SYNOPSYS®

2020 OPEN SOURCE SECURITY AND RISK ANALYSIS REPORT

Table of contents

INTRODUCTION

Welcome to the 5th edition of Synopsys' Open Source Security and Risk Analysis (OSSRA) report. The 2020 OSSRA includes insights and recommendations to help security, risk, legal, and development teams better understand the open source security and license risk landscape.

To help organizations develop secure, high-quality software, the Synopsys Cybersecurity Research Center (CyRC) publishes research that supports strong cyber security practices. Our annual OSSRA report provides an in-depth snapshot of the current state of open source security, compliance, and code quality risk in commercial software.

For over 16 years, security, development, and legal teams around the globe have relied on Black Duck® software composition analysis (SCA) solutions and open source audits to identify and track open source in code, mitigate security and license compliance risks, and automatically enforce open source policies using existing DevOps tools and processes.

Synopsys' Black Duck Audit Services team conducts open source audits on thousands of codebases for its customers each year, often supporting merger and acquisition transactions. In the context of software development, a codebase is the source code and libraries that underlie an application, service, or library. These audits are anonymized and used as the primary source of data for the OSSRA report. The data is cross-referenced with the Black Duck KnowledgeBase™ to identify potential license compliance and security risks as well as open source operational factors that may affect the overall codebase. The KnowledgeBase currently houses data on open source activity from over 20,000 sources worldwide, making it an authoritative source for open source projects and components

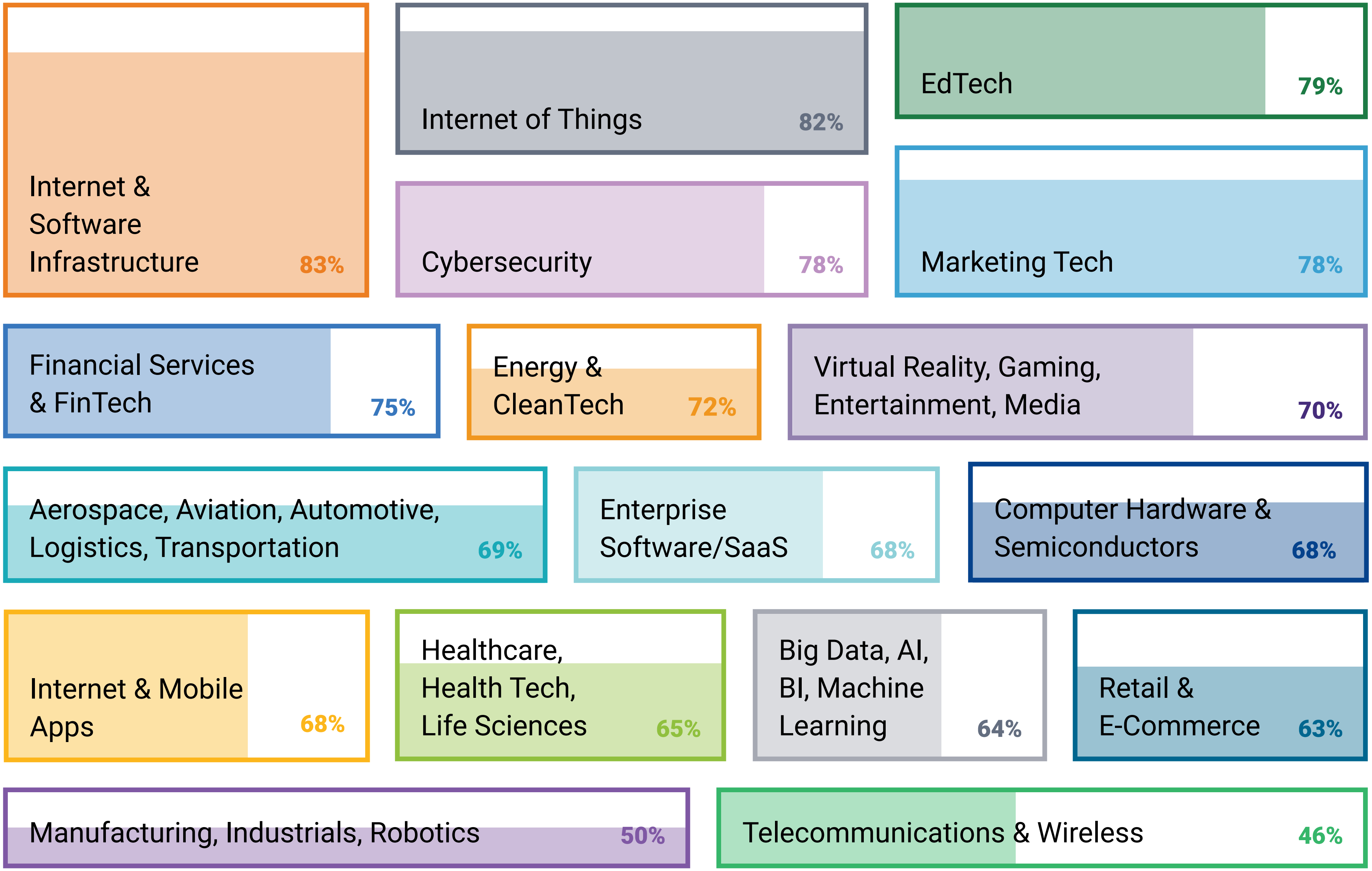
The 2019 audit data analysis was conducted by CyRC's Belfast and Boston teams. The Boston big data research team maintains the Black Duck KnowledgeBase, analyzing and refining open source activity from thousands of data sources to identify the most significant open source projects in use. Our Belfast team identifies the impact of open source vulnerabilities and their exploitability. As well as validating data used in the OSSRA, the Belfast team's work forms the basis of Black Duck Security Advisories (BDSAs), which offer enhanced vulnerability information that the team discovers, curates, analyzes, and publishes as a benefit for commercial Black Duck customers.

This year, the CyRC teams examined anonymized audit findings from over 1,250 commercial codebases in 17 industries, including Enterprise Software/SaaS; Healthcare, Health Tech, Life Sciences; Financial Services & FinTech; and Internet & Software Infrastructure (please see the next page for a full list).

As this report details, open source components and libraries are the foundation of literally every application in every industry. The need to identify, track, and manage open source has increased exponentially with the growth of its use in commercial software. License identification, processes to patch known vulnerabilities, and policies to address outdated and unsupported open source packages are all necessary for responsible open source use.

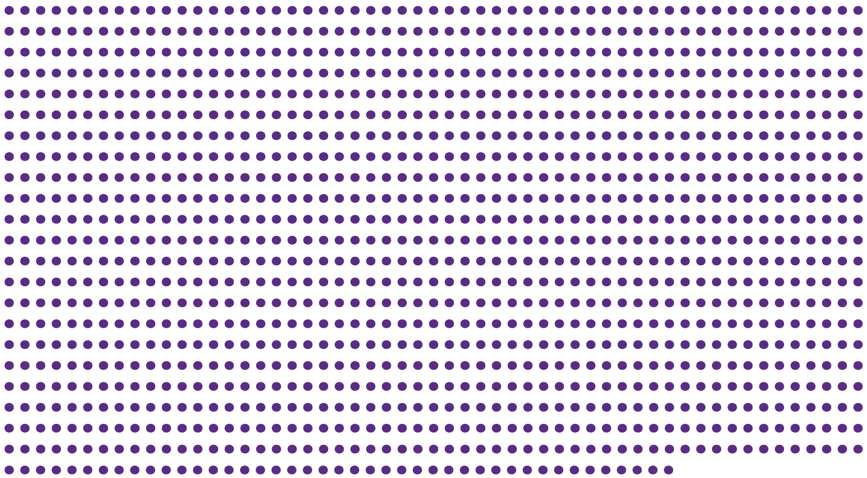
Industries represented in the 2020 OSSRA report

Percentage reflects amount of open source in codebases by industry



2020 OPEN SOURCE SECURITY AND RISK ANALYSIS

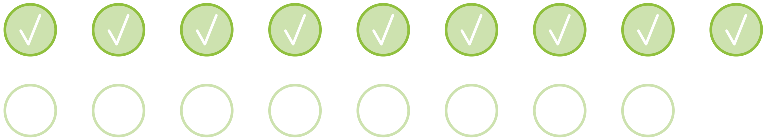
2020 AT A GLANCE



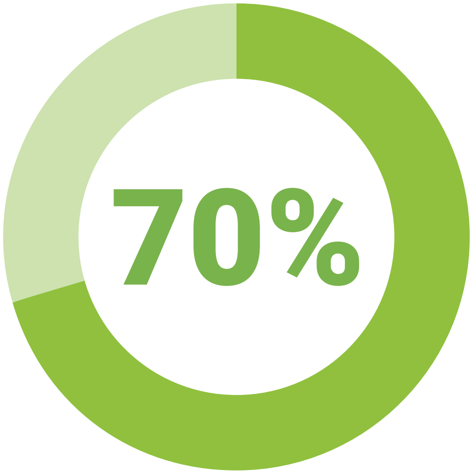
1,253 applications audited

Codebases & open source

99% of codebases audited in 2019 contained open source components.

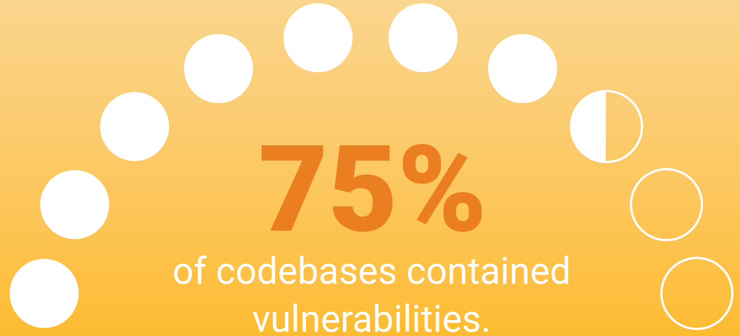


In 9 of 17 industries, 100% of the codebases contained open source.



Open source made up **70%** of the audited codebases.

Vulnerabilities



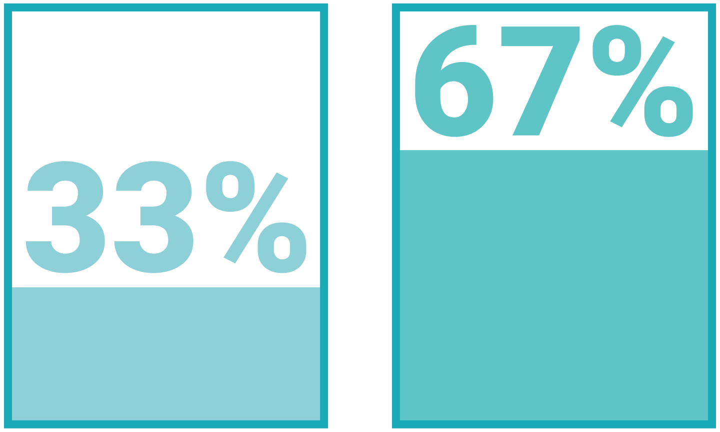
75%

of codebases contained vulnerabilities.

49%

of codebases contained high-risk vulnerabilities.

Licensing



33%

of codebases contained unlicensed software.

67%

of codebases had license conflicts.

Operational factors

82%

of codebases had components more than four years out of date.

88%

of the codebases had components with no development activity in the last two years.

The need for a software bill of materials

How can development teams know whether they're using high-quality open source components? Are those components' licenses permissive or restrictive? Are they one of the most commonly used licenses or variants? Is the component the most current version? Is that version the most stable? Is it the most secure? Is there a robust community actively maintaining the component?

The answers to those questions all begin with a software bill of materials, commonly referred to as a BOM.

In the November 2019 Gartner research paper "Technology Insight for Software Composition Analysis," analyst Dale Gardner notes that "comprehensive visibility into the open-source and commercial components and frameworks used in an application or service must be considered a mandatory requirement." Gardner goes on to recommend that organizations "continuously build a detailed software bill of materials (BOM) for each application providing full visibility into components."[1]

It may be possible to create and maintain a BOM manually, although some would argue that it is actually near impossible. In any case, doing so would require a significant investment of developer time. In turn, that would affect developer productivity, leading to higher development costs. "A BOM generated by an SCA tool provides more comprehensive information (specific versions, license, etc.)," Gardner writes, "and potentially a more advanced understanding of dependency mapping among various components and frameworks."[2]

Indeed, one of the outcomes of a Black Duck Audit is a comprehensive BOM of the open source components in any audited codebase, resulting in much of the snapshot data used in this report. Most SCA solutions also include the capability for development teams to generate a BOM themselves to identify, track, and manage the open source they use. A comprehensive BOM lists not only all open source components but also the versions used, the download locations for each project and all dependencies, the libraries the code calls to, and the libraries those dependencies link to.

The concept of a software BOM 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 auto manufacturer knows precisely which vehicles are affected and can begin the process of repair or replacement. Similarly, maintaining an accurate, up-to-date software BOM that includes an inventory of third-party and open source components is necessary for organizations to ensure their code is high quality, compliant, and secure. And, as in manufacturing, a BOM of open source components allows you to pinpoint vulnerable components quickly and prioritize remediation efforts appropriately.

Open source composition of codebases audited in 2019

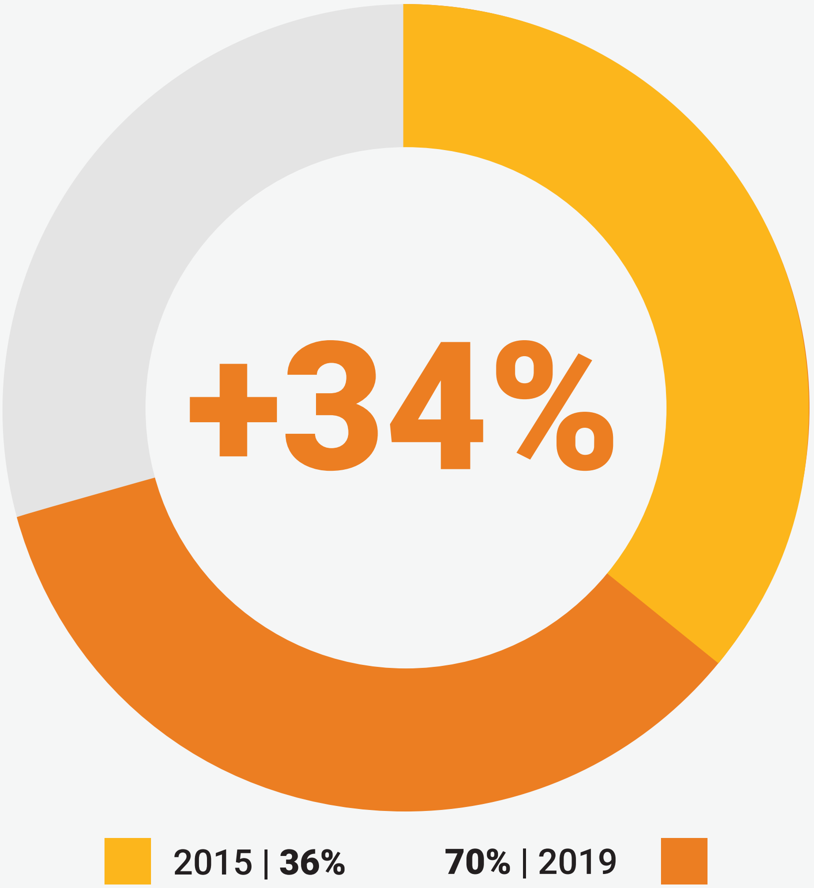
Black Duck Audits found open source in nearly 99% of codebases audited in 2019. In fact, 100% of the codebases from nine of 17 industries contained at least one open source component. Only 1.2% of codebases contained no open source components, and virtually all those comprised less than 1,000 files.

The war between open source and the idea that “we must use only proprietary code” is long over. Open source won by convincing the opposition of the benefits of joining the open source community. “Vulnerabilities in the Core,” a report published by the Linux Foundation and the Laboratory for Innovation Science at Harvard in early 2020, explains that contrary to the popular conception of unpaid programmers gamely coding open source in their basements, an analysis of GitHub data found that some of the most active open source developers contributed to projects under their Microsoft, Google, IBM, or Intel employee email addresses.[3]

As one of the report’s authors put it, “Open source was long seen as the domain of hobbyists and tinkerers. However, it has now become an integral component of the modern economy and is a fundamental building block of everyday technologies like smart phones, cars, the Internet of Things, and numerous pieces of critical infrastructure.”

Black Duck Audits identified an average of 445 open source components per codebase in 2019, a significant increase from 298 in 2018. While the percentage of codebases *containing* open source is nearing 100%, there has also been a dramatic, ongoing increase over the same period of the percentage of codebases *comprising* open source, which is replacing proprietary or commercial off-the-shelf software.

The first OSSRA reported that 36% of the audited code was open source. That percentage has now nearly doubled to 70%, up from 60% in 2018.



Our first OSSRA reported that 36% of the code we audited was open source. That percentage has nearly doubled in five years to 70%.

Even those percentages don't fully reflect the dominance of open source in commercial software. Open source allows developers to innovate faster because they don't need to reinvent core functionality. The Black Duck Audit Services team generally audits codebases from companies whose business is building software, versus enterprises for whom software supports their business. The primary value of software companies is in their proprietary code, and the ratio of open source to proprietary code in their codebases tends to be lower than the figures cited by analysts such as Forrester, who tend to look at enterprise IT groups for their reports. Analyst firms consistently report that over 90% of IT organizations use open source software in mission-critical workloads and that open source often composes up to 90% of new codebases.

What open source components are in use?

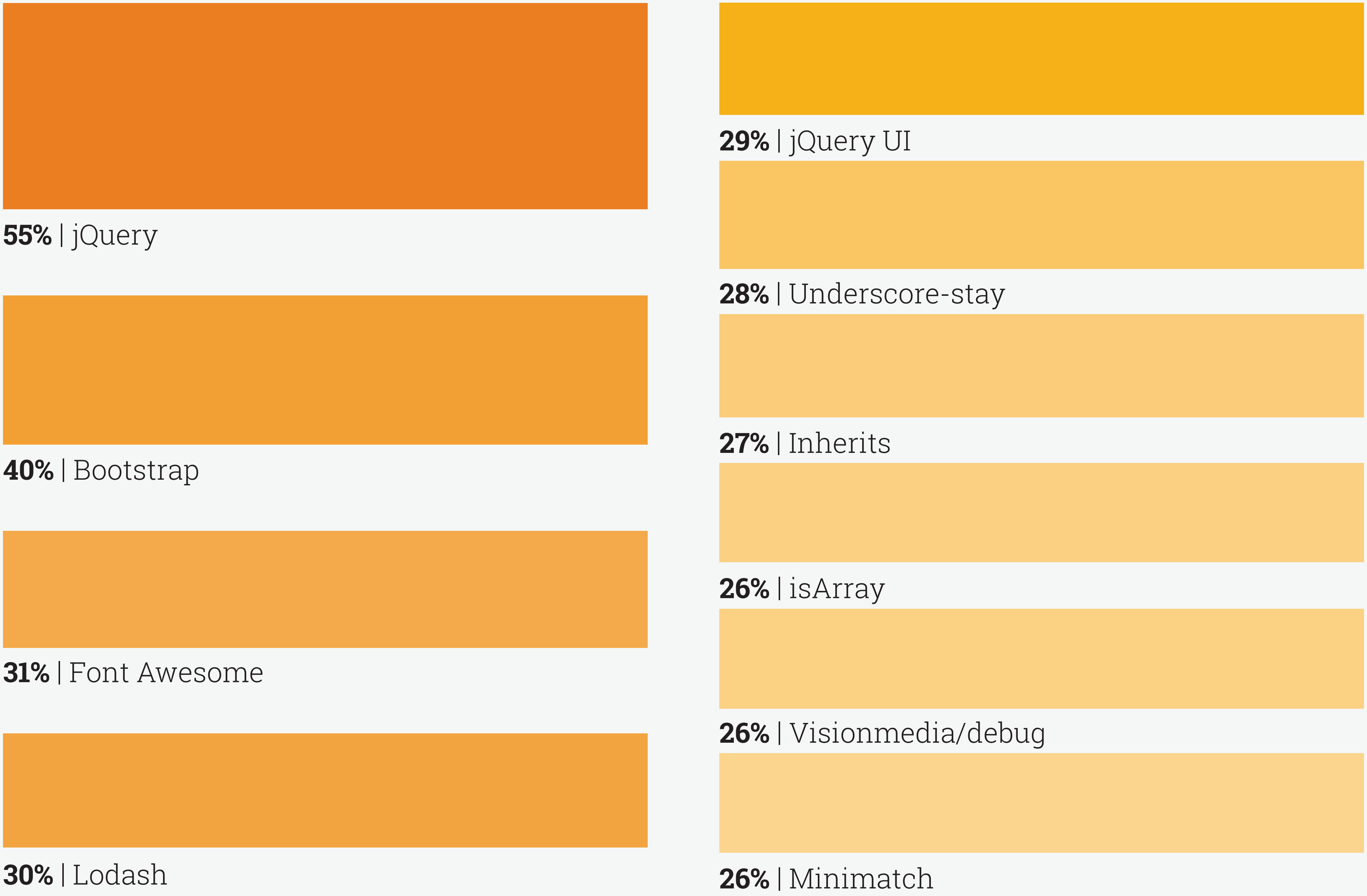
We found that 124 open source components were commonly used across the codebases of all 17 industries. The top five open source components (based on the percentage of codebases containing that component) included:

1. **jQuery:** a JavaScript library designed to simplify HTML
2. **Bootstrap:** a CSS framework directed at responsive, mobile-first front-end web development
3. **Font Awesome:** a font and icon toolkit
4. **Lodash:** a JavaScript library that provides utility functions for common programming tasks
5. **jQuery UI:** a collection of GUI widgets, animated visual effects, and themes

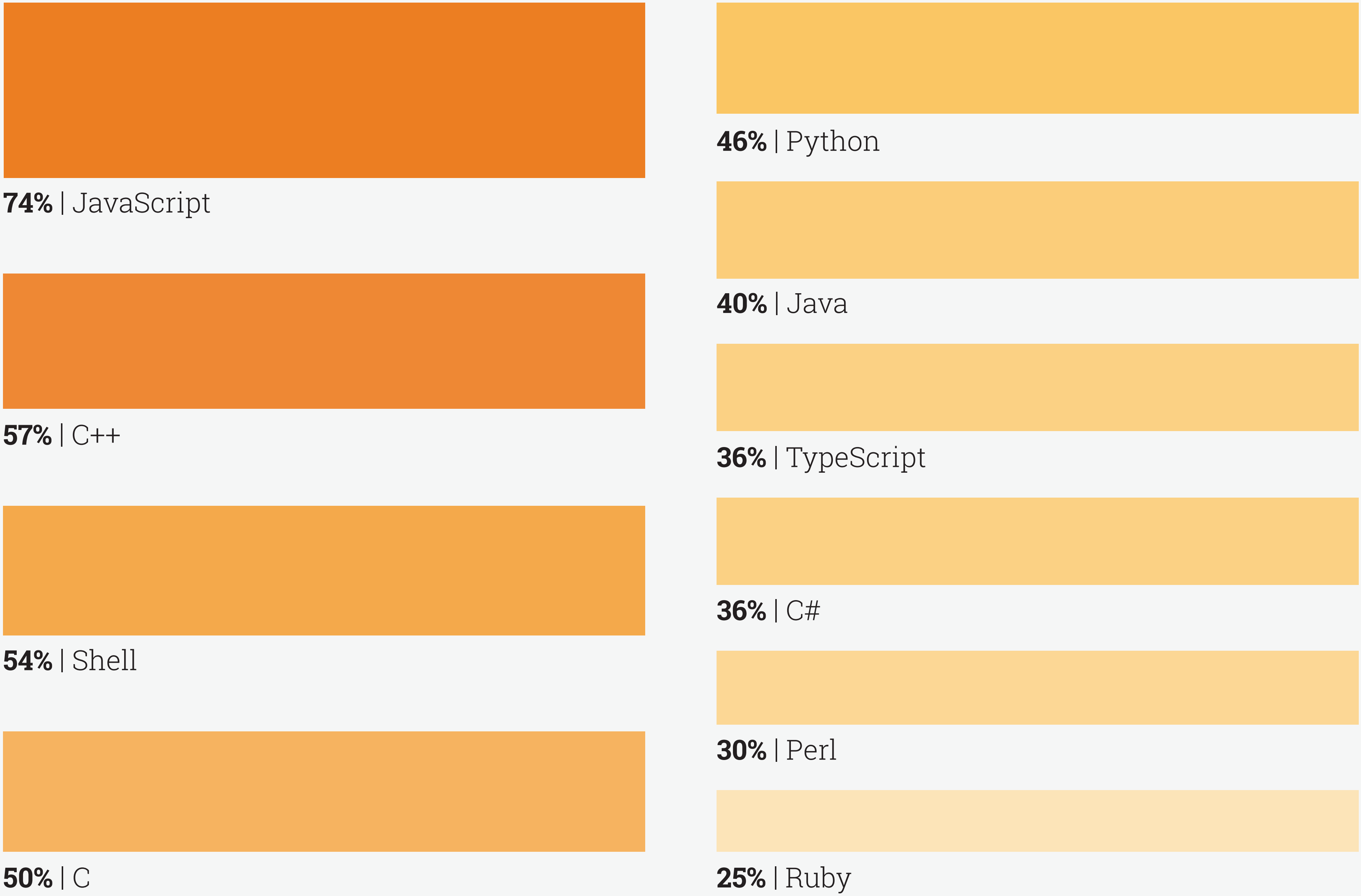
See the graphic on the next page for the top 10 open source components and the percentage of audited codebases containing that component.

JavaScript was the most commonly used programming language—found in 74% of the audited codebases. C++, Shell scripts, and C were found in 50% or more of the audited codebases. JavaScript was also the leading programming language of open source components, followed by C++ as a far second. See the graphics on pages 11 and 12 for more details.

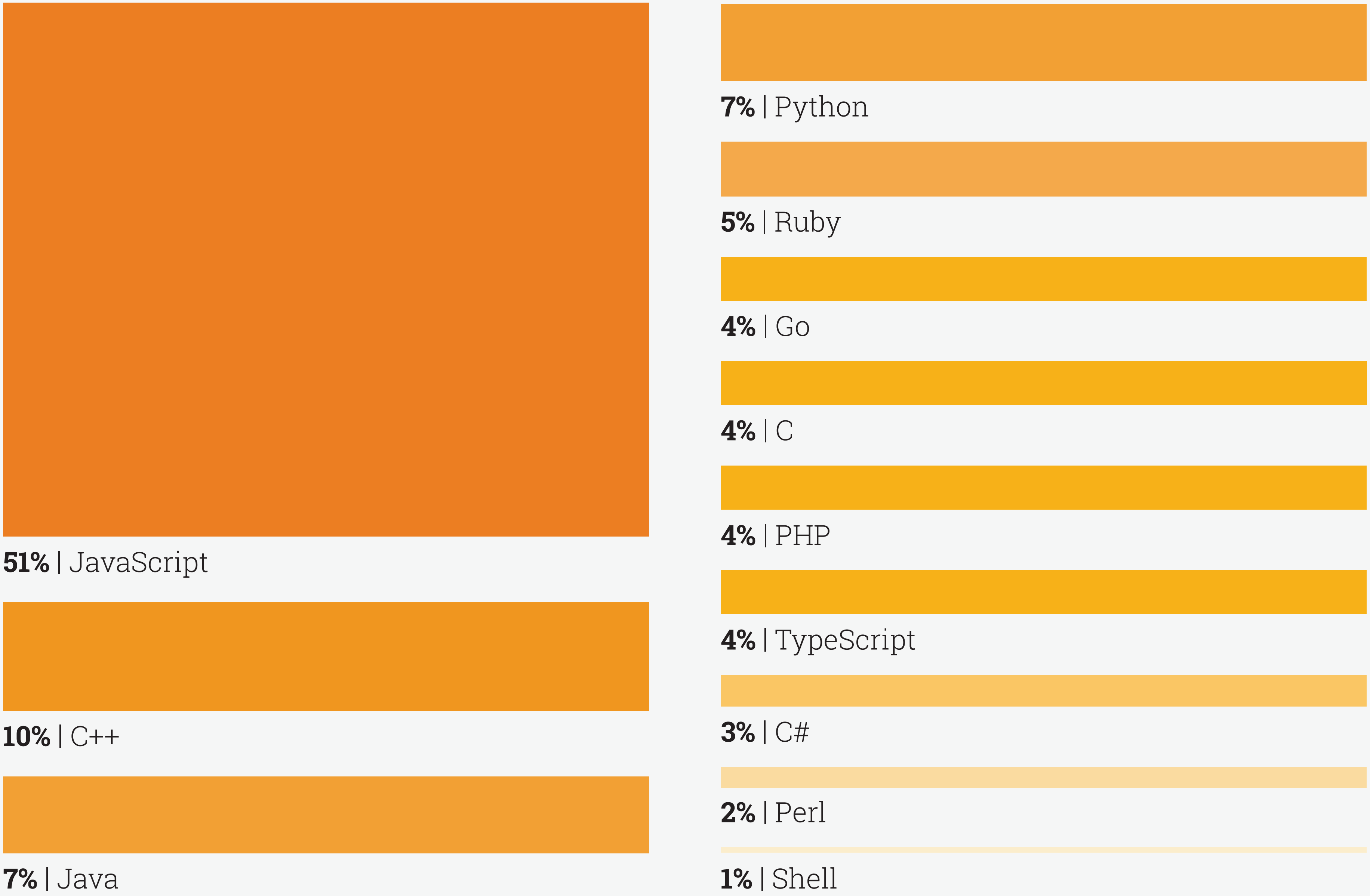
Top 10 open source components (percentage of codebases with the component)



Top 10 programming languages (percentage of codebases with the language)



Top 10 languages for open source (percentage of components using the language)



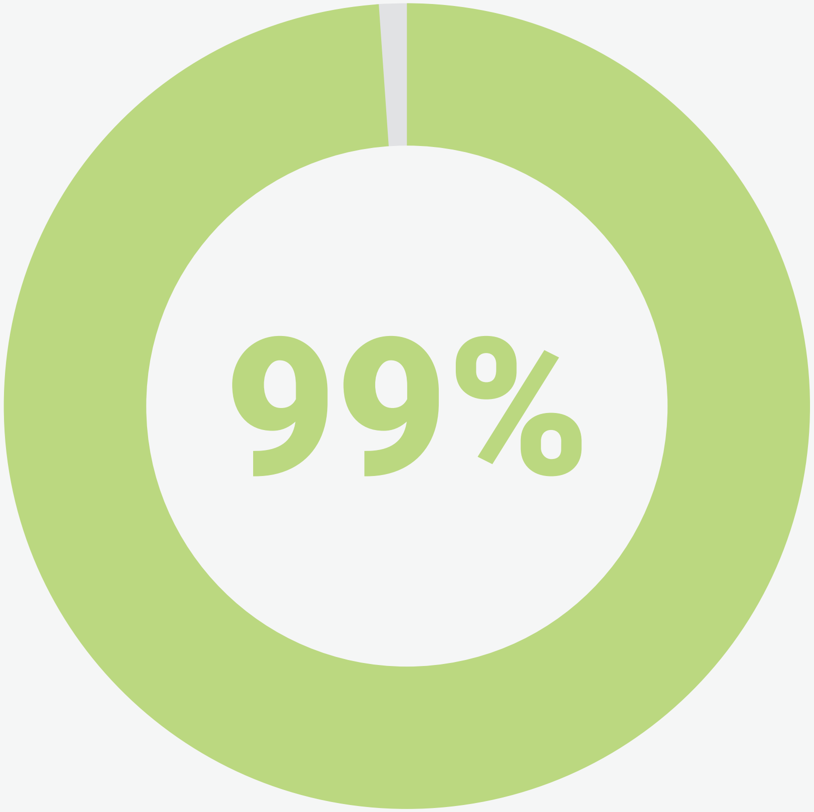
OPEN SOURCE RULES! BUT UNPATCHED VULNERABILITIES STILL THREATEN

Most organizations manage hundreds to thousands of software elements, ranging from mobile apps to cloud-based systems to legacy systems running on-premises. That software is typically a mix of commercial off-the-shelf packages and custom-built codebases, both of which are increasingly made up of open source components.

As we noted earlier, 99% of the codebases the Black Duck Audit Services team audited in 2019 contained open source. Here's the reality: If your organization builds or simply uses software, you can assume that software will contain open source. Whether you are a member of an IT, development, operations, or security team, if you don't have policies in place for identifying and patching known issues with the open source components you're using, you're not doing your job.

The open source community usually issues small updates at a much faster pace than the average commercial software vendor. When these updates contain security updates, companies need to have a strategy to adopt them rapidly. But because open source updates need to be "pulled" by users, an alarming number of companies consuming open source components don't apply the patches they need, opening their business to the risk of attack and applications to potential exploits.

In fact, many organizations are startlingly behind in using the latest version of any given open source component. As we'll detail in the section "Operational factors in open source use," 82% of the open source components found in our 2019 audits were out of date.



99% of the codebases the Black Duck Audit Services team audited in 2019 contained open source.

Augmenting CVE vulnerability information with BDSAs

This year we're not only reporting the most common CVEs (Common Vulnerabilities and Exposures) found in the audited codebases but also augmenting our report with vulnerability data published by our CyRC security research teams—the Black Duck Security Advisories (BDSAs).

A BDSA is a classification of open source vulnerabilities identified by the CyRC security research team. BDSAs provide early notification of vulnerabilities and deliver security insight, technical details, and upgrade/patch guidance.

Timeliness has always been a factor impacting the ability of the National Vulnerability Database (NVD) to publicize security vulnerabilities. There is often a significant time lag between the first disclosure of a vulnerability and its publication in the NVD, with some research reporting an average 27 days between initial announcement and NVD publication.[4] That time lag presents a huge window of opportunity for malicious actors to take advantage of vulnerabilities—an issue that BDSAs are designed to address.

Digging deeper into vulnerabilities found in 2019

Seventy-five percent of the codebases we audited in 2019 contained at least one public vulnerability—a depressing increase from the 60% of 2018, nearly returning to the 78% of 2017. An average of 82 vulnerabilities were identified per codebase. Four of the top 10 vulnerabilities found in the 2019 audited codebases did not have CVEs associated with them at the time of this writing. See the graphic on page 16 for the top 10 vulnerabilities uncovered in our 2019 audits.

The vulnerability addressed by BDSA 2014-0063 (related to CVE-2015-9251) and discovered in 23% of the audited codebases concerns security issues in jQuery 1.x and 2.x—specifically how scripts included in event attributes passed to the function parseHTML are executed immediately. This could leave a caller of this function vulnerable to cross-site scripting attacks if it does not properly sanitize untrusted input before passing it to the function.

In plain English, if you're currently using a version of jQuery earlier than 3.0 in your codebase, you should consider upgrading to guard against cross-site scripting attacks.

For a complete breakdown of the top 10 vulnerabilities discovered in our 2019 audits, see the appendix.

High-risk vulnerabilities

Similarly, the percentage of high-risk vulnerabilities increased to 49% in 2019, as opposed to 40% in 2018.

Slightly more encouraging were the results when we looked for infamous vulnerabilities. The Apache Struts vulnerability that was the target of the 2017 Equifax breach did not appear in any of the over 1,250 codebases audited, nor did the Heartbleed bug (disclosed in 2014 by members of our Finnish CyRC team). Concerning that last vulnerability, 2020 marks the first year since we began publishing the OSSRA report that we did not find Heartbleed in any of our audits of commercial software. *(continued on page 18)*

Top 10 vulnerabilities found (percentage of codebases)

37% | BDSA-2014-0063

37% | BDSA-2015-0567

36% | BDSA-2015-0110

34% | BDSA-2019-1138 (CVE-2019-11358)

30% | BDSA-2017-2930 (CVE-2015-9251)

27% | BDSA-2016-1585 (CVE-2016-10735)

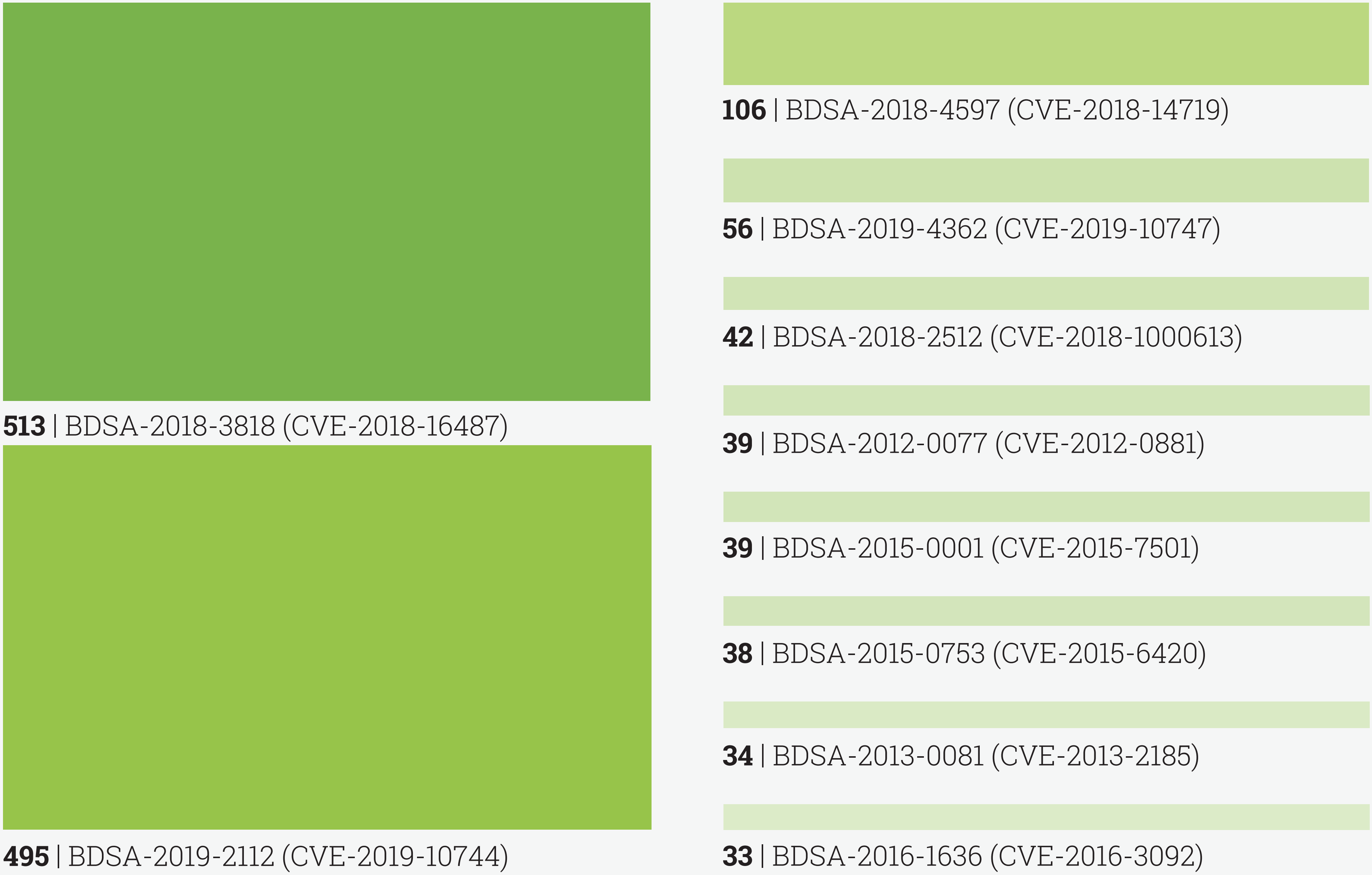
27% | BDSA-2016-1212

25% | BDSA-2018-4634 (CVE-2018-20677)

24% | BDSA-2018-3407 (CVE-2018-14042)

24% | BDSA-2018-3405 (CVE-2018-14040)

Top 10 high-risk vulnerabilities found (frequency of occurrences across all codebases)



Our results notwithstanding, Heartbleed still remains a global issue, with Shodan reporting over 91,000 instances of the vulnerability as of late 2019.[5]

Media publicity certainly played a role in prompting organizations to identify and resolve those well-known vulnerabilities. But media attention can't address the tens of thousands of equally problematic vulnerabilities, known only by their CVE listings, which also require identification and mitigation.

The average age (since first publication) of the vulnerabilities found in the audited codebases was a little less than 4 ½ years. The percentage of vulnerabilities older than 10 years was 19%. The oldest vulnerability—at a doddering 22 years—found in our audits was CVE-1999-0061.

As we noted earlier, 49% of the audited codebases contained high-risk vulnerabilities. The most common, CVE-2018-16487 (BDSA-2018-3818), a high-risk Lodash prototype pollution vulnerability affecting versions prior to 4.17.11, appeared over 500 times.

Polluting the prototype of a base object can sometimes lead to arbitrary code execution. For example, overwriting the prototype of a default JavaScript object may affect the behavior of all objects throughout the entire application. If an object method is hijacked, all objects may become polluted.

Yet another Lodash prototype pollution vulnerability frequently found in the 2019 scans (495 instances) was CVE-2019-10744 (BDSA-2019-2112), affecting all versions prior to 4.17.12. The dangers of both vulnerabilities range from property injection to code injection and denial of service.

Setting vulnerability patching priorities

There is a myth that the proverbial developer can fix each and every vulnerability, but no one can rationally expect developers to dig into vulnerabilities their management team hasn't prioritized for resolution. Your patch priorities should align with the business importance of the asset being patched, the criticality of the asset, and the risk of exploitation.

It's important to understand that patch policies for commercial software and open source components need to differ. While commercial vendors can push updates and security information, open source patches must originate from either the root project or the distribution channel where the component was originally obtained.

Only a fraction of open source vulnerabilities—such as those affecting Apache Struts or OpenSSL—are likely to be widely exploited. With that in mind, organizations should prioritize their open source vulnerability mitigation efforts based on CVSS (Common Vulnerability Scoring System) scores and CWE (Common Weakness Enumeration) information, as well as the availability of exploits, not only on "day zero" of a vulnerability disclosure but over the life cycle of the open source component.

Again, the importance of a comprehensive BOM is evident. To mitigate vulnerabilities, you first must know what software you're using and what exploits could impact their vulnerabilities.

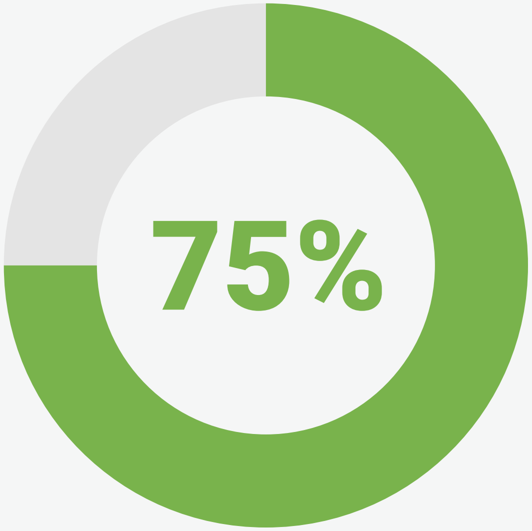
The **Common Vulnerability Scoring System** (CVSS) is an industry standard for assessing the severity of a vulnerability. Vulnerabilities in the National Vulnerability Database (NVD) have a base score that aids in calculating the severity and can be used as a factor for prioritizing remediation. The CVSS score (v2 and v3) provides an overall base score that takes both exploitability and impact into account.

Software composition analysis solutions, such as Black Duck SCA, can also provide customers with a temporal score in addition to the CVSS base, exploitability, and impact scores. Temporal scores take into account metrics that change over time owing to events that are external to the vulnerability. Remediation levels ("Is there an official fix available?") and report confidence ("Is the report confirmed?") can help temper the overall CVSS score to an appropriate level of risk.

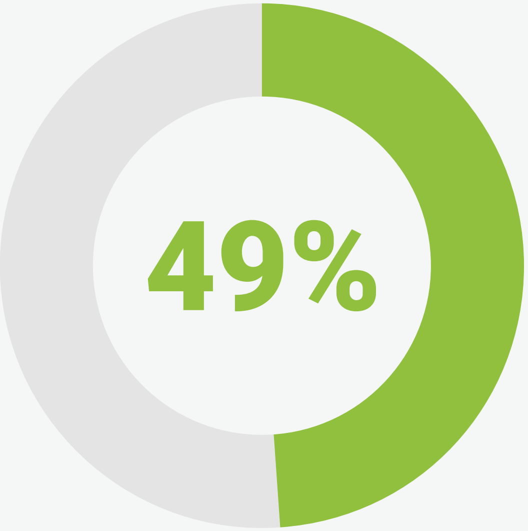
Common Weakness Enumeration (CWE) is a list of software or hardware weaknesses that have security ramifications. A CWE tells developers which weakness leads to the vulnerability in question. This information can help you understand what you're dealing with and adds one more piece to assessing the severity of the vulnerability. For example, a development team may prioritize a SQL injection differently than a buffer overflow or denial of service.

Is there an **exploit of the vulnerability**? The existence of an exploit will raise the risk score and help remediation teams prioritize highest-risk vulnerabilities first. Understanding whether there is an existing solution or workaround is another key piece of information to look to once you have assessed the overall risk. If you have two medium-risk vulnerabilities without exploits available, the final determination of which to fix first might come down to whether either has a solution or workaround available.

75% of the codebases we audited in 2019 contained at least one vulnerability.



49% of the audited codebases contained high-risk vulnerabilities.



OPEN SOURCE LICENSE AND LEGAL DEVELOPMENTS IN 2019

Open source license risk

According to copyright law, using software in any way requires permission in the form of a license describing the rights conveyed to users and the obligations those users must meet. Despite its reputation for being "free," open source software is no different from any other software in that its use is governed by a license.

An open source license is a type of license that allows the source code to be used, modified, or shared under defined terms and conditions. The Open Source Initiative (OSI), a nonprofit corporation that promotes the use of open source software in the commercial world, defines open source with 10 criteria and lists 82 OSI-approved licenses, with nine being "popular, widely used, or having strong communities." In contrast, the Software Package Data Exchange® (SPDX®), which focuses on commonly used licenses, lists some 350-odd commonly found open source licenses and includes the concept of deprecated licenses.

The Black Duck KnowledgeBase lists over 2,600 licenses associated with software whose source is freely available on the internet. Most of these licenses don't meet the strict OSI and SPDX definitions of "open source," and while many are acknowledgeable as one-offs, all specify rights, and many have obligations that users must attend to.

Black Duck analyses indicate that the 20 most popular licenses cover approximately 98% of the open source in use. But if your code uses an open source component, whether its license is one of those popular licenses or some variant, the license the author applied to it matters.

Popular open source licenses

The following OSI-approved licenses are popular, are widely used, or have strong communities:

- Apache License 2.0
- BSD 3-Clause "New" or "Revised" License
- BSD 2-Clause "Simplified" or "FreeBSD" License
- GNU General Public License (GPL)
- GNU Library or "Lesser" General Public License (LGPL)
- MIT License
- Mozilla Public License 2.0
- Common Development and Distribution License
- Eclipse Public License 2.0

BLACK DUCK ANALYSES INDICATE THAT THE **20 MOST POPULAR LICENSES COVER** APPROXIMATELY **98% OF THE OPEN SOURCE IN USE.**

Licensing legal developments in 2019

As open source becomes more ubiquitous, it has also become increasingly affected by societal issues, including both ethical and political issues. 2019 was a particularly volatile year in the world of open source licensing.

Activist Coraline Ada Ehmke created the Hippocratic License in 2019, adding the following provision to the MIT License: "The software may not be used by anyone for systems or activities that actively and knowingly endanger, harm, or otherwise threaten the physical, mental, economic, or general well-being of other individuals or groups, in violation of the United Nations Universal Declaration of Human Rights."

Another example is the JSON License, which essentially also uses the permissive MIT License with this addition: "The Software shall be used for Good, not Evil." In recent years, owners of many popular projects—notably, all Apache Foundation projects—have removed code under the JSON License because of the license's ambiguity.

Such license additions are often well intended but can still raise concerns, especially in merger and acquisition transactions, with lawyers needing to interpret the impact and risks of such modifications.

Many blockchain projects use open source licenses. In 2019, the Algorand blockchain project, a new blockchain, announced that its SDKs, example applications, and helper libraries were licensed under the permissive MIT License. However, the Algorand node software itself is licensed under the GNU Affero General Public License (AGPLv3), a reciprocal license published by the Free Software Foundation. Reciprocal licenses generally state that if you use a licensed component (or a derivative) in your software, you must make your source code available under the same conditions as the original component. This requirement is usually triggered when you distribute binaries of your software, but in the case of the AGPL, the trigger extends to the use of your software over a network. Additional restrictions cannot be placed on the licensee's exercise of the license.

Many companies' legal or compliance departments restrict them from using software licensed under the AGPLv3 because of the difficulty of ensuring compliance, which may jeopardize enterprise adoption of the Algorand project.

In late 2019, the U.S. Supreme Court granted Google's petition for *certiorari* in its ongoing copyright battle with Oracle Corp. over Google's use of 37 packages of Oracle's Java application programming interface (API) in the Android operating system. The Supreme Court will review the decision of the lower court and is expected to address the copyrightability of software and the defense of fair use.

A growing number of commercial open source companies have expressed concern that traditional open source licenses permit cloud service providers to use open source software without paying for it.

AS OPEN SOURCE BECOMES UBIQUITOUS, IT HAS BECOME INCREASINGLY AFFECTED BY SOCIETAL AND POLITICAL ISSUES.

In June 2019, CockroachDB—which provides open source software to store copies of data in multiple locations—adopted the Business Source License (BSL) restricting cloud providers from offering a commercial version of CockroachDB as a service without buying a license from the company. Redis Labs, providers of an open source database management system, introduced a hybrid Apache v2.0 license modified with the Commons Clause to limit the use of its product by cloud service providers. After confusion and controversy over the hybrid license, Redis created the Redis Source Available License (RSAL) in March 2019 for certain modules running on Redis, specifically restricting their use by database products.

In May 2019, the U.S. Department of Commerce placed Huawei Technologies Co., Ltd., on its so-called "Entity List," a list of companies that are unable to buy technology from U.S. companies without government approval. Google immediately pulled Huawei from its Android partner program and revoked its access to commercially licensed apps and Google services. While the U.S. Department of Commerce has granted multiple Temporary General License extensions to Huawei, there is an important open source aspect to this situation.

The Android open source license is Apache 2.0 and thus allows Huawei to continue using the base Android operating system, even though the base operating system license doesn't extend to any applications and proprietary extensions Google provides to its partners. This is an example of an "open-core" ecosystem where open source projects are supported by the commercial interests of vendors building on the open source project. In the case of Android, Google provides a number of value-add services via the Google Play Store but also provides a framework for compatibility testing. Google then limits access to its APIs for any Android device that hasn't passed compatibility testing.

Should all access to Google technologies be restricted permanently, Huawei can legally fork, or branch, their operating system from the Android Open Source Project. Doing so would not be without risk but would also incentivize Huawei to develop an independent Android experience that diverges from the one provided by Google. Within open source projects, forks are commonplace but can lead to independent ecosystems with differing levels of adoption. Debian vs. Fedora is a perfect example of this paradigm from the Linux world. While the situation with Huawei remains fluid (as of mid-March 2020, the U.S. government has granted its third temporary license extension), it is possible that the Android ecosystem will permanently split into two ecosystems: one based in the U.S. and the other based in China.

Examining license risk in open source components

Declared license conflicts arise when a codebase contains open source components whose licenses appear to conflict with the overall license of the codebase. For example, code under the GNU General Public License v2.0 (GPLv2) will generally pose a conflict issue when compiled into a normally distributed piece of commercial software. But the same code is not a problem in software that is considered software-as-a-service, or SaaS. The obligations of the GPL are triggered only on distribution of the associated software, and the GPL doesn't consider SaaS code to be "distributed." This is not to say SaaS software is immune from license conflicts; some licenses are problematic for SaaS as well.

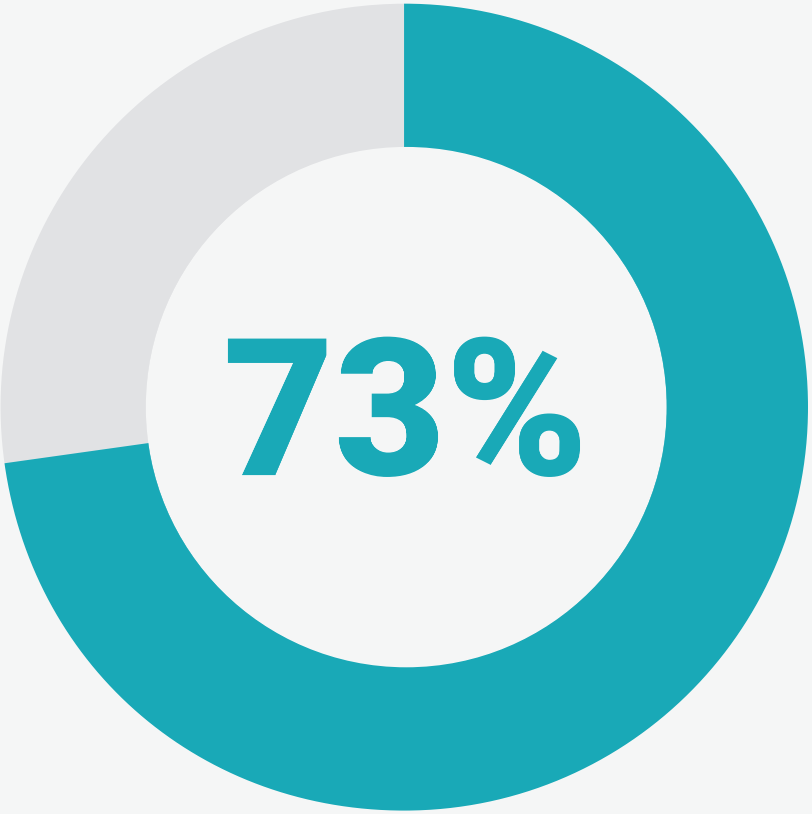
Black Duck Audits found that 67% of the 2019 audited codebases contained components with license conflicts, a percentage virtually unchanged from 2019. By industry, license conflicts ranged from a high of 93% (Internet & Mobile Apps) to a relative low of 59% (Virtual Reality, Gaming, Entertainment, Media).

The GPL is one of the more popular open source licenses, and its various versions can create license conflicts with other code in codebases. In fact, five of the top 10 licenses with conflicts were the GPL and its variants.

Open source components with no licenses or custom licenses

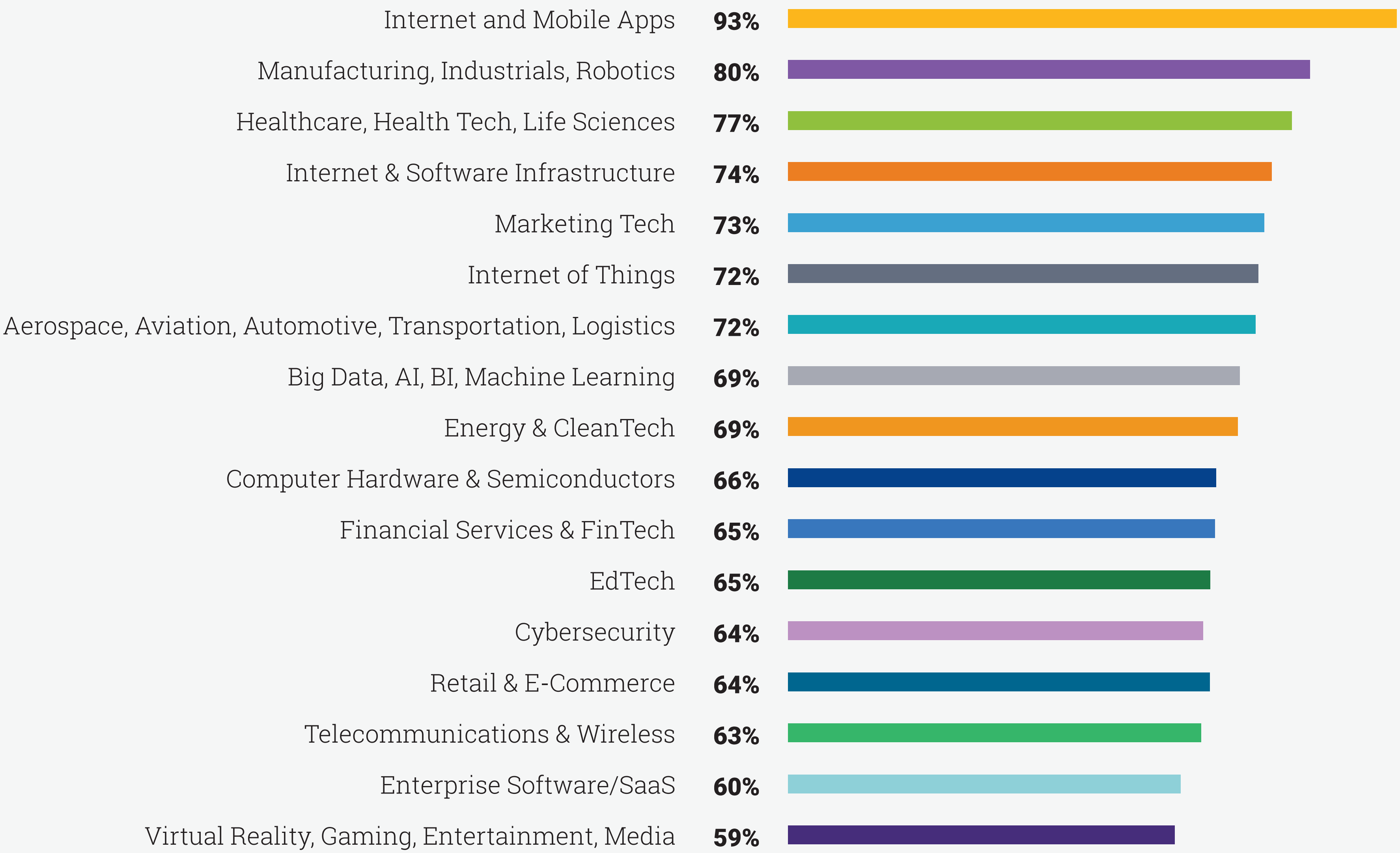
In the U.S. and many other jurisdictions, creative work is under exclusive copyright by default—including software code. Unless a license specifies otherwise (or the creators grant permission), no one else can legally use, copy, distribute, or modify that work without incurring the risk of litigation. Organizations that use code that is unlicensed are at greater risk of violating copyright law than those using licensed components.

Black Duck Audits designate a component as "not licensed" when the author has given no clear grant of license or terms



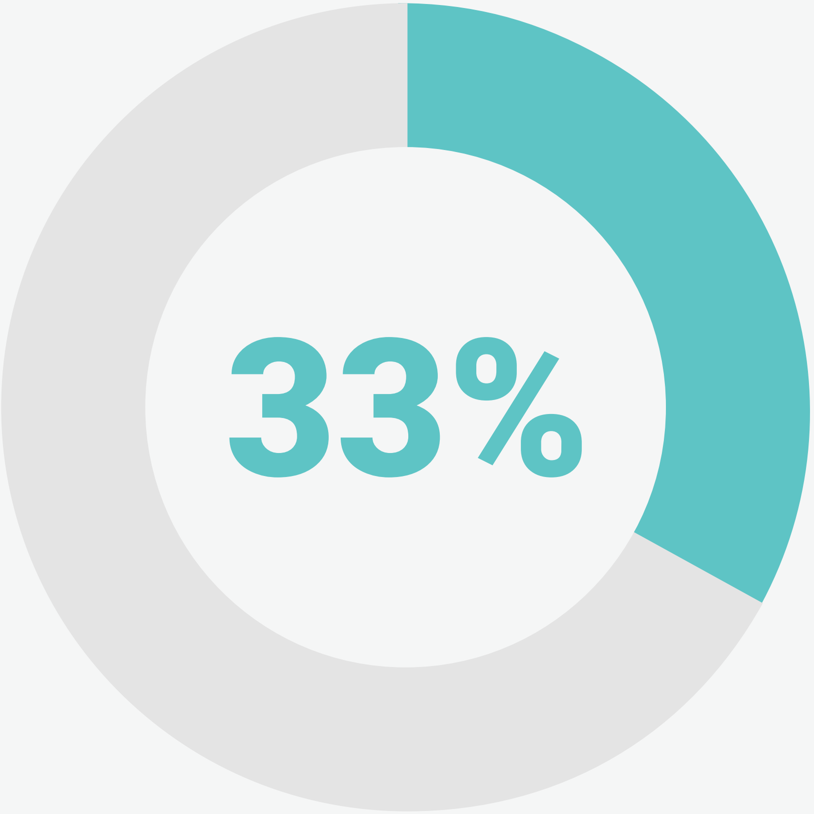
73% of the 2019 audited codebases contained components with license conflicts or no license.

License conflicts by industry (percentage of codebases)



of use in the code or associated files or on the site where it's hosted. Thirty-three percent of the codebases audited in 2019 contained components that fit the Black Duck Audits definition of "not licensed."

Custom licenses, on the other hand, are software components where the developer has used their own license language for the component, whether they've created the license wholesale or added to the language of a standard license, such as the Hippocratic License we mentioned earlier. As in the case of that license, lawyers will usually need to interpret the impact and risks of such modifications to the original license. In 31% of the codebases audited, Black Duck Audits found custom licenses that had the potential to cause conflict, or at least needed legal review, to determine the legal or business risk.



In 33% of the codebases audited, we found open source where the author had not given any clear grant of license or terms of use.

OPERATIONAL FACTORS IN OPEN SOURCE USE

During his examination of the current state of software composition analysis in “Technology Insight for Software Composition Analysis,” Gartner analyst Dale Gardner notes, “Mature organizations are expanding open-source management to include assessments of the overall ‘health’ of the software, based on a given package’s provenance and support.”[6]

As open source use has gained traction over the years, the focus on managing that open source has also evolved. Most organizations initially concentrated their efforts on open source license identification—a key part of any open source management strategy. As open source use grew in popularity, a risk beyond license risk emerged: identifying and mitigating known vulnerabilities, another critical factor of open source management.

Software “health” can be thought of as the state of an open source component’s code and whether anyone is maintaining that code at a given point in time.

One of the reasons behind the popularity of open source components is that viable open source projects usually have strong communities improving, updating, and patching vulnerability issues as they become known. Many developers don’t bother to vet the health of a community before downloading an open source component. However, even if a developer takes care to initially download components from robust open source communities, there’s no guarantee the community will remain active in maintaining that component or the specific version downloaded.

Black Duck Audits conducted in 2019 found that 91% of the codebases examined contained components that were more than four years out of date or had no development activity in the last two years. Besides adding to security risk, the danger of getting too far behind in versioning is that the simple act of updating to the latest version can introduce unwanted functional changes, such as the disappearance of key features.

Development teams might be concerned that using a newer version of an open source component will require modifying other code, causing a ripple effect that could bring development to a standstill. But many of those outdated components are the result of an “insert and forget” mindset. Developers typically don’t add version information about a component to the inventory spreadsheet before moving on to other work. Then, as long as the code continues to function as it’s supposed to, it’s ignored and eventually forgotten.

Eighty-eight of the codebases had components with no development activity in the last two years, exposing those components to a higher risk of vulnerabilities and exploits.

All software ages. As it ages, it loses support. With open source, the number of developers working to ensure updates—including feature improvements, as well as security and stability updates—decreases over time. The component becomes more likely to break without the support needed to provide fixes.

But at some point, as that open source component ages, it’s likely to break—or open a codebase to exploit. Without policies in place to address the risks that legacy open source can create, organizations open themselves up to the possibility of issues in their software.

91% OF CODEBASES HAD COMPONENTS THAT WERE MORE THAN FOUR YEARS OUT OF DATE OR HAD NO DEVELOPMENT ACTIVITY IN THE LAST TWO YEARS.

CONCLUSION

Five years after the publication of our first OSSRA report, our message remains the same:

As the data demonstrates, modern applications consistently contain a wealth of open source components with possible security, licensing, and code quality issues. How you manage your open source usage matters greatly. The more diligent your attention to the differences between commercial software and open source, the better the outcomes.

If your organization develops software, your codebases almost certainly include numerous open source components. The data makes it clear: You need processes and policies to manage open source components and libraries; to evaluate and mitigate your open source quality, security, and license risks; and to continuously monitor for vulnerabilities, upgrades, and the overall health of the open source you use.

Whether you are focused on creating the next great software innovation, buying core technologies to enable you to address new markets or innovate faster, or planning to buy/sell technical assets, the attention you pay to open source governance will pay off with higher quality products.

Here are our recommendations.

Core activity: Inventory your open source now

You can't possibly address any issues without an up-to-date, accurate software inventory—a.k.a. a software BOM—that includes all open source components, the versions in use, and download locations for each project in use or in development. The BOM should also include all dependencies, or the libraries your code is calling to, as well the libraries those dependencies are linked to.

This first step of creating a BOM is often the most daunting for many organizations, who worry about the possible impact of manually creating and maintaining such an inventory on developer productivity and development costs. If that's of concern to your organization, consider the advice of Gartner analyst Dale Gardner:

"A BOM generated by an SCA tool provides more comprehensive information (specific versions, license, etc.), and potentially a more advanced understanding of dependency mapping among various components and frameworks."[7]

Armed with the BOM, you can now manage your risks properly.

Security teams: Monitor for changes in external threats and vulnerability disclosures

Public sources, such as the National Vulnerability Database (NVD), are a good first step for information on publicly disclosed vulnerabilities in open source software. Keep in mind, however, that over 115 organizations contribute entries to the NVD. Not only does the NVD reflect those organizations' priorities, but there can be significant lags in data reporting, scoring, and actionability of the data in a CVE entry. Also, the format of NVD records often makes it difficult to determine which versions of a given open source component are affected by a vulnerability.

These factors make it unwise to rely solely on the NVD for vulnerability information. Instead, look to a secondary source that provides earlier notification of vulnerabilities affecting your codebase and, ideally, delivers security insight, technical details, and upgrade and patch guidance.

The job of open source management doesn't stop when the codebase ships in an application. Organizations need to continuously monitor for new threats for as long as their applications remain in service.

Once a threat has been identified, you'll need to determine what remediation needs to be done, assign the remediation work to the appropriate people, and track the remediation process: what's being reviewed, what's been reviewed, what's been fixed, what fixes have been deferred, and what's been patched.

Development and legal teams: Create policies to manage your open source activities

Educate your developers about the need for managed use of open source. By having clear policies and procedures around the introduction and documentation of new open source components, you can help to ensure you're controlling what enters the codebase and that it complies with company policies.

Consider putting in place an automated process that tracks open source components and their licenses and known security vulnerabilities, as well as operational risks such as versioning and duplications, and prioritizes issues based on their severity.

If you build packaged, embedded, or commercial SaaS software, open source license compliance should be a key concern. You'll need to determine the license types and terms for the open source components you use and ensure they're compatible with the packaging and distribution of your software. Even companies whose software is not their product per se are subject to license terms and should pay heed.

Using your BOM of open source components, you'll want to compile detailed license texts associated with those components so that you can flag any components not compatible with your software's distribution and license requirements. You'll also want to ensure the obligations of those licenses have been met, as even the most permissive open source licenses still contain an obligation for attribution.

You may want to involve your organization's general counsel—or seek outside legal advice—as understanding licensing terms and conditions and identifying conflicts among various licenses can be challenging. You'll want to get this right the first time, especially if you build packaged or embedded software, as license terms are often more explicit for shipped software and harder to mitigate after the fact.

M&A teams (buyers and sellers): Perform an open source due diligence audit

Are you involved in M&A transactions where software is a major part of the deal? If the software assets are a significant part of the valuation of the target company, a third party should audit the code for open source.

All software technology has issues, but it's critical for both sellers and buyers to have a clear picture before the deal is closed so they can address these issues. Risks in software include legal risks, primarily around improper licensing; security risks, or vulnerabilities in the code or design that could be exploited; and software quality risks. And buyers will also want to know if the code will have maintenance issues due to operational factors.

Everyone: Engage with your open source communities!
Open source is the foundation of software development, and it's built by people who have taken the time to contribute their skills and knowledge. But many people who rely on open source software have little understanding of how open source communities work or how to contribute to an open source project. It's not just code—whether you're a writer, translator, designer, event planner, or information security or legal specialist, you too can play a role in the open source community.

From a pragmatic standpoint, engaging with the communities whose open source projects your organization relies on is one of the best ways to ensure those projects stay healthy, vital, and up to date. Plus, you get the benefit of learning when important changes are in the works.

How do you get started? We recommend VM (Vicky) Brasseur's Forge Your Future With Open Source, a quick-start guide on how and when to work on an open source project.

Start making a contribution!

References

1. Gartner, Dale Gardner, Technology Insight for Software Composition Analysis, Nov. 1, 2019.

2. Ibid.

3. Frank Nagle, Jessica Wilkerson, James Dana, and Jennifer L. Hoffman, Vulnerabilities in the Core: Preliminary Report and Census II of Open Source Software, The Linux Foundation & The Laboratory for Innovation Science at Harvard, February 2020.

4. NopSec, 2018 State of Vulnerability Risk Management Report, Aug. 2019.

5. Shodan, Heartbleed Report, accessed Apr. 8, 2020.

6. Gartner, Dale Gardner, Technology Insight for Software Composition Analysis, Nov. 1, 2019

7. Ibid.

Appendix: Top 10 vulnerabilities found in 2019 audits

BDSA-2014-0063 is a high-severity vulnerability where jQuery is vulnerable to cross-site scripting (XSS) due to lack of validation of user-supplied input. A fix is available.

The vulnerability our security advisory classifies as **BDSA-2015-0567** affects all jQuery versions that make use of an unpatched UglifyJS parser, opening them to arbitrary code execution through crafted JavaScript files. This high-severity vulnerability was found in 22% of the audited codebases. A fix is available.

BDSA-2015-0110 is another high-severity vulnerability. FileAPI is vulnerable to cross-site scripting (XSS) in the FileAPI.flash.swf component via the ExternalInterface.call function. Without input validation, an attacker can cause arbitrary JavaScript to be executed in a victim's browser, which could allow the attacker to obtain sensitive information such as authentication tokens and user session cookies. No exploit has been published as of this writing, and a fix is available.

BDSA-2019-1138 (CVE-2019-11358) concerns an improper input validation vulnerability discovered in jQuery. An attacker could exploit this vulnerability to execute cross-site scripting (XSS) attacks, trigger a denial-of-service (DoS) condition, or gain unauthorized access to the application. No exploit has been published as of this writing, and a fix is available.

BDSA-2017-2930 (CVE-2015-9251) explains that jQuery is vulnerable to cross-site scripting (XSS) due to the way it processes certain types of Ajax requests. This can allow potential attackers to execute arbitrary code on the target system. No exploit has been published as of this writing, and a fix is available.

The final five BDSAs concern the Bootstrap open source component and various cross-site scripting (XSS) vulnerabilities affecting it. **BDSA-2016-1585** notes that Bootstrap is vulnerable to cross-site scripting due to the insufficient sanitization of user-provided input. An attacker could execute malicious scripts within a victim's browser by tricking them into clicking on a crafted link, allowing the attacker to obtain sensitive information such as browser cookies.

BDSA-2016-1212 and **BDSA-2018-4634** warn that an attacker could use Bootstrap XSS vulnerabilities to execute arbitrary JavaScript in the target's browser by crafting a malicious input. Among other bad outcomes, an attacker could steal an administrator's session tokens or execute arbitrary code on their behalf by sending the link to an unsuspecting user or waiting for them to discover it, or run malicious scripts on a victim's browser, should they follow the attacker's crafted link.

The last two BDSAs (**BDSA-2018-3407** and **BDSA-2018-3405**) address reflected XSS vulnerabilities, where an attacker could cause a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser.

Exploits have been published for all these Bootstrap vulnerabilities, and fixes are available for all.

The Synopsys difference

Synopsys helps development teams build secure, high-quality software, minimizing risks while maximizing speed and productivity. Synopsys, a recognized leader in application security, provides static analysis, software composition analysis, and dynamic analysis solutions that enable teams to quickly find and fix vulnerabilities and defects in proprietary code, open source components, and application behavior. With a combination of industry-leading tools, services, and expertise, only Synopsys helps organizations optimize security and quality in DevSecOps and throughout the software development life cycle.

About CyRC

The Synopsys Cybersecurity Research Center (CyRC) works to accelerate access to information around the identification, severity, exploitation, mitigation, and defense against software vulnerabilities. Operating within the greater Synopsys mission of making the software that powers our lives safer and of the highest quality, CyRC helps increase awareness of issues by publishing research supporting strong cybersecurity practices.

For more information, go to www.synopsys.com/software.

Synopsys, Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

Contact us:
U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com