alike want to have a certain level of confidence that their software is secure and reliable. Software producers (where typical software comprises 70-90% open source software) cannot ensure the level of security unless they know the risk in each dependent component. Additionally, emerging threats within the software supply chain landscape expose software to existing and new risks. Log4j underscores this challenge.

Recently, processes and frameworks have been proposed to help assess software security level; however, these methods to date are high-level, not adopted uniformly or ubiquitously, leading to ad hoc implementations.

The challenge is that no consistent, clear, testable, and repeatable ways exist to build and maintain fact-based trust between the components. This is because each latest open source version adds newer components, a new set of inherent risks, and a limited ability to parse through them.

The challenge is that no consistent, clear, testable, and repeatable ways exist to build and maintain fact-based trust between the components. This is because each latest open source version adds newer components, a new set of inherent risks, and a limited ability to parse through them.

Therefore, we developed an evidence-based quantitative methodology to assess software's inherent risk across key measures.

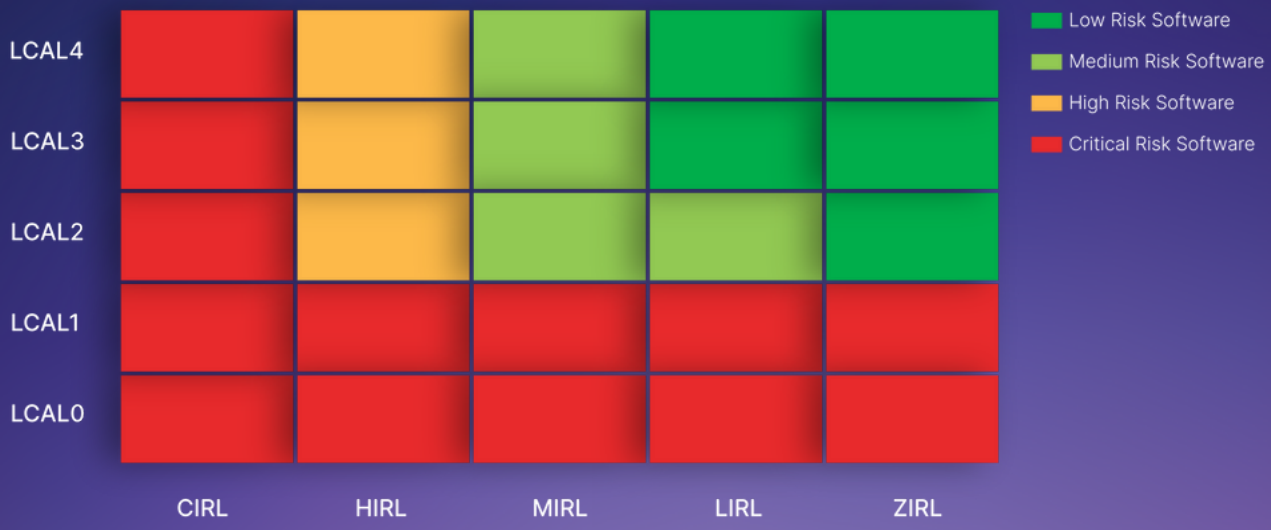
Lineaje Inherent Risk Levels (IRL)

Lineaje analyzes each component down to the leaf level (each component's dependencies, the dependencies' dependencies, down to the lowest level) against the key inherent risk measures. It then averages their scores to produce the inherent risk level of your software. The table below describes this process and provides recommendations for the different Inherent Risk Levels (IRLs).

Inherent Risk Level (IRL)	Description	Recommendation
Zero (0) (ZIRL)	Zero Inherent Risk Level. Average risk score for all components in the software across all six measures is zero.	Continue monitoring for changes in overall software to ensure risk remains None.
Low (0.1-3.9) (LIRL)	Low Inherent Risk Level Average risk score for all components in the software across all six measures is between 0.1 and 3.9.	Continue monitoring for changes in overall software to ensure risk remains low.
Medium (4.0-6.9) (MIRL)	Medium Inherent Risk Level Average risk score for all components in the software across all six measures is between 4.0 and 6.9.	Find opportunities to address some of the risk measures causing the average risk score to go to medium. Deploy risk mitigation solutions that target the identified high-risk components and vulnerabilities.
High (7.0-8.9) (HIRL)	High Inherent Risk Level Average risk score for all components in the software across all six measures is between 7.0 and 8.9.	Remove, update, or replace elevated risk components. Deploy secure coding practices including continuous security testing. Deploy risk mitigation solutions that target the identified high-risk components and vulnerabilities.
Critical (9.0-10.0) (CIRL)	Critical Inherent Risk Level Average risk score for all components in the software across all six measures is between 9.0 and 10.0.	Consider redeveloping the software starting with more secure components.

Software Supply Chain Lineaje Model: The Model from Olympus

Putting our Software Integrity and Software Risk Models together into a single picture (via the Software Supply Chain Lineaje Model) allows us to evaluate the risk of every component in each piece of software. Rolling it all up for an SBOM and averaging them for all components in the software enables the ability to plot software against each other.



This model is amazingly versatile. Here are some interesting and impactful use cases:

- Chief Product Officers can now view their entire portfolio on a single graph and understand which products are the riskiest for their customers. Given that they also know the root cause of those risks, with products like SBOM360, they can move their entire portfolio up and to the right.
- CISOs & Product Managers can now plot a product and its multiple versions and planned versions on one graph illustrating to management and customers how each release improves that product.
- Procurement Officers can take competing products from vendors and use the graph to compare them and pick the least risky one for their organizations.

  This allows software producers to view their portfolio in a single graph with subsequent drill-downs for a single SBOM with all its components ranked in an easy-to-use model. Similarly, software consumers can compare vendors' software in a chart choosing the one that best meets their expectations.

Quick Reference Cheat Sheet

Software Integrity Quick Guide

Lineaje Component Attestation Level (LCAL)	Attestation Description starting with component package
LCAL 0	**Unknown component:** The component package cannot be resolved to any known package.
LCAL 1	**Known component:** The component is deemed as Known if the component name, PURL are traceable and attestable. The package may be available at the PURL location, but the fingerprints do not match.
LCAL 2	**Attested component:** The component is deemed as Attested if the component name, PURL are traceable and attestable. The package is available at PURL location and the fingerprints match.
LCAL 3	**Attested Build & Source:** A component package is Attested, and its source exists, and the package is Attested to be built from that source.
LCAL 4	**Fully Attested:** Package & source attested to be untampered and malware free. Attestation certifies that there is no malicious code or tamper that was introduced in the original source or between the source and built output. LCAL4 attestation is compatible with builds produced in the past.

Software Risk Quick Guide

Inherent Risk Level (IRL)	Description
Zero (0) (ZIRL)	Zero Inherent Risk Level. Average risk score for all components in the software across all six measures is zero.
Low (0.1-3.9) (LIRL)	Low Inherent Risk Level Average risk score for all components in the software across all six measures is between 0.1 and 3.9
Medium (4.0-6.9) (MIRL)	Medium Inherent Risk Level Average risk score for all components in the software across all six measures is between 4.0 and 6.9
High (7.0-8.9) (HIRL)	High Inherent Risk Level Average risk score for all components in the software across all six measures is between 7.0 and 8.9
Critical (9.0-10.0) (CIRL)	Critical Inherent Risk Level Average risk score for all components in the software across all six measures is between 9.0 and 10.0

Software Supply Chain Integrity & Risk for Your Applications

Inherent Risk Level (IRL)	Description	Recommendation
Zero (0) (ZIRL)	Zero Inherent Risk Level. Average risk score for all components in the software across all six measures is zero.	Continue monitoring for changes in overall software to ensure risk remains None.
Low (0.1-3.9) (LIRL)	Low Inherent Risk Level Average risk score for all components in the software across all six measures is between 0.1 and 3.9.	Continue monitoring for changes in overall software to ensure risk remains low.
Medium (4.0-6.9) (MIRL)	Medium Inherent Risk Level Average risk score for all components in the software across all six measures is between 4.0 and 6.9.	Find opportunities to address some of the risk measures causing the average risk score to go to medium. Deploy risk mitigation solutions that target the identified high-risk components and vulnerabilities.
High (7.0-8.9) (HIRL)	High Inherent Risk Level Average risk score for all components in the software across all six measures is between 7.0 and 8.9.	Remove, update, or replace elevated risk components. Deploy secure coding practices including continuous security testing. Deploy risk mitigation solutions that target the identified high-risk components and vulnerabilities.
Critical (9.0-10.0) (CIRL)	Critical Inherent Risk Level Average risk score for all components in the software across all six measures is between 9.0 and 10.0.	Consider redeveloping the software starting with more secure components.

References

https://github.com/apache/echar ts	https://git hub.com/apache/dubbo
https://git hub.com/apache/groovy	https://git hub.com/apache/pinot
https://git hub.com/apache/iotdb	https://git hub.com/apache/arrow-datafusion
https://git hub.com/apache/commons-lang	https://git hub.com/apache/calcite
https://git hub.com/apache/flume	https://github.com/apache/incubator-hugegraph
https://git hub.com/apache/pulsar	https://git hub.com/apache/geode
https://git hub.com/apache/spark	https://git hub.com/apache/shiro
https://git hub.com/apache/avro	https://git hub.com/apache/mahout
https://git hub.com/apache/curator	https://git hub.com/apache/logging-log4j2
https://git hub.com/apache/activemq	https://git hub.com/apache/hudi
https://git hub.com/apache/linkis	https://git hub.com/apache/zookeeper
https://git hub.com/apache/maven-mvnd	https://git hub.com/apache/shenyu
https://git hub.com/apache/ignite	https://git hub.com/apache/shardingsphere-elasticjob
https://git hub.com/apache/cordova-android	https://git hub.com/apache/kylin
https://git hub.com/apache/dubbo-spring-boot-project	https://git hub.com/apache/dolphinscheduler
https://git hub.com/apache/cordova-ios	https://git hub.com/apache/maven
https://github.com/apache/air flow	https://git hub.com/apache/rocketmq
https://github.com/apache/incubator-seatunnel	https://git hub.com/apache/storm
https://git hub.com/apache/zeppelin	https://git hub.com/apache/hbase
https://git hub.com/apache/dubbo-admin	https://git hub.com/apache/kafka
https://git hub.com/apache/flink	

LINEAJE