SYNOPSYS®

[2023]
OPEN SOURCE SECURITY AND RISK ANALYSIS REPORT

[table of contents]

[introduction]

About the 2023 Open Source Security and Risk Analysis Report and the CyRC

In its 8th edition this year, the 2023 "Open Source Security and Risk Analysis" (OSSRA) report delivers our annual in-depth look at the current state of open source security, compliance, licensing, and code quality risks in commercial software. We share these findings with the goal of helping security, legal, risk, and development teams better understand the open source security and license risk landscape. The Synopsys Cybersecurity Research Center (CyRC) provides the data for this report. The CyRC's mission is to provide and publish security advisories and research that help organizations better develop and consume high-quality software.

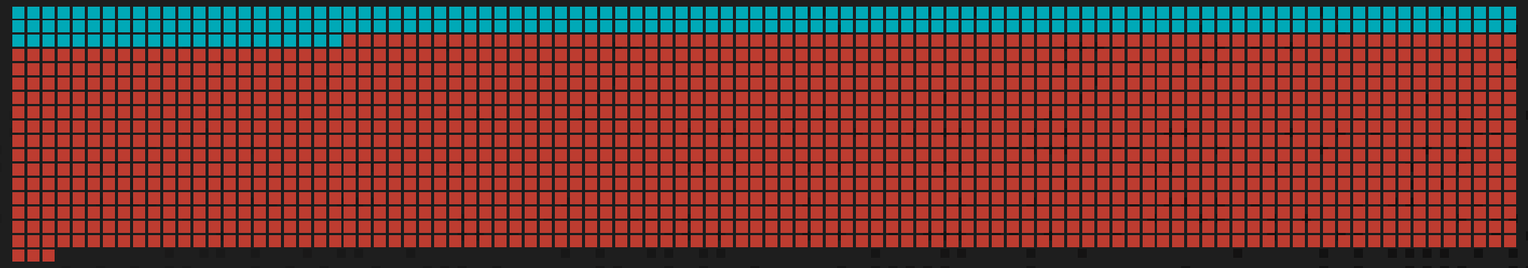
The annual OSSRA report represents CyRC findings from the previous year's data. Thus, our 2023 report is indicative of 2022 data. In 2022, the CyRC studied anonymized findings from over 1,700 commercial codebases across 17 industries. Our Audit Services team audits thousands of codebases for our customers each year, with the primary aim of identifying a range of software risks during merger and acquisition (M&A) transactions. Despite 2022's economic ambiguity and a corresponding slowdown in tech mergers and acquisitions, audit numbers remained promisingly strong.

The Synopsys Black Duck® software composition analysis (SCA) product team and the CyRC Audit Services team have helped security, development, and legal teams around the world strengthen their security and license compliance programs for almost 20 years. Black Duck SCA enables organizations to identify and track open source code and integrate automated open source policy enforcement across their existing development environments. Black Duck audits cover all aspects of software risk and are generally performed in the context of an M&A transaction. The audits also provide a comprehensive, highly accurate software Bill of Materials (SBOM) covering the open source, third-party code, web services, and application programming interfaces (APIs) in an organization's applications. The Audit Services team relies on data from the Black Duck KnowledgeBase™ to identify potential license compliance and security risks. This KnowledgeBase, sourced and curated by the CyRC, includes data on more than 6.1 million open source components from over 28,000 forges and repositories.

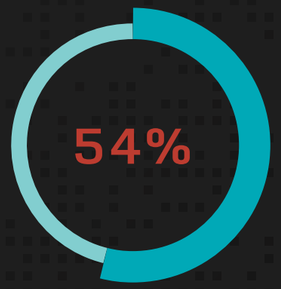
No matter what industry you operate in, or what role you play in relation to organizational security and risk, the OSSRA continues to highlight the ever-growing presence of the open source fueling your business–as well as the pitfalls of failing to effectively manage it. We say it every year: Open source is the foundation for every application we rely on today. Identifying, tracking, and managing open source effectively is therefore critical to a successful software security program. This report offers key recommendations and insights to help developers and consumers of open source better understand the open source ecosystem and how to manage it responsibly.

> **No matter what industry you operate in, the OSSRA continues to highlight the ever-growing presence of open source fueling your business and highlights the pitfalls of failing to effectively manage it.**

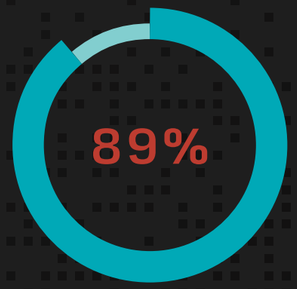
[overview]



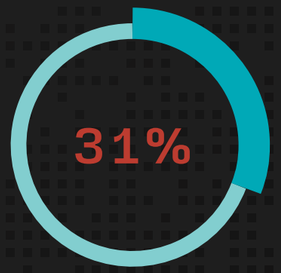
Of the **1,703** codebases scanned in 2022, **87%** included security and operational risk assessments.



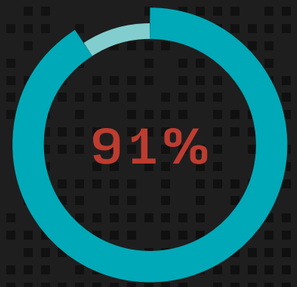
of codebases had license conflicts



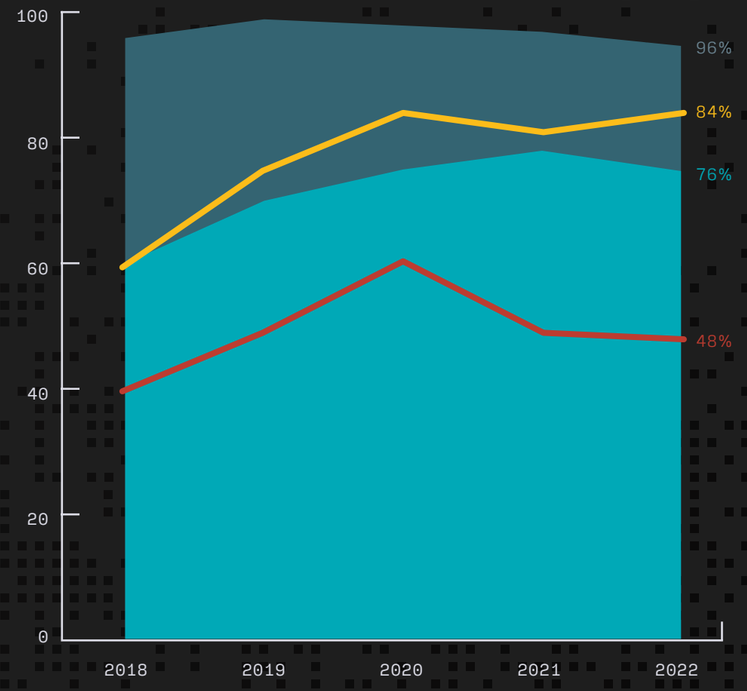
of codebases contained open source more than four years out-of-date



of codebases contained open source with no license or a custom license



of codebases contained components that had no new development in the past two years



Percentage of codebases containing open source

Percentage of code in codebases that was open source

Percentage of codebases containing at least one vulnerability

Percentage of codebases containing high-risk vulnerabilities

Open Source by Industry

Terminology

Codebase

The code and associated libraries that make up an application or service.

Binary analysis

A type of static analysis that is used to identify the contents of an application when access to the source code isn't possible.

Black Duck Security Advisory (BDSA)

Detailed, timely, consistent information on an open source vulnerability, researched and analyzed by the CyRC security research team. BDSAs provide Synopsys customers with early and supplemental notification of open source vulnerabilities and upgrade/patch guidance. Black Duck Security Advisories go beyond the National Vulnerability Database (NVD) and provide enhanced data, completeness, and accuracy, so you get early warning and comprehensive insight. BDSAs deliver same-day notification, actionable mitigation guidance and workaround information, severity scoring, references, and more.

Software component

Prewritten code that developers can add to their software. A software component might be a utility, such as a calendar function, or a comprehensive software framework supporting an entire application.

Dependency

A software component becomes a dependency when other software uses it—that is, when software becomes dependent on that component. Any given application or service may have many dependencies, which themselves may depend on other components.

Open source license

A set of terms and conditions stating end-user obligations when an open source component (or a snippet of a component's code) is used in software, including how the component may be used and redistributed. Most open source licenses fall into one of two categories.

Permissive license

A permissive license allows use with few restrictions. Generally, the main requirement of this type of license is to include attribution of the original code to the original developers.

Copyleft license

A copyleft license generally includes a reciprocity obligation stating that modified and extended versions are released under the same terms and conditions as the original code, and that the source code containing changes must be provided upon request. Commercial entities are wary of including open source with copyleft licenses in their software, as its use can call the overall codebase's intellectual property (IP) into question.

Software composition analysis (SCA)

A type of application security tool used to automate the process of open source software management. SCA tools integrate within the software development lifecycle to identify the open source used in a codebase, provide risk management and mitigation recommendations, and perform license compliance verification.

Software Bill of Materials (SBOM)

A comprehensive inventory of the software components and dependencies in a codebase, often generated by a software composition analysis tool. As phrased by the National Telecommunications and Information Administration (NTIA), "An SBOM should include a machine-readable inventory of your software components and dependencies, information about these components, and their hierarchical relationships." Since SBOMs are intended to be shared across companies and communities, having a consistent format (that is both human- and machine-readable) with consistent content is critical. U.S. government guidelines currently specify two standards as approved formats: Software Package Data Exchange (SPDX) and CycloneDX.

Executive Order 14028

U.S. President Biden issued an order titled "Improving the Nation's Cybersecurity" in May 2021, instructing various agencies of the federal government to create software security guidelines for companies doing business with the federal government. This order includes a timeline for activities that, as of the writing of this report, do not mandate contractual obligations. However, despite the lack of hard requirements, the order has prompted many organizations to re-examine their security practices and scrutinize their level of software security risk. The use of an SBOM is a key element promoted by EO 14028, as it facilitates the communication of software supply chain information between producers and consumers of software.

Apache Log4J2 vulnerability(ies) (BDSA-2021-3887, CVE-2021-44228, et al)

The open source component Apache Log4J2 (commonly known as Log4J) is broadly used within the Java community to implement application logging. Several vulnerabilities have been identified in Log4J, including remote code execution (RCE), denial of service, and LDAP vulnerabilities.

Zero-day vulnerability

A vulnerability either unknown to those who would be interested in its mitigation (including the vendor or creator of the target software) or known and without a patch to correct it.

OpenSSL vulnerabilities

In November 2022, OpenSSL, a popular open source command-line tool, released an advisory warning of two critical vulnerabilities (CVE-2022-3602 and CVE-2022-3786). Their severity level was later downgraded to "high." These are buffer overflow/overrun vulnerabilities in the certificate validation. Exploit of the former vulnerability could lead to crashes and introduce the potential of arbitrary code execution, and exploit of the latter could lead to memory corruption issues.

Buffer overflow / overrun vulnerability

Buffers are used for temporary memory storage during the execution of an application. When the volume of data being written to a buffer exceeds the buffer's capacity, a buffer overflow or overrun occurs, which can lead to crashes, memory issues, or other unexpected behavior. Attackers can exploit this vulnerability to accomplish tasks like altering files and accessing sensitive information.

[vulnerabilities and security]

Open Source Vulnerabilities and Security

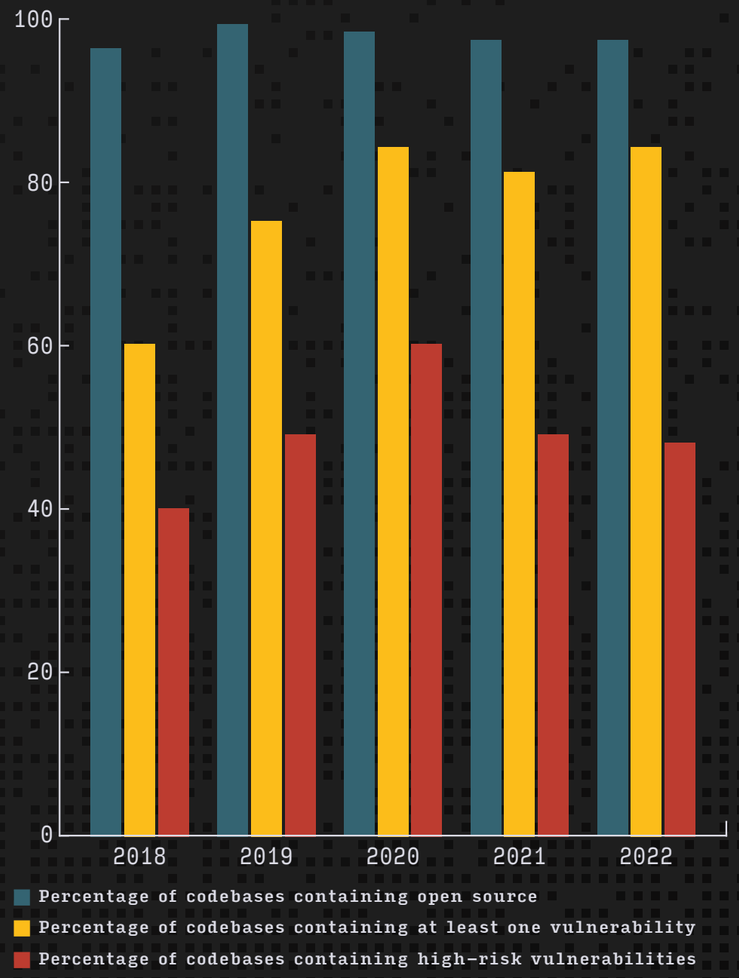
Of the 1,703 codebases analyzed by the Black Duck Audit Services team for this year's report, 96% contained open source. Seventy-six percent of the total codebases scanned were open source, meaning 76% of the total code in our study was open source code.

The average number of open source components in a given application this year was 595. When monitoring for security vulnerabilities and performing security maintenance activities, what might be practical for a small number of components becomes overwhelming and virtually impossible at this scale, and demands the use of an automated solution like SCA.

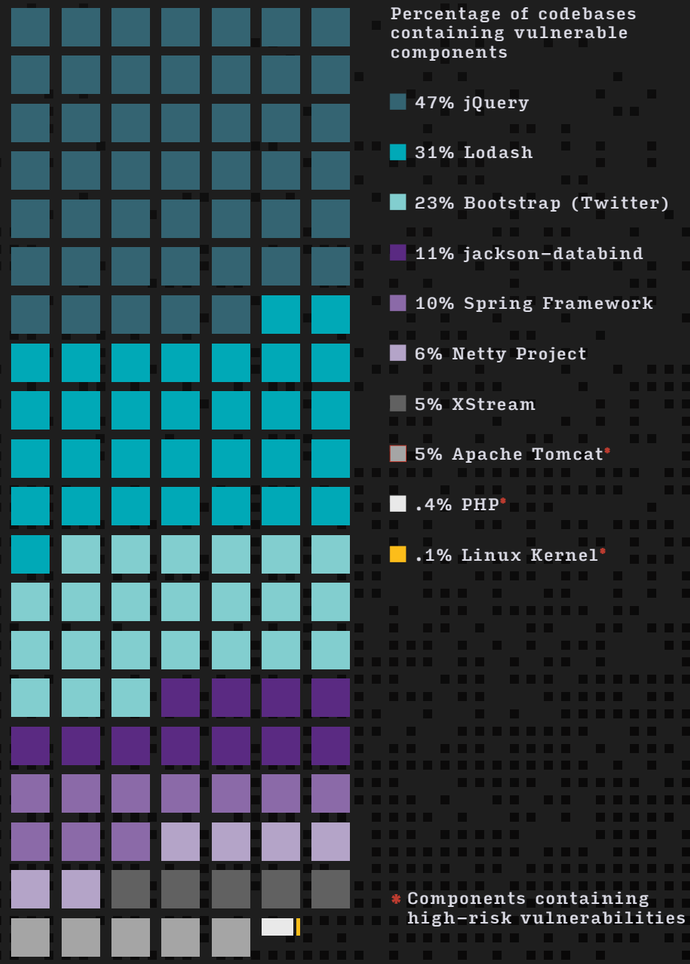
Eighty-four percent of codebases contained at least one known open source vulnerability, an almost 4% increase from the 2022 edition of the OSSRA report. And 48% of the codebases we examined contained high-risk vulnerabilities, down only 2% from last year. High-risk vulnerabilities are those that have been actively exploited, already have documented proof-of-concept exploits, or are classified as remote code execution vulnerabilities.

All Black Duck audits examine open source license compliance. Customers can opt out of the vulnerability / operational risk assessment portion of the audit at their discretion. In this 2023 edition, the Black Duck Audit Services team conducted 1,703 audits. Of those audits, 87% (1,481) opted to undergo a vulnerability / operational risk assessment. The data in the "Vulnerabilities and Security" and "Open Source Maintenance" sections of the 2023 OSSRA report is based on the 1,481 codebases that included risk assessments, whereas the data in the "Licensing" section is based on all 1,703 codebases.

Vulnerabilities and High-Risk Vulnerabilities



Components with Vulnerabilities



The Gordian Knot: Open Source Software Risk and Supply Chain Security

A recent research report, "Walking the Line: GitOps and Shift Left Security," cosponsored by Synopsys and the Enterprise Strategy Group, explored market concerns and how organizations are tackling current security challenges. Seventy-three percent of organizations surveyed for that report said they had "significantly increased their efforts to secure open source software, container images, and third-party software components as a result of recent software supply chain attacks." Troublingly, 34% of respondents also stated that they had experienced "exploit(s) that took advantage of known vulnerabilities in open source software" within the last 12 months.

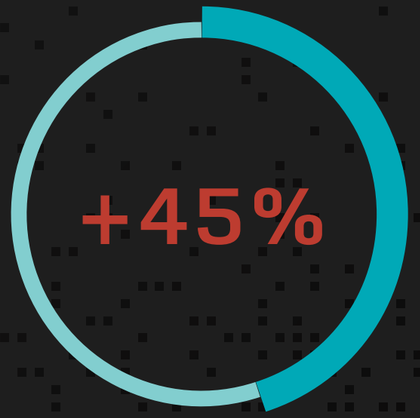
By now, anyone remotely involved in software security is likely concerned with the software supply chain. Software supply chain security has dominated the news and touched organizations across industry verticals. But nearly two years out from President Biden's executive order (EO 14028), organizations are still struggling with supply chain basics—understanding the breadth of their software supply chain, establishing visibility into the software they depend on, and satisfying growing transparency pressures for the software they distribute and sell.

So what's to be done? The first step in securing the software supply chain is managing the open source and third-party code in your applications.

If you can't effectively manage and ensure the security of your open source and third-party software, no other efforts made toward securing your supply chain will work—or frankly, even matter. Managing this software entails gaining complete visibility into your dependencies and having the ability to easily gather information pertaining to the risk introduced by these components. Once this risk has been identified, you need tooling and practices in place to manage, prioritize, and remediate it.

Further, communicating any identified risk to key stakeholders is also important, as it provides visibility into and support of risk management activities and initiatives. Additionally, all these capabilities and practices should be built into existing development pipelines, leveraging automation wherever possible.

If it sounds complex, that's because it is. The final product of your supply chain, as well as its users, are affected by every component, person, activity, material, and procedure involved in its creation. Only with complete visibility into your supply chain, and its numerous inputs, can you begin to secure it. And this visibility begins with *verifying* that you are truly secure. The old Russian proverb "Trust, but verify" has never been more fitting; managing your open source and third-party software means you have verified its security. Without verification, you're placing baseless trust in the weakest links of your supply chain.



of organizations worldwide will have experienced attacks on their software supply chains by 2025

–Gartner

Open source software risk and supply chain security are inextricably linked.

So what's to be done? The first step in securing the software supply chain is managing the open source and third-party code in your applications.

Vulnerabilities in Industries

Each year, we watch the percentage of codebases containing open source rise, and this year, even the industries with the smallest amount (Manufacturing, Industrials, and Robotics) still had open source in 92% of their codebases.

We were troubled, however, when we looked at the presence of open source in tandem with vulnerabilities; of particular note was the Aerospace, Aviation, Automotive, Transportation, and Logistics sector. All—100%—of the codebases we examined in this sector contained open source, and open source made up 73% of its code. Sixty-three percent of its codebases contained vulnerabilities classified as high risk (those with a severity score of 7 or higher).

We saw more of the same with the Energy and Clean Tech sector, in which 78% of the total code was open source and 69% contained high-risk vulnerabilities. The total open source identified within this sector's codebases was 95%.

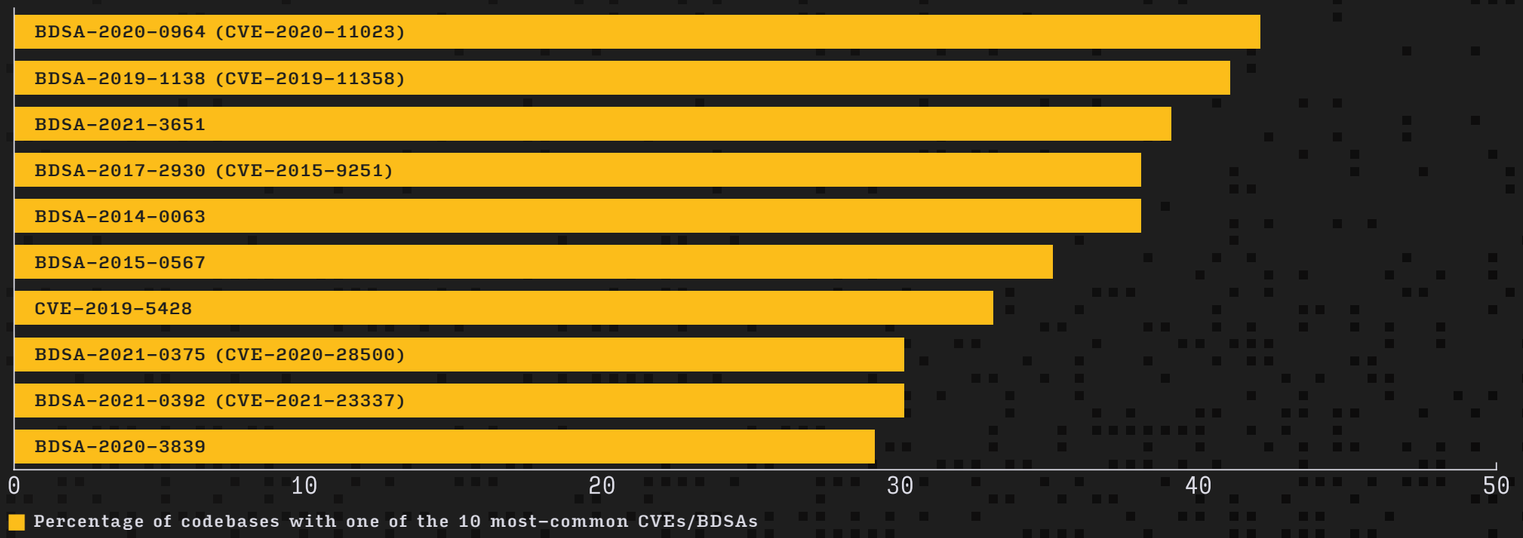
Similar findings, to lesser degrees, played out across industries, painting a picture that security teams would do well to heed. Open source was in nearly everything we examined this year; it made up the majority of the codebases across industries, and it contained troublingly high numbers of known vulnerabilities that organizations had failed to patch, leaving them vulnerable to exploit. It is crucial to understand that while open source itself does not pose any inherent level of risk, failing to manage it does.

Open Source and Vulnerabilities by Industry

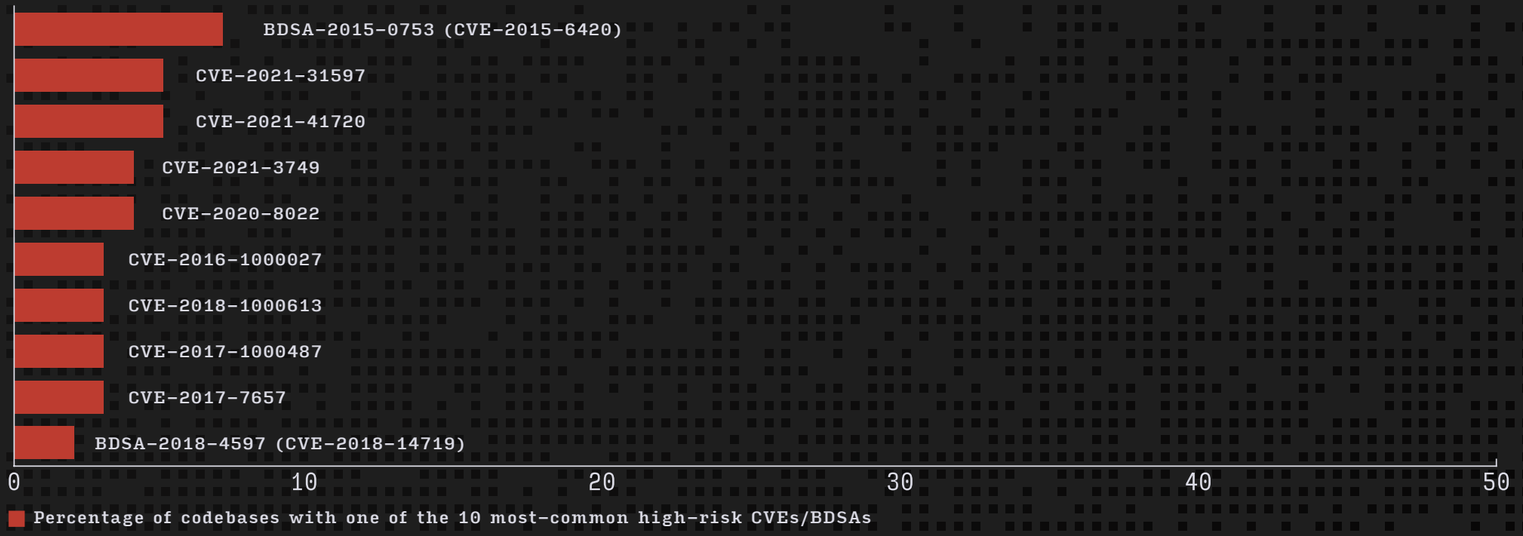
Vulnerability Severity Scoring

The Synopsys vulnerability severity scoring system gathers as many variables as possible when determining a score. These scores are part of our Black Duck Security Advisories (BDSAs). BDSAs leverage the CVSS scoring system, as specified by FIRST.org, to assign severity scores in alignment with CVSS, but Synopsys vulnerability severity scores are assigned by the CyRC, rather than simply parroting those issued by the NVD. When assigning scores, BDSAs take many factors, such as exploitability, into consideration, helping to ensure the most precise CVSS score. In addition, BDSAs include temporal metrics in scoring considerations, whereas sources like the NVD do not. Our aim is to provide the most finely tuned and accurate scores possible, helping customers prioritize triage activities accurately.

CVEs/BDSAs



High-Risk CVEs/BDSAs



Five-Year Rewind

This year, we took a five-year lookback at OSSRA report data, with the goal of identifying notable trends. Here's what we found.

Finding 1. Open source adoption varies by industry

We report each year on the continued growth and adoption of open source, but the five-year view reveals disparities in adoption across verticals. As we noted earlier, all verticals this year contained open source in upward of 92% of their codebases. However, when we look at the total percentage of open source in codebases over the past five years, we see great variation.

From 2018 to 2022, the percentage of open source code within scanned codebases grew by 163% in EdTech; 97% in Aerospace, Aviation, Automotive, Transportation, and Logistics; and 74% in Manufacturing and Robotics. We attribute EdTech's explosive open source growth to the pandemic; with education pushed online and software serving as its critical foundation, this growth makes sense. Open source is free (although it must be managed properly), making it particularly attractive to budget-conscious industries looking to make significant enhancement in a short period of time. Many EdTech systems are home-grown and maintained by volunteers, so open source has likely served as the footing for new and emerging EdTech technologies.

When examining the Aerospace, Aviation, Automotive, Transportation, and Logistics vertical, though, perhaps the slower adoption of open source (with a sudden 22% jump just since last year) can be attributed to these industries being highly regulated. Maybe there has been more intentional avoidance of open source in the past, as organizations in these industries lacked adequate resources and capabilities to effectively secure open source.

Open Source and High-Risk Vulnerabilities by Industry

Joe Jarzombek, former director of government and critical infrastructure programs at Synopsys, shed another perspective on the numbers. Jarzombek noted that "technical debt from poor software quality hampers delivery of new capabilities. Like other parts of industry and government, the defense industrial base spends more time and effort correcting technical debt than on proactive, creative, or preventive work." He went on to note that the recent Cybersecurity Maturity Model Certification has helped to "alleviate security-conscious data protection concerns associated with many sectors," thereby freeing these industries to be more innovative and agile.

That is to say, slower adoption of open source--and the innovation, speed, and dexterity it provides--allows archaic practices to persist. But technical debt and heavy regulations are slowly giving way, enabling more usage of open source.

Finding 2. Organizations aren't fixing high-risk vulnerabilities

When we compared the incidence of high-risk vulnerabilities by industry, we saw the same story, to varying degrees, across all industries. Since 2018, there has been a minimum increase of 42% in high-risk vulnerabilities (in the Marketing Tech sector). The number of high-risk vulnerabilities has jumped as much as 557% (in the Retail and eCommerce sector) since 2018, a real reason for concern.

Considering the Aerospace, Aviation, Automotive, Transportation, and Logistics vertical again (which saw a massive 232% increase in high-risk vulnerabilities), could this be a case of simply not needing to mitigate risk? A key driver of vulnerability remediation is the likelihood and potential impact of exploitation. Much of the software and firmware used in these industries operate within closed systems, which can reduce the likelihood of an exploit and may lead to a lack of urgency in the need to patch it. Furthermore, vulnerabilities can be mitigated in other ways that don't necessarily involve updating components.

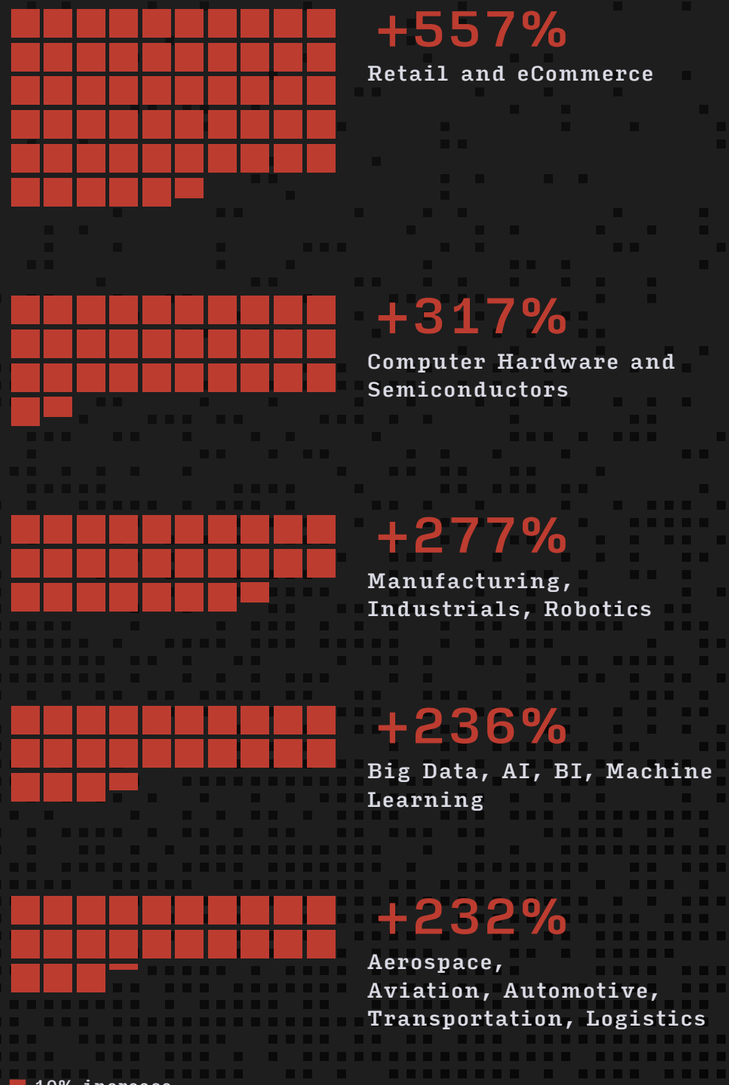
If we consider how the software in this vertical is deployed, distributed, and operated, we find another possible explanation for the number of high-risk vulnerabilities surfaced. Embedded software and firmware are often used to power hardware that does not have access to a network where updates can easily and automatically be issued, as they can for a SaaS application. When applying patches means having to download and install newer versions, or physically plug a thumb drive into a device, the patches will be applied less frequently, resulting in more unpatched vulnerabilities.

Additionally, software in this vertical is constructed with different standards and practices than those used by independent software vendors and companies that build enterprise and SaaS applications. Perhaps the failure to address high-risk open source software vulnerabilities is a symptom of this distinction.

Looking at the Internet of Things (IoT) vertical, we see a different story. Since 2020, 100% of the codebases we have scanned contained open source. The total amount of open source in each codebase has increased, too; it's up 35% since 2018, with 89% of the total code being open source code. The IoT is a fantastic representation of the benefits of open source; IoT organizations (for example, the smart devices offered by Ring, Amazon, and Nest) are under extreme pressure to produce new software, fast. In an industry with strong competition, open source allows organizations to be quick on their feet. Without open source, it wouldn't be possible to keep up with the breakneck pace of development. The downside though, is the presence of vulnerabilities.

The IoT vertical has seen a 130% increase in high-risk vulnerabilities since 2018 and this year, 53% of its audited applications contained high-risk vulnerabilities (one of the higher percentages in our findings). This is particularly concerning when we think about the utility of IoT devices; we connect many aspects of our lives to these devices and trust in the inherent safety in doing so. IoT devices power our lights on automated schedules, so they contain data about when we are home and when we are out. We use cameras that have images of the inside our homes, smart locks on our front doors, and baby monitors to watch over children.

Largest Increases in High-Risk Vulnerabilities Over Five Years, by Industry



[licensing]

Open Source Licensing

The Black Duck Audit Services team found that 54% of all 2022 audited codebases contained open source with license conflicts, a minimal increase of 2% from last year, but still a dramatic decrease (down 17%) from 2020, when the number was 65%.

The Creative Commons ShareAlike 3.0 (CC BY-SA 3.0) license was the most prevalent cause of license conflict this year; 22% of audited codebases had some form of conflict with that license. The CC BY-SA 3.0 license conflict numbers illustrate an often overlooked issue when it comes to open source licenses. Both commercial and open source developers often introduce code snippets, functions, methods, and operational pieces of code into their software, generally termed "dependencies" (because the overarching software is dependent on that code). Therefore, open source projects frequently contain subcomponents licensed under a different license than the overall project, with terms and conditions that go beyond those in the primary license, and that creates a conflict.

Eighty-five percent of the license conflicts found by the Black Duck Audit Services team involved content pulled from Stack Overflow, an online Q&A platform for finding and sharing technical knowledge. The popularity of Stack Overflow, along with the fact that the CC BY-SA 3.0 license is highlighted as a conflict within both distributed and SaaS deployment models, makes it unsurprising that it was the greatest offender.

By industry, we saw a dramatic decrease in open source license conflicts. Last year, one industry (Computer Hardware and Semiconductors) had 93% of the codebases with open source license conflicts. That number was down to 75% this year. Overall, we saw similar improvements across the board; the vertical with the highest percentage of open source license conflicts this year was IoT, with 78%. This collective improvement in open source license conflicts might be due to economic uncertainty and general anxiety around supply chain risk.

Open Source and License Conflicts by Industry

Understanding License Risk

In the U.S. and many other jurisdictions, creative work (including software) is protected by exclusive copyright by default. No one can legally use, copy, distribute, or modify that software without explicit permission from the creator/author in the form of a license that grants the right to do so. Even the friendliest open source licenses include obligations the user takes on in return for use of that software.

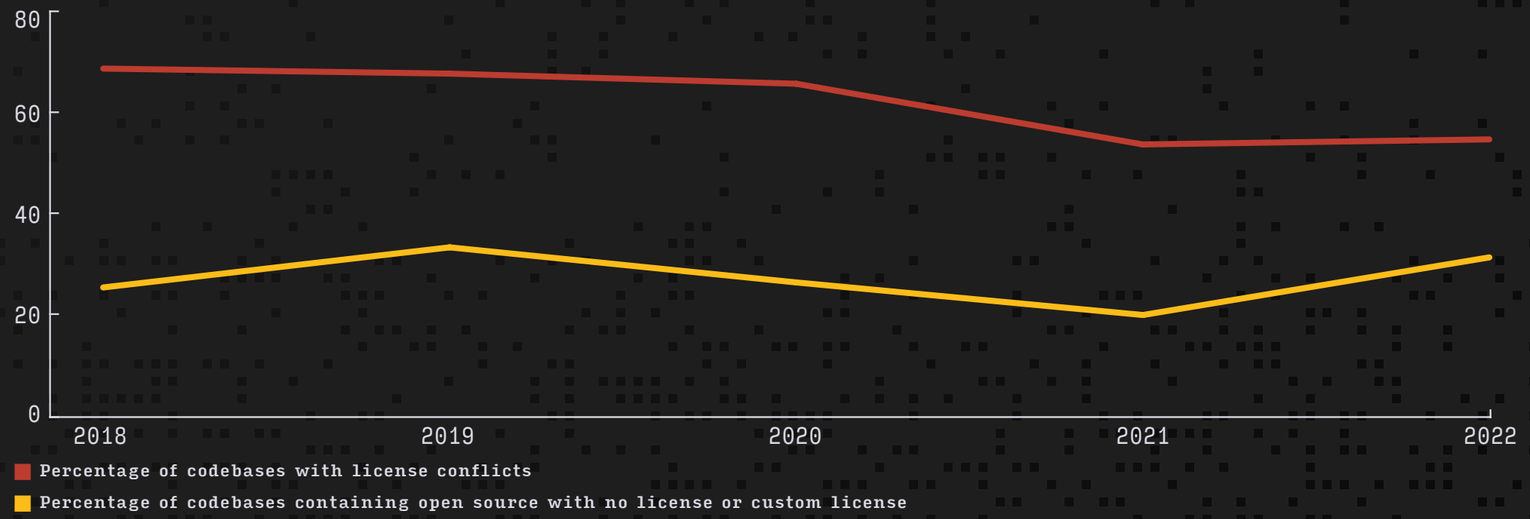
Potential license risk arises when a codebase includes open source with licenses that appear to conflict with the overall license of the codebase. The GNU General Public License (GPL) is the most common copyleft license applied to open source projects. Conflicts can arise when the code licensed under GPL is included in commercial, closed source software.

Thirty-one percent of the 2022 audited codebases were using code with either no discernible license or a customized license, a 55% increase from last year. There are two possible reasons for this finding. First, it could mean that developers manually added snippets or partial components into the codebases. When doing so, developers usually fail to bring the snippet's associated licenses along with it. Second, this number could simply mean that the maintainers of a project are making licenses and terms that stray from the bevy of known and standard licenses. While not problematic on its face, not fully understanding the implications of a license can be risky and should be avoided whenever possible.

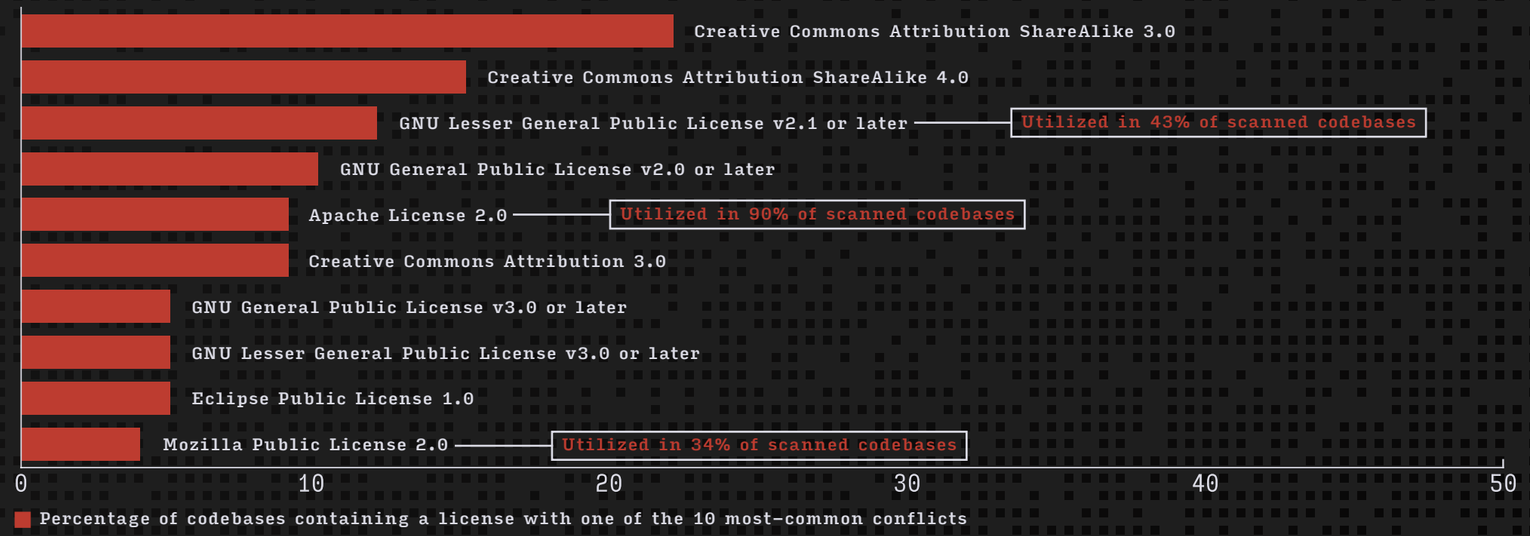
Unlicensed code can be found in GitHub repositories where a developer has made code publicly available but with no indication of license, either in the code, a .txt file, or metadata associated with the project. Or developers might copy code from a blog or website, assuming that if someone is making the source code available, it must be OK to use. This might seem reasonable, but copyright law begs to differ.

Further, variants or customized versions of standard open source licenses can place undesirable requirements on the licensee and require legal evaluation for possible IP issues or other implications. The JSON license is a prime example of a customized license. Based on the permissive MIT license, the JSON license adds the restriction that "The software shall be used for good, not evil." The ambiguity of this statement leaves its meaning up to interpretation, and many lawyers would advise against using software so licensed, especially in the context of M&A scenarios.

Issues in Licensing



Conflicts in Licenses



[open source maintenance]

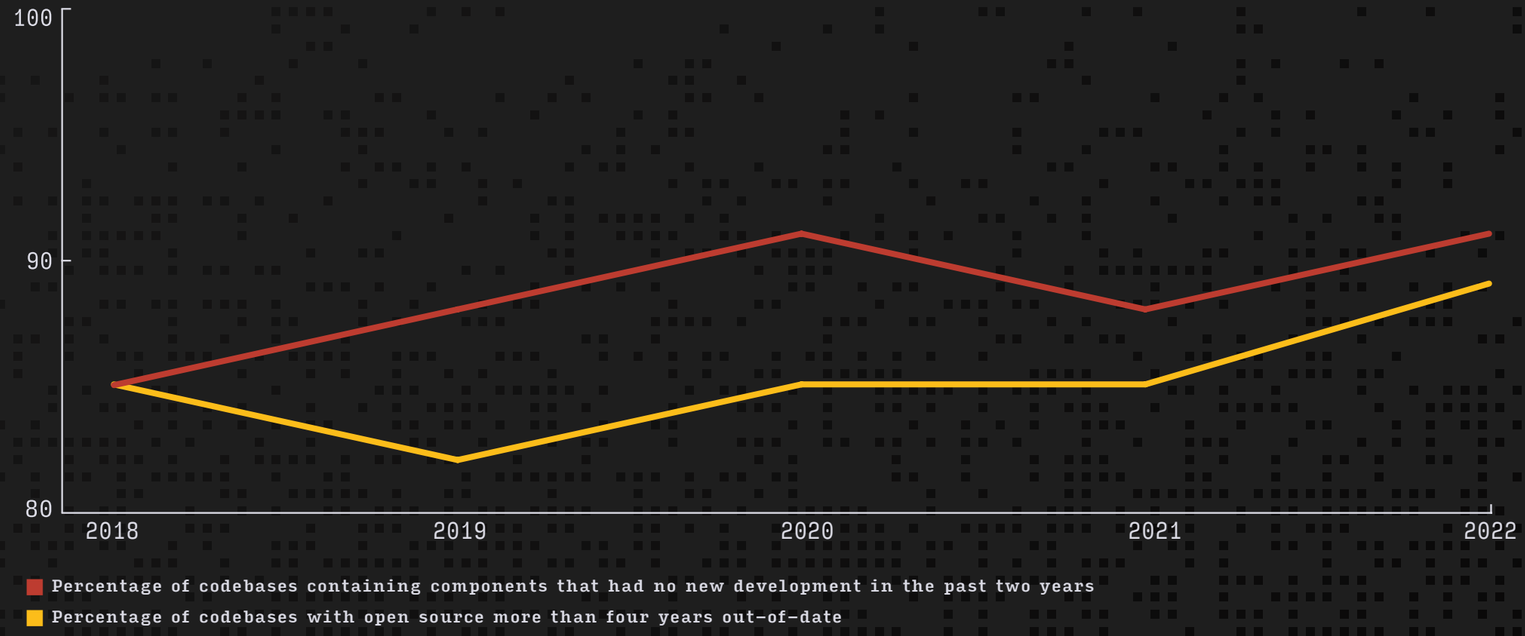
Maintenance by Open Source Developers

Ideally, developers will select open source components supported by robust communities. Linux, for example, is improved every day by thousands of developers from hundreds of organizations. However, of the 1,481 codebases examined by the Black Duck Audit Services team that included risk assessments (see page 7), 91% contained open source that had no development activity in the last two years—no feature upgrades, no code improvements, and no security issues fixed over the past 24 months. This probably means that the project is no longer being maintained, especially in the case of smaller projects.

Open source projects, by definition, are the product of an undefined number of contributors and maintainers. This structure makes open source a collaborative effort, but the challenge is the lack of incentive for contributors to perform maintenance activities. Important projects like Kubernetes often have healthy support, but there are also plenty of projects maintained by only a handful of people.

A recent commentary on open source projects noted that there is "a chunk of people who are maintaining vital—and heavily depended-on—projects with little recognition." While some organizations (Microsoft, RedHat, Google, etc.) have incentive programs in place to motivate open source project maintenance and participation, the vast majority do not. The result is foundational open source that fuels the software we rely on every day going untouched, sometimes for years.

Sustainability Within Codebases



A very newsworthy example of abandoned open source projects in the past few months is Twitter, which has "turned its back on open source development," according to Will Norris, Twitter's former open source lead. "Most of the key people that were working on open source at Twitter have left. All of the engineers that I worked with on open source are gone. With new management in place and new business priorities, open source projects that were once high priorities are being deprioritized in favor of newer and what are deemed to be more important initiatives," he said.

The major challenge with situations like what we are seeing at Twitter is the fallout from lack of maintenance. If people aren't maintaining a project, the best-case scenario is technical debt—and the worst-case scenario is security risk.

Beyond Known Risk

Aside from abandoned projects, there are also cases of malicious sabotage of open source projects. In the context of supply chain security, we often hear of organizations identifying and mitigating known vulnerabilities, but what about emerging forms of open source software risk that are gaining attention within the context of the supply chain? Malicious open source software packages, "protestware," dependency confusion, and typo-squatting all pose new and concerning threats to open source security.

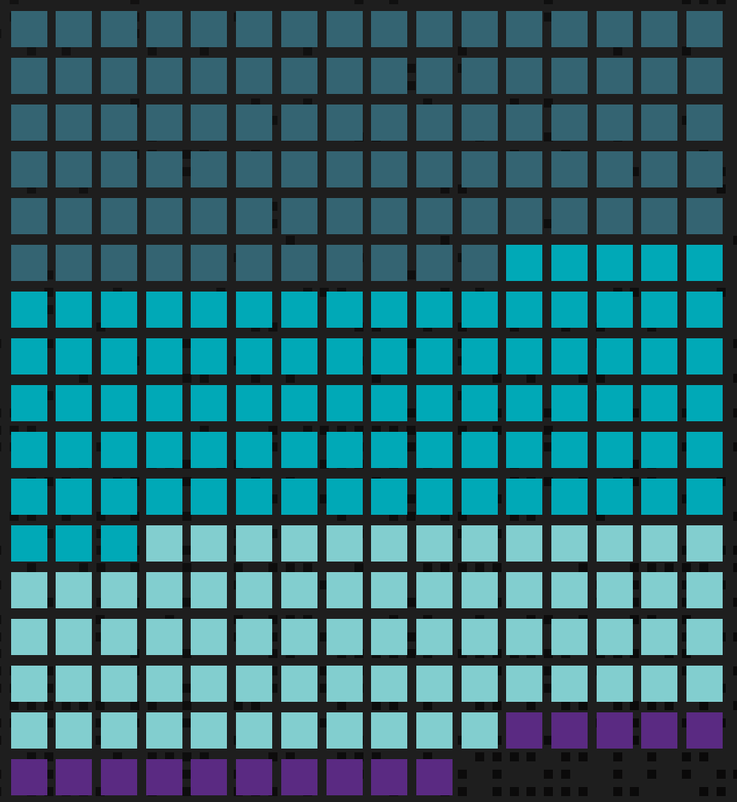
Looking back at breaches and notable vulnerabilities over the past five years or so, a common theme is trust. Organizations and end users must have some level of trust in the software they use, and in the people who develop and supply it. Simply put, companies are trusting that every node in their supply chain has the same security and quality safeguards as they do—a dangerous assumption.

A recent report, "The Cost of Poor Software Quality in the US," cosponsored by Synopsys and the Consortium for Information and Software Quality cited ransomware, crypto jacking, IoT and OT attacks, and supply chain attacks as some of the hottest cybercrime trends of 2022. We see the theme of trust here again: We trust that open source maintainers and developers will find and fix vulnerabilities. We trust that open source contributors have the same mindset and motivations as us. Unfortunately, this is not always the case. It's important to remember that operational risk goes beyond known vulnerabilities. Lessening operational risk is also about anticipating future threats and taking steps to prevent them.

Last summer, several malicious packages were found in npm, (the default package manager for the JavaScript runtime environment Node.js.), that had the capability of harvesting sensitive data from forms embedded in mobile applications and websites. The packages, downloaded thousands of times, typo-squatted other popular and trustworthy packages. Any version of these packages should still be considered vulnerable and malicious. (Examples include Icon-package, Ionicio, Ajax-libs, and more.)

As you read about this example, did you have an idea of whether you use npm? Have those responsible for security at your organization properly examined it and ensured that you're secure? If not, you might be a bit too trusting. It is vital to have a clear grasp of the reputation of your projects and who maintains them, as their motivations for doing so will likely differ from yours. Maintainers with bad intentions could take advantage of your trust in your supply chain and work malicious code into your applications without your due diligence.

Issues with Versioning



- 91% of codebases contained components that were not the latest version
- 88% of codebases contained components with no activity for the last 24 months and latest version of the component not in use
- 72% of codebases contained components with no activity for the last 24 months and latest version of the component in use
- 15% Codebases using more than 10 versions of a component

Maintenance by Open Source Consumers

Of the 1,481 codebases examined by the Black Duck Audit Services team that included risk assessments (see page 7), 91% contained outdated versions of open source components. That means an update or patch was available but had not been applied.

There can be valid reasons for not keeping software up-to-date. A DevSecOps team might determine that the risk of unintended consequences outweighs whatever benefit would come from applying the newer version. Embedded software may be at minimal risk from vulnerabilities that can only be introduced from an external source. Or it could be a time/resources issue. With many teams already stretched to the limit building and testing new code, updates to existing software can become a lower priority except for the most critical issues.

It seems most likely that the majority of the 91% is due to the DevSecOps team not knowing that there is a newer version of the open source component available—if they are aware of the component at all. Organizations are often most familiar with commercial software, for which patches and updates are pushed automatically, so they don't have to concern themselves with monitoring for updates. Open source functions quite differently. Implied with the use of open source is the user's responsibility to be aware and in control of a component's security and stability, and to seek out new versions and patches for the components as they become available.

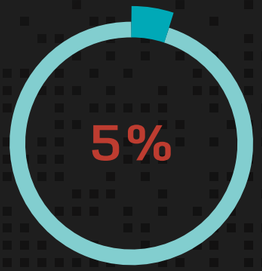
Failure to Patch

Over a year later, Log4Shell is a perfect example of a longtail vulnerability. Despite the media attention it received, and the numerous avenues organizations can take to confirm its presence in their codebases and remediate it, we still identified it in our audits this year. We found vulnerable versions of Log4J in 5% of the total codebases we scanned, and in 11% of the Java codebases.

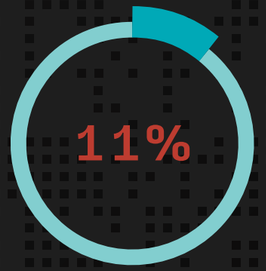
Most likely, we are still seeing vulnerable versions of Log4J because organizations simply don't know they have it. Log4J is a foundational component, which in many cases likely makes it less visible to security teams. Patching efforts for foundational components are also more complex than for superficial client-side components, so perhaps teams know they have it but simply have yet to resolve it.

We expect to see vulnerabilities like Log4J in organizations' foundational components for years to come. The reasons for this are first that organizations place inherent trust in open source. They trust that it is undergoing security checks and practices like those that come with commercial software. Second, organizations fail to identify open source components in their applications. Whether this is from a lack of security practices or capabilities, or both, is unclear.

The first step toward squashing business risk from open source, proprietary, and commercial code involves a comprehensive inventory of all software the business uses, regardless of where it came from or how it was acquired. Only with this complete inventory—a trustworthy SBOM—can security teams chart a path forward, with plans in place to address risk stemming from new security disclosures like Log4Shell.



of audited codebases contained a vulnerable version of Log4J



of Java codebases contained a vulnerable version of Log4J

[conclusion]

"Trust, but Verify"

Hailing from a wartime origin, "Trust, but verify" feels oddly appropriate in the context of today's influx of supply chain attacks. Trusting the software you use and develop is not itself a problem, but failing to validate that it has undergone necessary security analysis is.

Even Gartner is cementing the urgency around implementing adequate supply chain security tactics. Gartner predicts that "by 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021." In an environment of digital warfare, how can an organization best prepare for the unpredictable?

The Problem with Trust

We discussed earlier how misplaced trust in the security of an application (or the code that comprises it) can lead to potentially devastating consequences (read: security breaches). This built-in trust should be replaced by thinking like an attacker. Threat actors are, at their core, opportunistic. Their target will always be the smoothest path forward, with the largest yield. Without assurance that your software is secure, how can you be sure you're not an easy target?

It's impossible to predict every avenue of attack, but fostering organizational ideology around business risk can help lessen the threat. All organizations today rely on software—and that means they are in the software business. Once you start thinking about the security of your organization through this lens, you can appreciate the risk of simply trusting that your software is secure. It is your responsibility to secure your own software, whether inherited or developed internally.

Verify with an SBOM

In the fight against software supply chain attacks, an SBOM should be your weapon of choice. The concept of an SBOM derives from manufacturing, where the classic Bill of Materials is an inventory detailing all the items included in a product. When a defective part is discovered, the manufacturer knows which of its products is affected and can begin the process of repair or replacement. Similarly, maintaining an accurate, up-to-date SBOM that inventories open source components is necessary to ensure that code remains high quality, compliant, and secure. As in manufacturing, an SBOM of open source components allows you to pinpoint at-risk components quickly and prioritize remediation appropriately. A comprehensive SBOM lists all open source components in your applications as well as those components' licenses, versions, and patch status—the perfect defense against supply chain attacks.

The purpose of an SBOM is to help manage open source and third-party code usage, offer visibility into the "ingredients" of your applications, and standardize how that information is communicated. In addition to its basic function as a document or record, we encourage you to view SBOMs as a management system or a collective effort of tooling, practices, and procedures. You should be able to identify open source components with origin awareness, and then map those components to vulnerability data. Effective open source management helps encourage conversations around strategy and needed security program improvements for successful supply chain risk management.

In 2022, where 96% of commercial code contained open source, getting visibility into the components used in your applications should be a baseline requirement for any modern DevSecOps program. With insight into your business risk and overall security, you no longer have to *trust* that you are secure; you can *verify* it.

Additional Reading

- White paper: Walking the Line: GitOps and Shift Left Security
- Article: M&A Activity Looks Anemic Heading to Year-End as LBOs Shrivel
- Article: Global M&A market slows in 2022 first half—but shows signs of strength
- Wikipedia entry: Trust, but verify
- Press release: Strategic Direction for Cybersecurity Maturity Model Certification (CMMC) Program
- Webpage: Improving the Nation's Cybersecurity: NIST's Responsibilities Under the May 2021 Executive Order
- Interview: 1:1 with Joe Jarzombek
- Article: Twitter turns its back on open-source development

The Synopsys difference

Synopsys provides integrated solutions that transform the way you build and deliver software, accelerating innovation while addressing business risk. With Synopsys, your developers can secure code as fast as they write it. Your development and DevSecOps teams can automate testing within development pipelines without compromising velocity. And your security teams can proactively manage risk and focus remediation efforts on what matters most to your organization. Our unmatched expertise helps you plan and execute any security initiative. Only Synopsys offers everything you need to build trust in your software.

©2023 Synopsys, Inc. All rights reserved. Synopsys is a trademark of Synopsys, Inc. in the United States and other countries. A list of Synopsys trademarks is available at www.synopsys.com/copyright.html. All other names mentioned herein are trademarks or registered trademarks of their respective owners. April 2023

SYNOPSYS®