SYNOPSYS®

2024 Open Source Security and Risk Analysis Report

Your guide to securing your open source supply chain

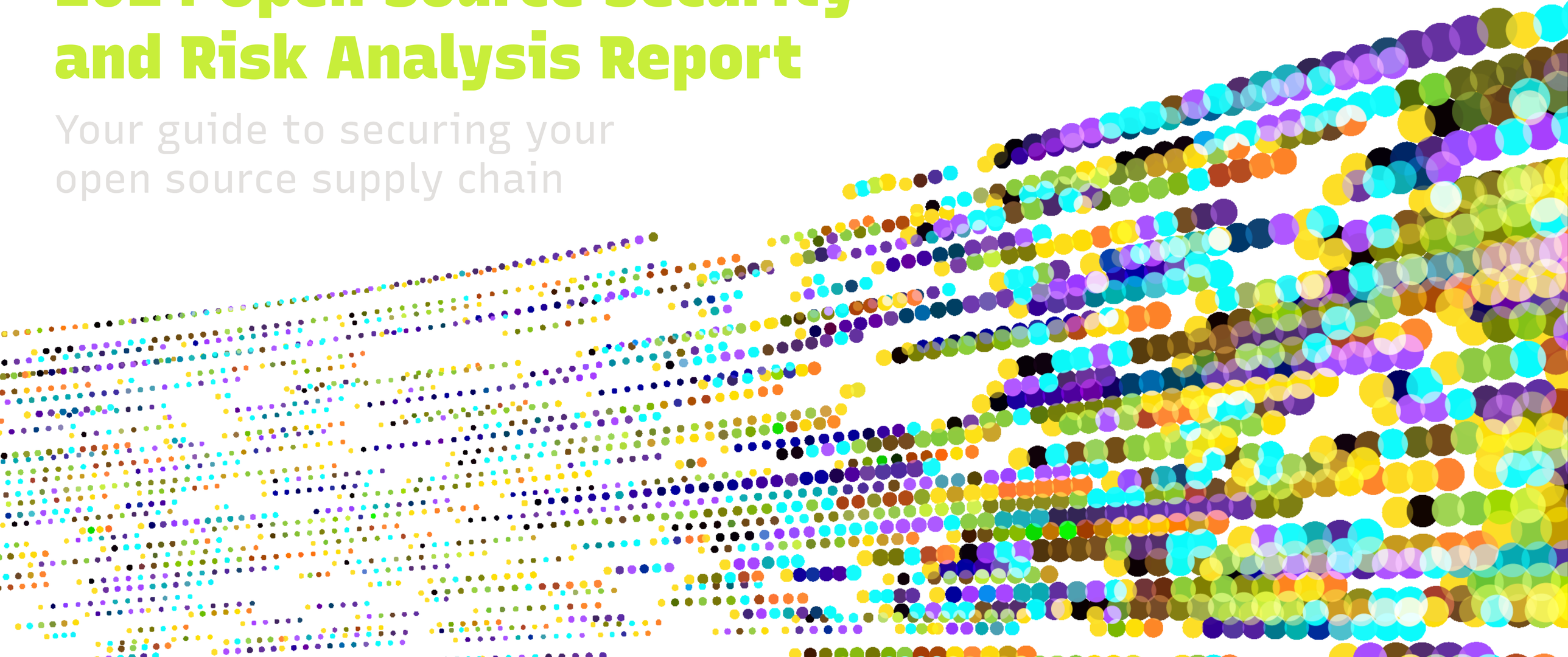


Table of Contents

Executive Summary

This report offers recommendations to help creators and consumers of open source software manage it responsibly, especially in the context of securing the software supply chain. Whether a consumer or provider of software, you *are* part of the software supply chain, and need to safeguard the applications you use from upstream as well as downstream risk. In the following pages, we examine

- Persistent open source security concerns
- Why developers need to improve at keeping open source components up-to-date
- The need for a Software Bill of Materials (SBOM) for software supply chain management
- How to protect against the security and IP compliance risk introduced by AI coding tools

For nearly a decade, the major theme of the "Open Source Security and Risk Analysis" (OSSRA) report has been *Do you know what's in your code?* In 2024, it's a question more important than ever before. With the prevalence of open source and the rise in AI-generated code, more and more applications are now built with third-party code.

Without a complete view of what's in your code, neither you, your vendors, nor your end users can be confident about what risks your software may contain. Securing the software supply chain begins with knowing what open source components are in your code, as well as identifying their respective licenses, code quality, and potential vulnerabilities.

About the 2024 OSSRA

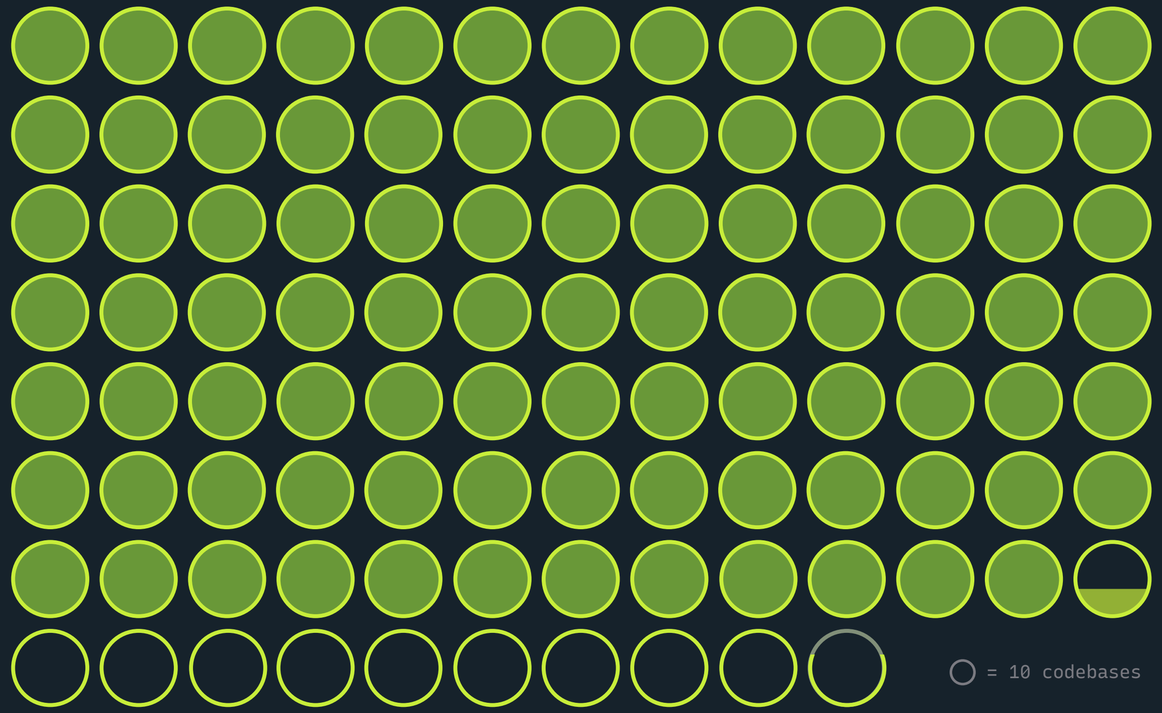
In this, its ninth edition, the 2024 OSSRA report delivers an in-depth look at the current state of open source security, compliance, licensing, and code quality risks in commercial software. The findings in this report are presented with the goal of helping security, legal, risk, and development teams better understand the open source security and license risk landscape.

This report uses data from the Synopsys Black Duck® Audit Services team's analysis of anonymized findings from 1,067 commercial codebases across 17 industries during 2023. The Audit Services team has helped security, development, and legal teams around the world strengthen their security and license compliance programs for over 20 years. The team audits thousands of codebases for our customers each year, with the primary aim of identifying software risks during merger and acquisition (M&A) transactions.

The audits also provide a comprehensive, accurate Software Bill of Materials 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. Sourced and curated by the Synopsys Cybersecurity Research Center (CyRC), the KnowledgeBase includes data on more than 7.8 million open source components from over 31,000 forges and repositories.

The OSSRA report highlights the prevalence of open source in software as well as the potential dangers of not properly managing it. Open source is the foundation for all applications that businesses and consumers rely on today. Identifying, tracking, and managing open source effectively is critical to a successful software security program—as well as a key element to strengthening the security of the software supply chain.

Overview



1,067 codebases **scanned** in 2023

936 codebases underwent **risk assessments**

96%
of the total codebases **contained** open source

53%
of the total codebases **contained license conflicts**

77%
of all code in the total codebases **originated** from open source

31%
of the total codebases contained open source with **no license** or a **custom license**

10 years

2.8 years

24 months

12 months

14% of the codebases assessed for risk contained vulnerabilities **older than 10 years**

2.8 years was the **mean age of vulnerabilities** in the codebases assessed for risk

49% of the codebases assessed for risk had components that had **no development activity** in the **past 24 months**

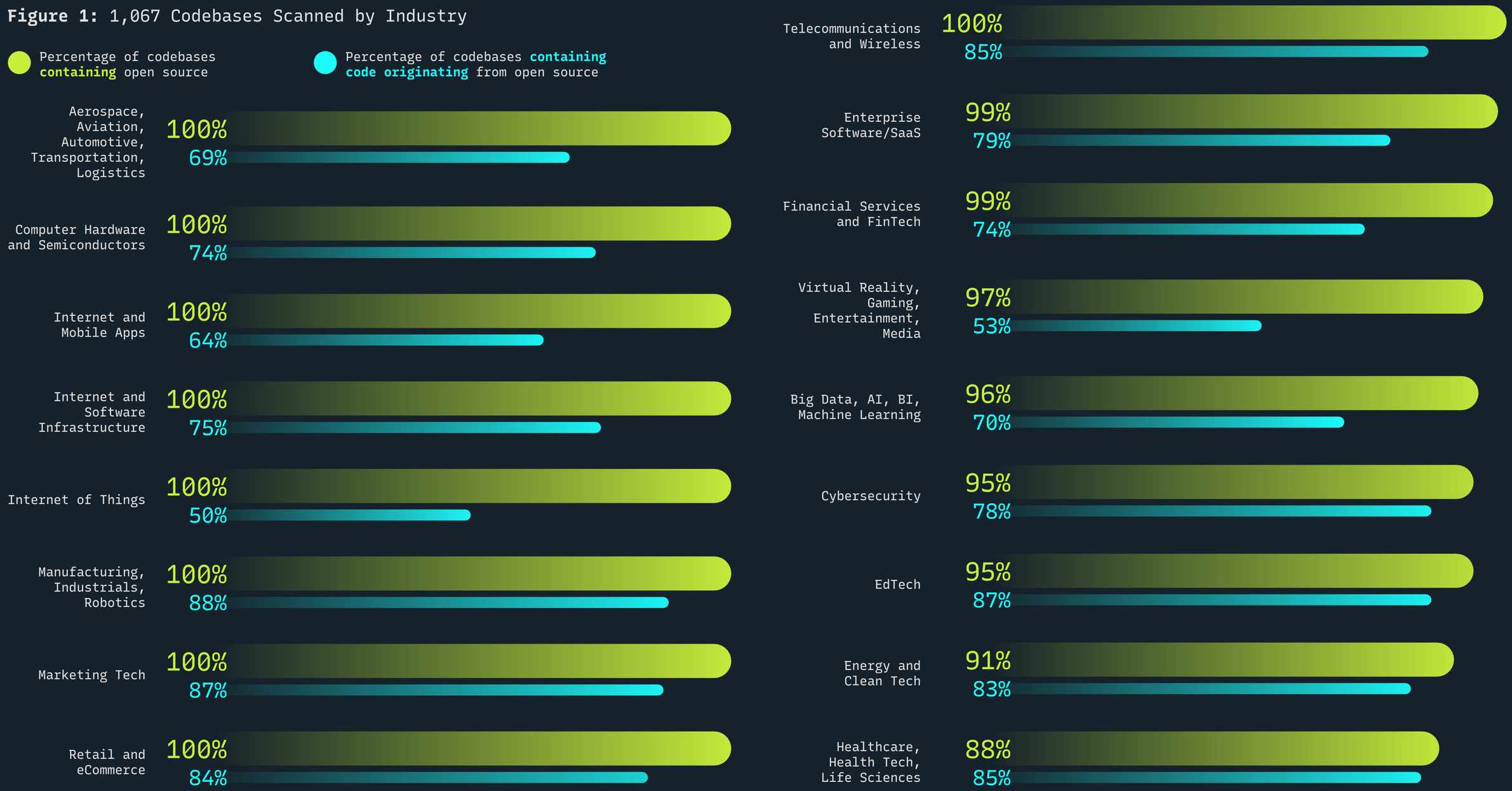
1% of the codebases assessed for risk had components that were at least **12 months behind** on code maintainer **updates/patches**

84%
of codebases assessed for risk contained **vulnerabilities**

74%
of codebases assessed for risk contained **high-risk vulnerabilities**

91%
of the codebases assessed for risk contained components that were **10 versions** or more **behind the most current version** of the component

Figure 1: 1,067 Codebases Scanned by Industry



Open Source Vulnerabilities and Security

A Note on the Audits

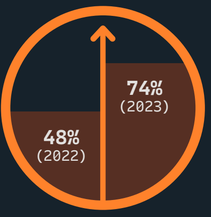
All Black Duck audits examine open source license compliance. Customers can opt out of the vulnerability/operational risk assessment portion of the audit at their discretion. During 2023, the Black Duck Audit Services team conducted 1,067 audits. Of those audits, 88% (936) also underwent a vulnerability/operational risk assessment. The data in the "Open Source Vulnerabilities and Security" and "Operational Factors Affecting Open Source Risk" sections of the 2024 OSSRA report is based on the 936 codebases that included risk assessments. The data in the "Open Source Licensing" section is based on all 1,067 codebases.

Of the 1,067 codebases analyzed by the Black Duck Audit Services team and used as the base data for this year's OSSRA report, 96% contained open source. Seventy-seven percent of all the source code and files scanned originated from open source code.

The average number of open source components in a given application this year was 526—a practical example of the importance if not absolute necessity for automated security testing. Manual testing, which might be feasible for a small number of components, becomes virtually impossible at scale and requires the use of an automated solution like software composition analysis (SCA). Unlike manual testing, automated security tests can be executed quickly and consistently, allowing developers to identify issues early in the development process without impacting delivery schedules or productivity.

Eighty-four percent of codebases that included a risk assessment contained at least one known open source vulnerability. Seventy-four percent of those codebases contained high-risk vulnerabilities, a significant increase from 2022, when only 48% of the codebases were found to contain high-risk vulnerabilities. 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.

84% of codebases contained at least **one open source vulnerability**



54% increase in codebases containing **high-risk vulnerabilities** in the past year

There's no single answer for the 54% increase (26 percentage points) in high-risk vulnerabilities between 2022 and 2023. One possible explanation is the economic downturn and consequent layoffs, which reduced the number of resources available to locate and patch vulnerabilities. Additionally, 91% of the codebases were found to contain components 10 versions or more behind the most current available version of the component. The simple conclusion is that the vast majority of open source consumers simply aren't updating the components they use.

Forty-nine percent of the codebases contained components that had no development activity within the past 24 months, and 1% had components at least 12 months behind on code maintainer updates/patches.

Broadly, the term "maintainers" refers to those contributors who lead an open source project. They may be the final decision-makers on which portions of source code go into a build or release, they may do all the code review and host the code under their names for smaller projects, and they may make the ultimate decision over the direction of a project. Their day-to-day work may vary but it can consist of reviewing pull requests and other contributions, releasing new versions of software, triaging and handling security fixes, and community management and moderation.

Most maintainers are diligent about keeping the open source projects they're involved with up-to-date. In fact, many companies specifically hire people to maintain open source projects the organization's software relies upon. The same diligence needs to be encouraged in open source consumers. Consumers of open source need to stay aware of the versions they have in use, establish a regular cadence for updates, and practice software hygiene when it comes to open source—only downloading from projects with a healthy ecosystem of maintainers and contributors.

Common Weakness Enumerations (CWEs) and Common Vulnerabilities and Exposures (CVEs) are popular lists used to identify and classify security weaknesses and vulnerabilities in software. Black Duck Security Advisories (BDSAs) are Synopsys proprietary reports designed to provide customers with timelier and more detailed information at a higher level than National Vulnerability Database (NVD) CVE notices. BDSAs deliver actionable advice and details about vulnerabilities affecting items in customers' SBOMs to help ensure that they have a complete picture of the risk that an open source vulnerability may pose.

Eight of the top 10 vulnerabilities map to one pillar CWE, **CWE-707**, which addresses security requirements that are not being met, and can lead to exploits such as cross-site scripting and SQL injection.

As shown in Figure 2, CWE-707 is the pillar for CWEs 20, 79, 80, 97, and 937. CWE-707 concerns security requirements that are not being met before data is read from an upstream component or sent to a downstream component. Failing to properly neutralize input can lead to exploits such as cross-site scripting (XSS) and SQL injection. A common and dangerous exploit, XSS is associated with the majority of the top 10 vulnerabilities highlighted in this report.

Cross-site scripting occurs when an attacker takes advantage of a flaw in a website by sending malicious, malformed code, usually written in JavaScript. Since this input isn't properly neutralized or escaped, the exploit can manipulate the otherwise trustworthy host into performing malicious tasks. The end target of most XSS attacks isn't the host itself, though; it's other users of the web application. Once the malicious script is injected, it can be used to steal sensitive information, like session cookies.

Not only is XSS in our top 10 vulnerabilities list, it is also one of the vulnerabilities in the OWASP Top 10 list—a report that lists the 10 most critical web application security risks. In its list, OWASP refers to XSS as A03:2021 – Injection. The reason for the prevalence of XSS vulnerabilities can be credited to increased reliance on web-based applications as a way for organizations to interact with customers, and for users to interact with one another. For example, consider how many eCommerce companies, banks, internet service providers, insurance companies, and others offer web experiences to engage with their customers and partners.

Again, the data clearly shows that development teams need to improve at keeping open source components up-to-date, especially when it comes to popular open source components such as jQuery. The consequences of using older, more vulnerable versions of open source can be grim. For example, #2 of the top 10 vulnerabilities found in the audits is BDSA-2020-0686 (CVE-2020-11022), an XSS vulnerability in jQuery versions 1.2 to 3.5.0. This vulnerability enables passing HTML from untrusted sources—even after sanitizing it—to one of jQuery's DOM manipulation methods, and may execute untrusted code.

This issue was patched in jQuery 3.5.0, but as our data illustrates, a third of the codebases scanned for security risks were found to be using a jQuery version still vulnerable to it. Malicious data could be used to breach a system, or sensitive data—passwords, credit information—could be exposed. As noted earlier, cross-site scripting is one of the most common vulnerabilities in applications, classically using a code injection attack against the various language interpreters in the browser, such as HTML or JavaScript.

Mitigating Risk: Tips for Using jQuery and Other Popular Open Source Securely

All 10 of the top 10 open source components identified in the audits were written in JavaScript. The bulk of vulnerabilities found in the audits were also associated with JavaScript libraries, notably vulnerabilities in outdated versions of the jQuery JavaScript library.

- Use the latest version of jQuery. As shown in this report, older versions of jQuery often contain security vulnerabilities.
- Consider subscribing to a security advisory service—such as Black Duck Security Advisories—to get the latest vulnerability information. New security vulnerabilities are discovered in jQuery regularly.
- Use a secure coding framework to help you to identify and avoid potential security vulnerabilities in your code.
- Use automated security testing—including static analysis, software composition analysis, and dynamic analysis tools—to address code quality and security flaws throughout the software development life cycle.

jQuery is not inherently insecure. In fact, it is a well-maintained open source library with a large community of users, developers, and maintainers. But problems often accompany popularity. According to the audits, jQuery was the component most likely to have vulnerabilities, even though each of the jQuery vulnerabilities listed in this report have available patches. It is important for users of jQuery—and indeed users of all open source—to be aware of the potential security risks associated with older versions of software, and to take steps to mitigate those risks.

Taking Action to Prevent Vulnerabilities from Entering Your Software Supply Chain

- Create and maintain a Software Bill of Materials (SBOM). In the fight against software supply chain attacks, having an accurate, up-to-date SBOM that inventories open source components is critical to assessing exposure, and ensuring that your code remains high quality, compliant, and secure. A comprehensive SBOM lists all the open source components in your applications as well as those components' licenses, versions, and patch status—a strong defense against supply chain attacks, including those using malicious packages.
- Stay informed. Ensure that you have the means to be informed of newly identified malicious packages, malware, and disclosed open source vulnerabilities. Look for newsfeeds or regularly issued advisories that provide actionable advice and details about issues affecting open source components in your SBOM.
- Perform code reviews. Examine the code of downloaded software before including it in your project. Check for any known vulnerabilities. For additional insight, consider including a static analysis of source code to check for unknown security weaknesses.
- Be proactive. Just because a component isn't vulnerable today doesn't mean that it won't be tomorrow. Intentionally malicious packages may never even be discovered as "vulnerable." Pay attention to component health and provenance before implementation to avoid future security issues.
- Use an automated software composition analysis (SCA) tool. An SCA tool automates the process of identification, management, and mitigation of software security issues and allows developers to focus their energies on writing code. Such tools can evaluate open source and third-party code.

Eight of the Top 10 Vulnerabilities Can Be Traced Back to One CWE

A “pillar weakness” is defined by the CWE project as the highest-level weakness representing a base for all class/variant weaknesses related to it.

Figure 2: Top 10 CVEs/BDSAs

Why Some BDSAs Don’t Have CVEs

Public sources, such as the National Vulnerability Database (NVD), are a good first step for information on publicly disclosed vulnerabilities in open source software. But there can be lags in the reporting of any given NVD CVE entry. Timeliness has always been a factor affecting the ability of the NVD to publicize security vulnerability data. In fact, there is often a significant time lag between the first disclosure of a vulnerability and its publication in the NVD, with some research reporting a month on average between the initial announcement and NVD publication.

Another problem with the NVD is that it often provides incomplete vulnerability data. Many CVEs published in the NVD do not include vulnerable version ranges and are often too short to be useful. This is commonly because no resources were available to research the entry.

Clearly, it’s unwise to rely solely on the NVD for vulnerability information. Many commercial SCA solutions can provide richer vulnerability data than the NVD alone, in addition to finding issues more accurately and helping you fix them more quickly when warranted. For example, if you are using Black Duck, an SCA solution from Synopsys, you can take advantage of BDSAs, which are advisories of open source vulnerabilities identified by Synopsys security research teams. Often these advisories provide earlier notification of vulnerabilities affecting your codebase—it can be days or weeks before they appear in the NVD. BDSAs also provide more complete vulnerability data, delivering security insight, technical details, and upgrade/patch guidance beyond anything else commercially available today. For example, there are three BDSAs without associated CVEs from the NVD in our top 10 vulnerability listing for 2023.

BDSA-2021-3651

According to the BDSA, some versions of jQuery contain commented references to a hijacked domain. It is a security issue of sorts, and the safest thing to do was remove the links to the hijacked domain, which is what jQuery did in version 3.6.1, because users without awareness of a domain’s status could still be exposed to unspecified attacks if they attempted to follow the links to the hijacked site. Although the sites cannot be reached by running the code, there is value in flagging the issue, as the malicious site could accidentally be clicked by a developer or anyone with access.

The Synopsys CyRC team advises that this BDSA is tagged as “informational,” a tag used when the “fix” provided by the vendor takes the form of a warning in the code or the product documentation, or when the vendor has rejected the CVE or disputed the findings, and deems the reported vulnerability as expected/designed behavior in the component.

Category

CWE-546: Suspicious Comment

CVSS v3.1 score

5.10

BDSA-2014-0063

This is an older vulnerability, first raised as an issue in January 2014 and relating to a potential XSS vulnerability in jQuery caused by a lack of user-supplied input validation. This could allow an attacker to inject arbitrary web scripts and steal a victim’s session cookies.

According to the BDSA, a function parses an HTML string into an array of DOM nodes. Any scripts included in event attributes passed to this function are immediately executed. This could leave a caller of the function vulnerable to XSS attacks if they do not properly sanitize untrusted input before it is passed to the function. An attacker could exploit this by crafting malicious HTML to be supplied to a victim. If processed, any web scripts included will be executed on their system.

This vulnerability was mitigated in jQuery 3.0.0-rc1. However, the mitigation does not sanitize malicious input and will still allow scripts to be executed. The default behavior of the parser is changed such that if the context is unspecified or given as null/undefined, a new document is created. This delays execution of parsed HTML until it is injected into the document, allowing the opportunity for tools to traverse the created DOM and remove unsafe content after the function call.

Category

CWE: 79: Improper Neutralization of Input During Web Page Generation (“Cross-Site Scripting”)

CAPEC-588: DOM-Based XXS

CVSS v3.1 score

8.60

BDSA-2015-0567

Another older vulnerability, this time with jQuery vulnerable to arbitrary code execution—versions of jQuery that use an unpatched UglifyJS parser are vulnerable to arbitrary code execution through crafted JavaScript files. Ultimately, this can allow attackers to run rogue code.

The vulnerability was fixed in 1.12.0 and in 2.2.0.

Category

CWE Category A9: Using Components with Known Vulnerabilities

CAPEC-251: Local Code Inclusion (The Common Attack Pattern Enumeration and Classification [CAPEC] effort provides a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy)

CVSS v3.1 score

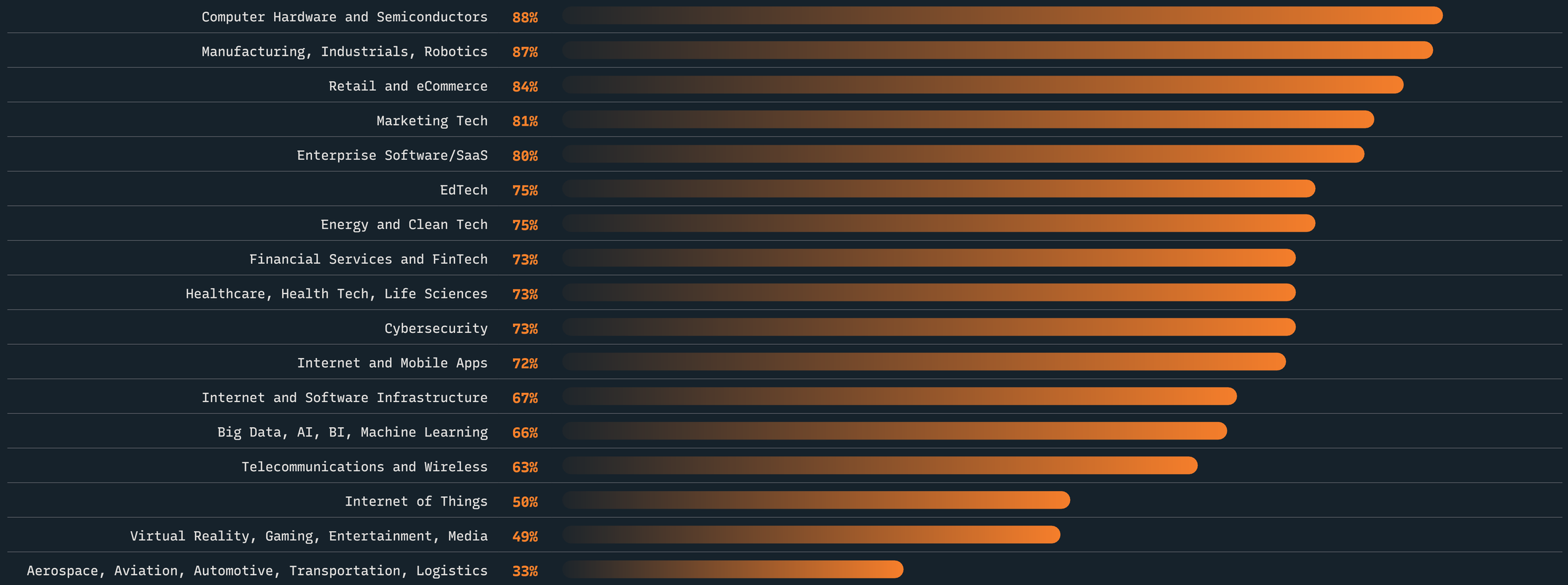
7.9

Vulnerabilities by Industry

Eighty-eight percent of the codebases associated with the Computer Hardware and Semiconductors sector contained vulnerabilities classified as high-risk (those with a severity score of 7 or higher), closely followed by Manufacturing, Industrials, Robotics; and Retail and eCommerce, with 87% and 84% respectively.

Similar findings played out across each sector. Even the lowest percentage—for the Aerospace, Aviation, Automotive, Transportation, Logistics sector—is still unsettling, with a third of that industry's codebases containing high-risk vulnerabilities. As shown in Figure 1, open source was in every industry codebase we examined; it made up most of the codebases across each industry sector. Figure 3 demonstrates that these codebases also contain high numbers of known open source vulnerabilities that organizations have failed to patch, leaving them vulnerable to exploit.

Figure 3: Percentage of Codebases Containing High-Risk Vulnerabilities

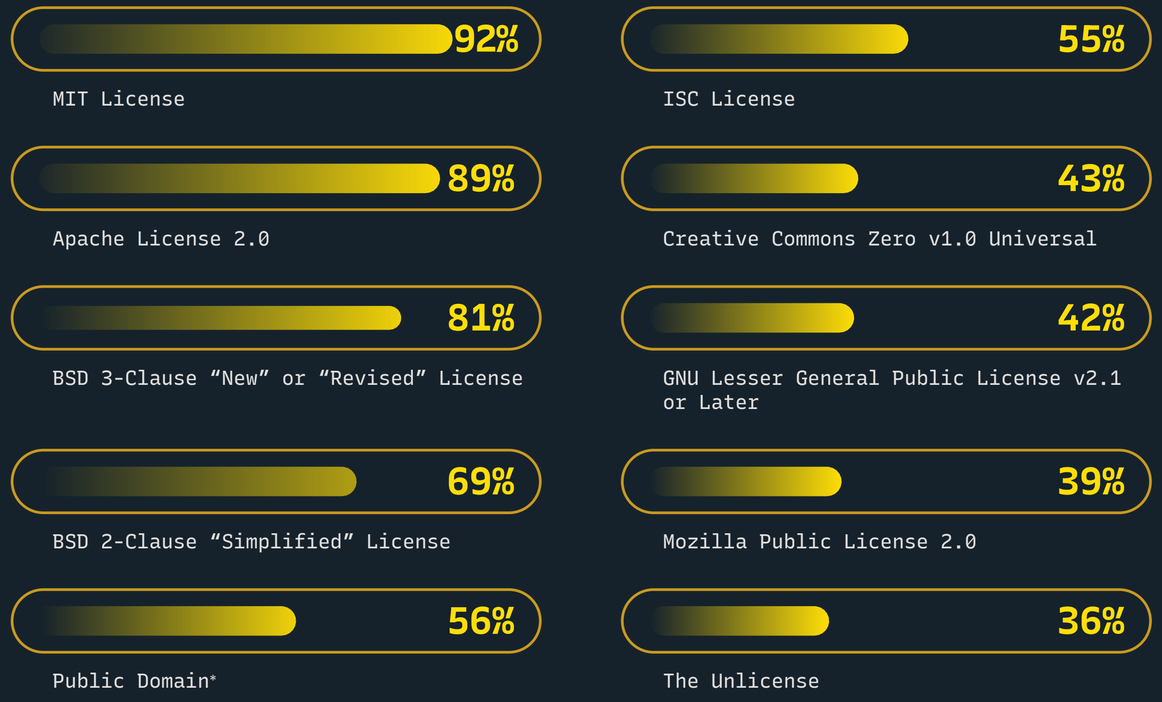


Open Source Licensing

Effective software supply chain management requires licensing as well as security compliance. You're using open source components and libraries to build software and know those components are governed by open source licenses, but do you know those licenses' details? Even one noncompliant license in your software can result in legal issues, loss of lucrative intellectual property, time-consuming remediation efforts, and delays in getting your product to market.

The Black Duck Audit Services team found that over half—53% —of the 2023 audited codebases contained open source with license conflicts.

Figure 4: Percentage of Top 10 Licenses Found in Codebases



*Component includes a statement that it is in the public domain but does not use a specific public domain license such as the Unlicense or Creative Commons

The MIT license was found in 92% of the open source audited by Black Duck Audit Services in 2023. As a permissive license that permits reuse within proprietary software, the MIT license has high compatibility and low risk with other software licenses. You can be fairly certain that if you include third-party components in your software, you're likely to find popular, permissive licenses such as MIT, Apache, BSD, ISC, and Unlicense.

It should be noted that terms such as "low-risk" are only a guideline and should not be used to make decisions about using the open source software governed by each license. For example, although Apache 2 software—generally considered to use a low-risk license—can be included in projects licensed under GNU General Public License 3.0 (GPLv3), GPLv3 software cannot be included in Apache projects. The licenses are incompatible in that situation as a result of Apache Software Foundation's licensing philosophy and the GPLv3 authors' interpretation of copyright law. The safest strategy is for developers to consult their corporate policies and legal teams for specific guidance regarding license compliance.

Creative Commons licenses were found in the 2023 audits to be the most prevalent cause of license conflict. Creative Commons ShareAlike 3.0 (CC-SA 3.0) alone was found to be the cause of 17% of the identified license conflicts.

"Snippets"—lines of code that have been copied and pasted into source code—are quite frequently found by Black Duck audits. They are often taken from the popular blog site Stack Overflow, which automatically licenses all publicly accessible user contributions under Creative Commons ShareAlike. Unfortunately, the blanket license also covers code snippets posted on the site. We say "unfortunately," because these licenses are not intended for software, with Creative Commons explicit about this in its FAQ: "We recommend against using Creative Commons licenses for software." The CC-SA license can be read in some situations as having a similar "viral" effect (that is, any work derived from a copyleft-licensed work must also be licensed under the same copyleft terms) as the GNU Public License and can become a concern from a legal standpoint.

Understanding License Risk

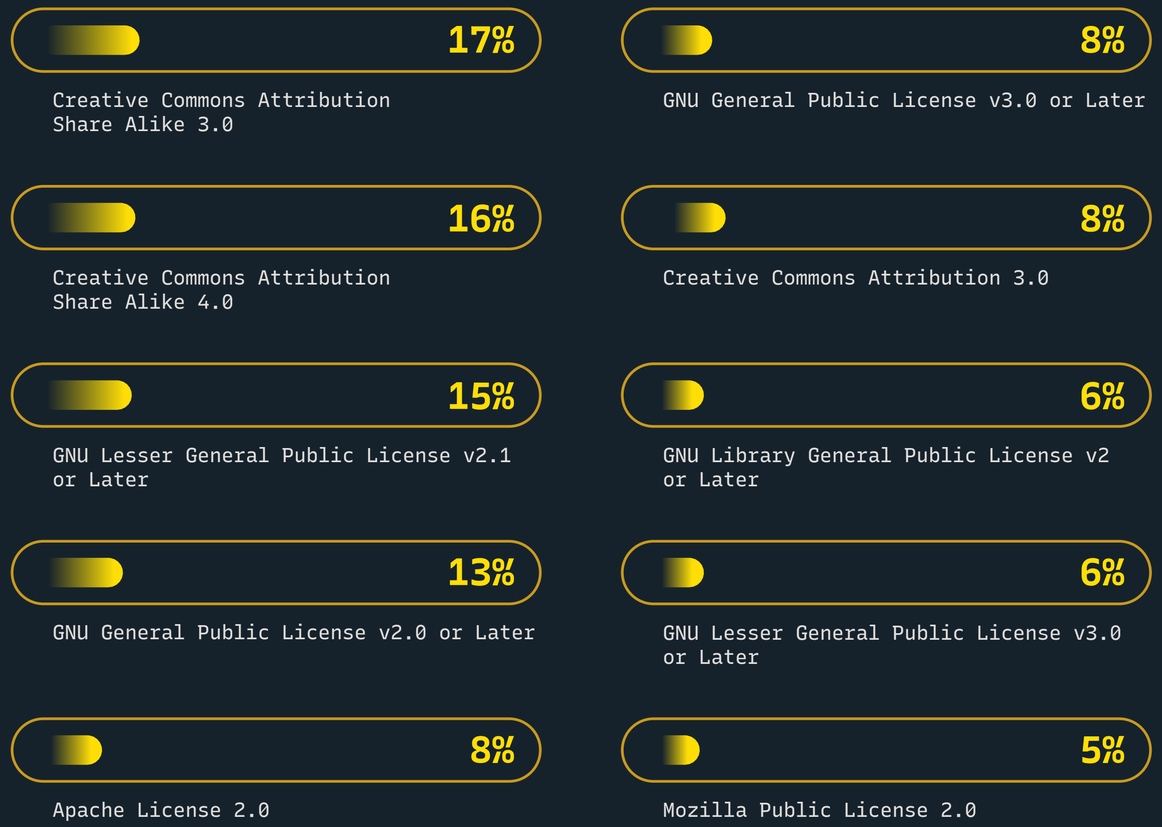
In the U.S. and many other jurisdictions, creative work (including software) is protected by exclusive copyright by default. No one can legally use, copy, distribute, or modify that software without explicit permission from the creator/author in the form of a license that grants the right to do so.

Even the friendliest open source licenses include obligations that 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.

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.

Thirty-one percent of the 2023 audited codebases were using code with either no discernible license or a customized license, virtually the same as last year's findings. It's not uncommon for open source audits to find code snippets from sites that, unlike Stack Overflow, have no discernable terms of service or mention of software terms. Another common cause of open source code without license terms is a developer using a code snippet but failing to bring the snippet's associated licenses along with it. Yet another issue when it comes to code with no associated license is the growing use of AI-assisted coding tools (see next section).

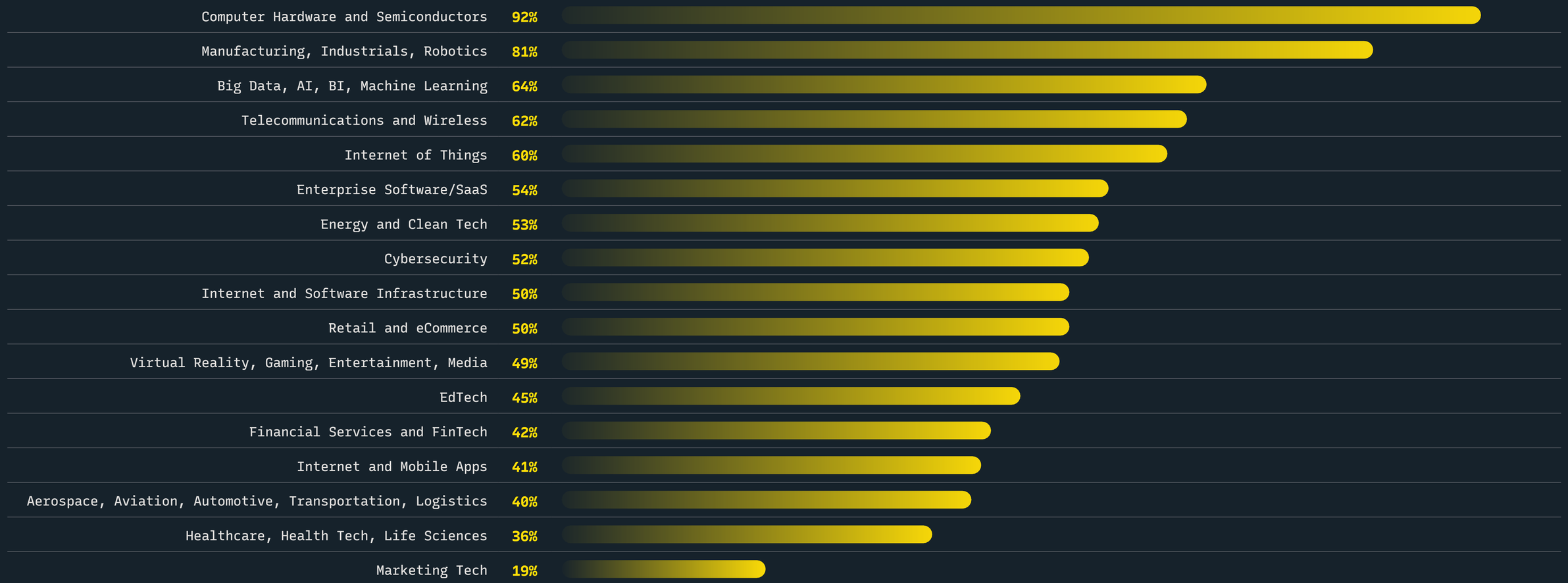
Figure 5: Percentage of Top 10 Licenses with Conflicts



A few probable reasons why several industry sectors have such a high percentage of open source license risk (see Figure 6) include

- Many of the industries with a high number of conflicts tend to license and distribute their software as on-premises products. Many restrictive licenses apply specifically to software that is distributed in this manner. Other industries with lower numbers may do more subscription-based or SaaS type deployments, which are not traditionally considered "distributing" and are not subject to the same license terms.
- Semiconductor and hardware companies rely heavily on software and firmware, much of which incorporates open source code (see Figure 1). Complex system-on-a-chip designs can have millions of lines of code from diverse sources. Keeping track of licenses and obligations at that scale can be challenging.
- Open source is extremely prevalent in lower-level system software, firmware, drivers, and so on, which are integral to hardware products. Much of this is under GPL-type "copyleft" licenses that have strong sharing requirements if distributed.
- Software supply chain sharing of firmware, drivers, and tools among companies and between hardware designers and manufacturers leads to open source spreading without tracking of origin or licenses through an SBOM inventory.

Figure 6: Percentage of Codebases Containing License Conflicts



Protecting Against Security and IP Compliance Risk Introduced by AI Coding Tools

Arising with the use of AI-powered coding suggestion tools are questions around ownership, copyright, and licensing of the generated code. For example, a class-action lawsuit filed against GitHub, Microsoft, and OpenAI claims that GitHub Copilot—a cloud-based AI tool that offers developers autocomplete-style suggestions as they code—violates both copyright law and software licensing requirements. The lawsuit further claims that the code suggested by Copilot uses licensed materials without attribution, copyright notice, or adherence to the original licensing terms.

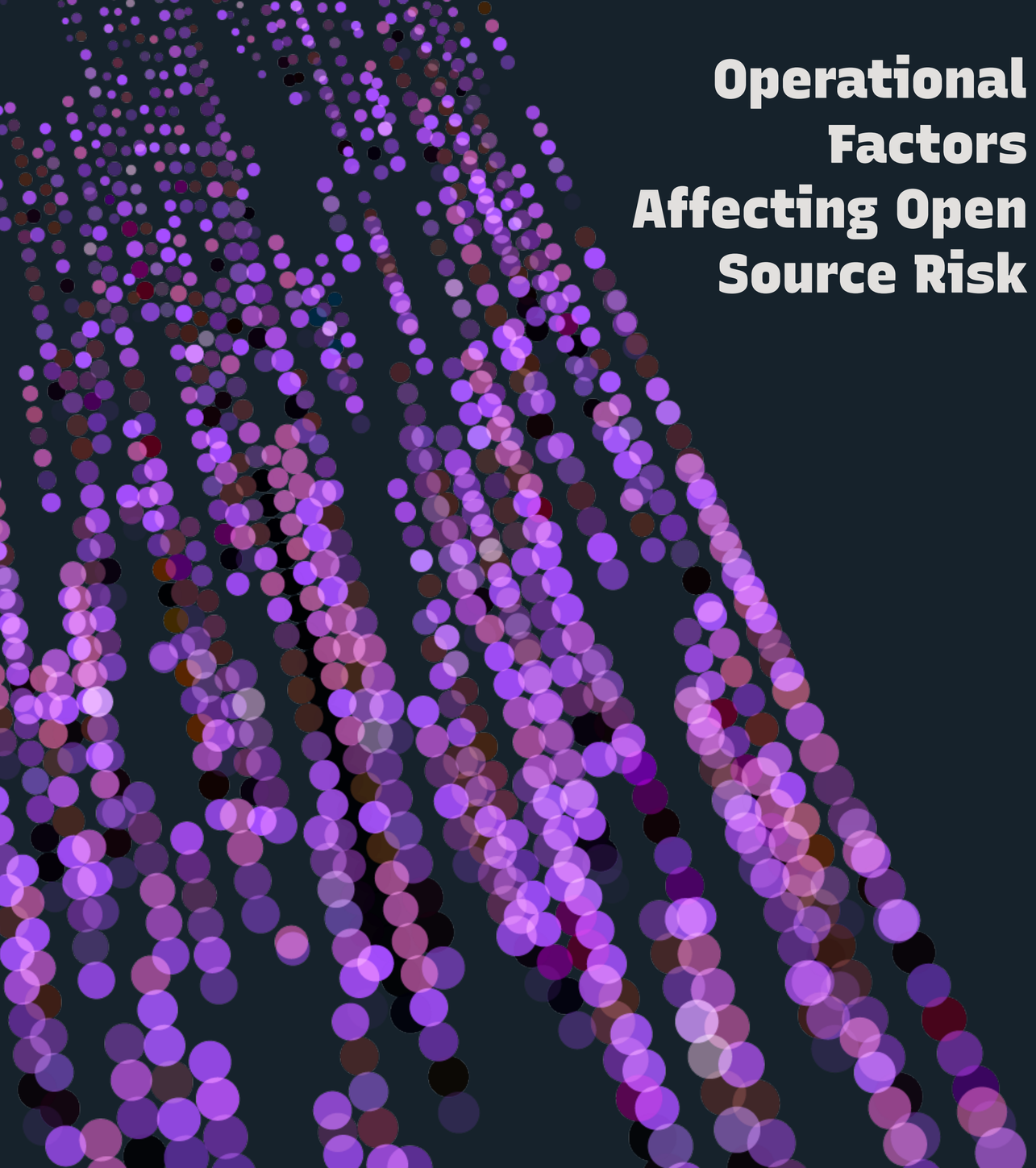
The Copilot case highlights the legal complexities surrounding the use of AI-generated code. For software developers, refraining from using AI-assisted coding tools until the issue is resolved by legal or government decision is obviously the safest way to avoid an action for license or copyright violations.

Developers who do want to use AI-assisted tools should exercise caution and avoid unnecessary risks. At a minimum, they should have their organization ask their AI tool vendors whether its recommendations include source code subject to open source licenses, and if so, whether that code can be highlighted or excluded from recommendation altogether.

Another solution is to use one of the available code scanners, such as Synopsys Black Duck, which uses snippet analysis to scan source code and match individual lines of code back to any open source project they may originate from. Development teams can identify the applicable license and subsequent terms for the open source code introduced by AI code assistants.

Best Practices for Open Source License Management in Software Supply Chain Governance

- Conduct a thorough inventory of all third-party software components in your software, including both open source and commercial software.
- Be aware that AI-assisted coding tools may produce code with the potential for license violations and intellectual property infringement.
- Evaluate the license terms and conditions of each component and assess whether they are compatible with the intended use of the product.
- Check for compatibility between the licenses of different components, as some licenses may not be compatible with each other.
- Use automated scanning tools to identify and track license obligations and restrictions for each component.
- Implement a process for ensuring ongoing license compliance, including regular license scans and periodic reviews of license compliance procedures.
- Establish a review process and workflow for new or unfamiliar licenses.
- Ensure effective communication between legal, technical, and business stakeholders to properly prioritize and execute license clearance (that is, the process by which a company decides whether a particular component's license is acceptable for use in its products) efforts.
- Document all license clearance activities, including license assessments and compliance procedures, to ensure a record of compliance efforts and facilitate future audits.

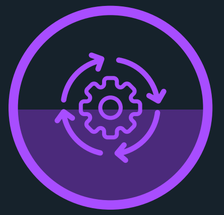


Operational Factors Affecting Open Source Risk

Ideally, open source consumers use only components supported by robust communities. Linux, for example, is improved every day by thousands of developers from hundreds of organizations. However, of the 936 codebases examined by the Black Duck Audit Services team that included risk assessments, 49% contained open source that had no new development in the last two years. If a project is no longer being maintained—especially in the case of smaller projects—there have been no feature upgrades, no code improvements, and no discovered security problems fixed.

It's not an uncommon issue with open source projects. According to some reports, nearly 20% of Java and JavaScript open source projects that were being maintained in 2022 are no longer being maintained in 2023, opening those projects to vulnerabilities and exploits. Open source is largely the product of volunteer contributors and maintainers. While some organizations such as Microsoft, RedHat, and Google have incentive programs in place to motivate open source project maintenance and participation, the vast majority of companies do not. When maintainers have stopped maintaining a project, one consequence is elevated security risk.

88% of codebases analyzed in 2023 underwent **risk assessments**



49% of codebases that had risk assessments contained open source that had **no new development in the last two years**

Open Source Consumers Need to Improve Maintenance Practices

Of the 936 codebases examined by the Black Duck Audit Services team in 2023 that included risk assessments, 91% contained components 10 versions or more behind the most current version of the component.

There can be valid justifications for open source consumers not keeping an open source component up-to-date. If they're aware of the situation—which, unfortunately, many are not—a development team might determine that the risk of unintended consequences outweighs whatever benefit would come from applying the newer version. For example, embedded software may be at minimal risk from exploits 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. The Synopsys "2023 Global State of DevSecOps" report found that in a survey of 1,000 IT security professionals, 28% said their organizations take as much as three weeks to patch critical security risks/vulnerabilities in deployed applications, with another 20% saying it could take up to a month. And these figures pertain to *all* vulnerabilities—proprietary, commercial, and third-party software as well as open source.

As noted in nearly a decade of OSSRA reports, open source is different from commercial software—not worse, not better, but different—and it requires different techniques to manage. For example, patches are handled quite differently for commercial and open source software. The purchase of commercial software usually requires some review as part of a vendor management program. On the other hand, open source may simply have been downloaded at the developer's discretion. There may be some organizational guardrails—*use only code with permissive licenses,* for example—but in many organizations, not even this guidance exists.

All organizations that use commercial software are familiar with patches and updates being "pushed" to their software, or at a minimum, receiving a notice from the vendor that an update—often an urgent update—is available for download. That's seldom the case with open source, where the open source consumer is expected to stay aware of a component's status and download new versions as they become available.

There's only one viable solution to stay aware of the open source versions you use. You need an accurate, comprehensive inventory of open source, as well as automated processes to monitor vulnerabilities, upgrades, and the overall health of the open source in your software.

Findings and Recommendations

Whether a single developer or a large company, everyone has a responsibility to maintain software supply chain security practices in order to mitigate risks. As the number of software supply chain attacks grows, effectively managing open source usage, components, and dependencies becomes even more critical to managing risk. Organizations that include open source in their products—which, as this report demonstrates, is literally all organizations—should proactively manage open source risks as a part of their secure software development practices.

"Securing the Software Supply Chain: Recommended Practices for Managing Open Source Software and Software Bill of Materials," published by the United States Cybersecurity and Infrastructure Security Agency in late 2023, provides detailed guidelines for the use of open source in the software supply chain, including

- **Integrate open source into the secure build process of the product using the same policies and process as with in-house developed components.**
  The security team commonly defines security policies, processes, and tools concerning open source. Developers will ideally select a component with the needed features from a prevetted internal repository, which has had an initial vulnerability assessment through an SCA security analysis tool, and then run further scans during the development and/or build stage to catch issues as early as possible.

- **Track updates to open source and monitoring for issues and vulnerabilities.**
  When a vulnerability is identified, affected software should be assessed to identify the prevalence of the component and its use within the product. The component should be updated, or, if it's no longer being maintained, strong consideration should be given to finding an alternative solution.

- **Use an SBOM.**
  Understanding which components are included in software is essential to accurate and complete management of code. An SBOM is a formal record containing the details and supply chain relationships of the components in the software. SBOMs increase software transparency and document component provenance. In the context of vulnerability management, SBOMs support the identification and remediation of vulnerabilities. From a code quality standpoint, the existence of an SBOM may be indicative of a supplier's use of secure software development practices across the software development life cycle.

Executive Order (EO) 14028, "Improving the Nation's Cybersecurity," states that organizations may be requested to provide an SBOM directly to the purchaser or publish it on a public website, and that both government and nongovernment parties may be required to review the SBOM to ensure that software products comply with the minimum elements for an SBOM. The EO also directed the Department of Commerce and the National Telecommunications and Information Administration (NTIA) to publish "The Minimum Elements for a Software Bill of Materials (SBOM)," which outlined the activities and data required for an SBOM as well as example formats that fulfill SBOM requirements. SPDX and CycloneDX were identified as the two most widely used machine-readable SBOM formats. In mid 2023, the Office of Management and Budget (OMB) published Memorandum OMB-23-16 updating an earlier OMB memo that allows federal agencies to require

- SBOMs based on the criticality of software or other criteria, as determined by each agency
- SBOMs in one of the formats defined by the NTIA

Creating a Secure Software Development Framework

Software producers have a critical role to play in securing software supply chains for the benefit of their customers and users. The National Institute of Standards and Technology's (NIST) Secure Software Development Framework (SSDF) presents a series of practices that serve as a baseline for securely developing software in a standardized way. Attestation to conformance with the NIST SSDF has been signaled by the U.S. government as a probable requirement for all software procured directly or indirectly by the U.S. government, and it's likely that software suppliers will need to self-attest to their adherence to the SSDF sometime in the near future.

Assessment tools such as the Synopsys SSDF Readiness Assessment can identify whether your organization's software development practices align with the practices and tasks of the SSDF so that you can attest with confidence that your software development processes conform to SSDF standards. Similarly, Synopsys SBOM Services builds upon Black Duck Audit Services processes to perform a full security audit of your software and generate an SBOM—a valuable service for organizations that do not yet have SBOM-generation capabilities and need a baseline SBOM.

Software producers with regulatory or contractual obligations may receive requests for an audited SBOM. Software consumers might want to audit the SBOM produced by one of their suppliers. In each of these scenarios, a trusted third party with a strong reputation in software audits is required. The Synopsys SBOM Audit and Validation service builds on those proven processes to audit the software and confirm whether the SBOM produced by the client accurately reflects the supply chain.

Knowing What's in Your Code

Recapping the report's findings

- 96% of the 1,000+ scanned codebases contained open source
- 77% of all their source code and files originated from open source
- 53% of the codebases had open source license conflicts
- 84% of the codebases assessed for security risks had vulnerabilities; 74% had high-risk vulnerabilities
- 91% of those codebases had components 10+ versions behind the latest version

Whether your organization develops or uses software, there's a near certainty it has open source components. Do you know exactly what those components are and whether they pose security or license risks? In the world of 2024, where 96% of code was found to contain open source, visibility into your code needs to be a priority. When 91% of risk-assessed codebases are using open source far behind the current version, consumers need to do better in keeping their code up-to-date, especially when it comes to popular open source components.

Keeping open source updated should be treated with the same priorities as the code your team develops. Create and maintain an SBOM that details what you have in your code, including details on version, license, and provenance. Set a regular cadence for upgrades, especially if you're using open source libraries from popular projects that have frequent maintainer activity.

Without comprehensive visibility into your code and keeping proactive software hygiene practices, you're exposing your software to potential exploits from open source vulnerabilities and IP compliance questions. Begin by using automated SCA tools to find security, code quality, and licensing issues early in the SDLC—because you need to know without question what's in your code.

Without comprehensive visibility into your code and keeping proactive software hygiene practices, you're exposing your software to potential exploits from open source vulnerabilities and IP compliance questions.

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.

For more information about the Synopsys Software Integrity Group, visit us online at www.synopsys.com/software.

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.

CWE
Common Weakness Enumeration (CWE) is a community-developed list of software and hardware weakness types assembled in three tiers. CWE has over 600 categories, including classes for buffer overflows, path/directory tree traversal errors, race conditions, cross-site scripting, hard-coded passwords, and insecure random numbers.

CVE
Common Vulnerabilities and Exposures (CVE) is a list of publicly disclosed information security flaws.

Black Duck Security Advisory (BDSA)
Detailed, timely, consistent information on open source vulnerabilities, BDSAs provide Synopsys customers with early and supplemental notification of open source vulnerabilities and upgrade/patch guidance. BDSAs deliver same-day vulnerability 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.

Snippet
Snippets are small, reusable pieces of code that developers cut and paste into their own code. Although software may include only a snippet of open source, users of the software must still comply with any license associated with that snippet.

Open source license
A set of terms and conditions stating end-user obligations, including how the component may be used and redistributed, when an open source component (or a snippet of a component's code) is used in software. 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 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 life cycle to identify the open source 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, "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.

Contributors

The "Open Source Security and Risk Analysis" report is produced through the collaborative effort of the Synopsys Software Integrity Group, including members of our Audit Services, Consulting, Research, Legal, and Marketing teams. Their work has made the OSSRA the leading report on open source code quality, security, and license compliance for the past decade.

Special thanks for their contributions to this year's report to Nancy Bernstein, Scott Handy, Siobhan Hunter, Matt Jacobs, Natalie Lightner, Merin McDonell, Mike McGuire, Phil Odence, Rie Sekine, Liz Samet, Jenny Stout, and Jack Taylor.

Rachel Bay has performed her incredible design magic during the nine years we've worked on the OSSRA together. It's been my privilege and honor to write it. — Fred Bals