Contrast SECURITY

2021 State of Open-Source Security Report

Trends and Best Practices from Real-world Software Supply Chains

Table of contents

01 | Forward

Already an accelerating trend before the world-changing events of 2020, digital transformation is now moving at breakneck speed to bring radical change to the way organizations conduct business.

Applications are at the heart of this phenomenon, delivering new experiences for both business customers and consumers while improving operational efficiency and creating new revenue streams.

Somewhat hidden in this process are millions of software developers, who have honed their craft to the point that it functions as a fast and efficient "software factory," with extensive automation and standardization of processes across the software development life cycle (SDLC). Using methodologies like Agile and DevOps, they have accelerated release cycles while improving quality. One practice that contributes to this efficiency is code reuse, which includes open-source libraries and frameworks. The typical application today contains dozens and quite often hundreds of libraries, many of which provide indispensable core functionality and help propel digital transformation.

But the efficiency brought about by the extensive use of libraries is not without risk. The increased reliance on applications has not escaped the attention of cyber criminals, who have shifted more attention to this attack vector. The massive SolarWinds attack that was revealed in late 2020 is a stark reminder of the vulnerability of the software supply chain and the risk it poses.

Recognizing the importance of securing the software supply chain, Contrast Labs is pleased to announce the publication of research findings regarding open-source utilization and risk. The analysis is based on telemetry from tens of thousands of real-world applications and application programming interfaces (APIs) that are assessed and protected by Contrast solutions. This data comes from real-world examples of the software supply chain.

The report identifies five areas of risk around open-source libraries and frameworks: active and inactive libraries, active and inactive library classes, library age, open-source vulnerabilities, and licensing risk.

Each of these areas brings risk to organizations that can hamper operational efficiency, the ability to prevent and thwart attacks by cyber criminals, and avoid legal problems regarding software ownership.

We frequently assert that not every software vulnerability should be treated the same, and this is especially true with open-source software. Indeed, our data shows that 62% of libraries present in an application are not used at all by the software, and thus they present no risk. But the issue is deeper: Of libraries that are used, only 31% of the classes in those libraries are invoked by the application. The truth is, while third-party libraries comprise the majority of an application in terms of lines of code, less than one-tenth of the code that actually runs comes from open source. The rest comes from custom code written by developers. Unfortunately, legacy software composition analysis (SCA) tools focus on everything equally, and fail to identify what really matters. This ratchets up risk while increasing inefficiencies.

The result is that a huge share of the vulnerabilities found in open-source code in a typical application are inactive and pose no risk. Further, as traditional SCA tools identify all vulnerabilities and view them the same, this translates into a tremendous amount of wasted time. This operational inefficiency is compounded by the fact that not all vulnerabilities found in active libraries and classes should be treated the same—only a fraction pose serious risk. The lack of comprehensive observability also impacts the ability to track and manage open-source licensing: A surprising percentage of applications have open-source licensing exposures.

Open-source software is firmly embedded in every organization's software stack. Each company must adapt its software factory with processes and technologies to identify software supply chain issues and prevent them from exposing the businesses to attack. Our goal with the 2021 State of Open-source Security Report is to help organizations understand the layers of risk presented by open-source software, and the strategies they can employ to mitigate that risk. Taking these steps can help organizations to take advantage of the full potential that modern software offers to organizations in all industries, while minimizing risk.

Sincerely,

JEFF WILLIAMS
CTO and Co-Founder

DAVID LINDNER
Chief Information Security Officer

02 | Executive Summary

As open-source libraries continue to increase in importance to developers in producing business-critical software against aggressive deadlines, such libraries proliferate in number and in complexity. The 2021 Contrast Labs Open-source Security Report uses telemetry from actual applications protected by Contrast OSS and Contrast Assess to reveal key trends about library usage, vulnerabilities, and best practices from thousands of real-world software supply chains. Key findings include:

- While the average application contains 118 libraries, the more important metric is that only 38% of libraries are active—that is, used by the application. Further, only 31% of library classes within active libraries are actually ever invoked by a given piece of software. While libraries comprise a large percentage of the lines of code present in an application, less than 10% of code in applications is active third-party library code.
- The average library uses a version that is 2.6 years old. This increases the risk of unaddressed vulnerabilities while expanding the amount of work required when an update is finally done.
- The average Java application has 50 open-source library vulnerabilities, and the odds are 16% that a given Java library in an application will have a vulnerability.
- Software composition analysis (SCA) tools, which do not differentiate between vulnerabilities in inactive libraries and classes and active ones, return false positives when they identify a CVE that poses no risk. The false positivity rate is 23% for Java applications, 13% for .NET applications, and 69% for Node applications.
- High-risk licenses are present in 69% of Java applications and 33% of Node applications. These expose organizations to significant legal risk by legally obligating the license holder to make any resulting software open source.

Given recent vulnerability exposures and attacks of the software supply chain, it is imperative that organizations pay much closer attention to the open-source code used in their applications. There are significant risks in open-source libraries, but identifying and remediating the ones that matter requires a different approach, one that provides a comprehensive picture of active and inactive libraries and classes, library age, vulnerabilities, and licensing issues. Legacy SCA and application security tools simply do not provide the level of accuracy and observability required—especially when the C-suite and boards of directors are pressing for greater business acceleration.

Key Findings

118

LIBRARY USE

The average application contains 118 open-source libraries.

38%

ACTIVE/INACTIVE LIBRARIES

Only 38% of libraries present in applications are used; Node applications are the lowest of all languages with only 24%.

32%

ACTIVE/INACTIVE LIBRARY CLASSES

Only 32% of classes are invoked by active Java libraries.

2.6

LIBRARY AGE

The average library uses a version that is 2.6 years old.

Key Findings

OPEN-SOURCE VULNERABILITIES

50

The average Java application has 50 open-source vulnerabilities.

16%

Java libraries in applications have a 16% chance of having a Critical or Major vulnerability.

44%

The odds of an application having a vulnerability in a Java library increase from 7% to 44% as the library age goes from 1 year to 4 years.

FALSE POSITIVITY RATES FOR LEGACY SCA TOOLS:

FOR JAVA

23%

FOR .NET

13%

FOR NODE

69%

LICENSING

69%

69% of Java applications and 33% of Node applications include a library with a high-risk license.

99%

99% of organizations have at least one high-risk Java license.

03 | Introduction

The discipline of software development has dramatically improved its speed and efficiency in recent years. Methodologies like Agile and DevOps leverage principles from manufacturing to streamline and automate as much of the software development life cycle (SDLC) as possible. These advances not only enable software to be developed much more quickly than a decade ago but have also improved the quality of the software from both a back end and user experience perspective. The transformation has been so complete that the term "software factory" has recently been resurrected to describe the operation.[1]

Like a well-run manufacturing floor, today's software factory uses a unified team for every aspect of the SDLC, from development to operations. The software factory team uses clear policies, automated processes, and standardized development tools. And importantly, they leverage software reuse as a deliberate strategy. While some of that repurposed code comes from internal repositories, much of it comes from open-source libraries.

The efficiency and effectiveness gains from this approach are real. A recent McKinsey report found that open-source adoption was the biggest differentiator for organizations in the top quartile of their Developer Velocity Index (DVI).[2] As the authors of the study note, "We found that building an open-source culture is about more than using open-source software within the code; it extends to encouraging contribution and participation in the open-source community as well as adopting a similar approach to how code is shared internally—that is, strong InnerSource adoption."[3]

SECURITY CHALLENGES FOR OPEN-SOURCE LIBRARIES AND FRAMEWORKS

But this increase in efficiency is not without cost. The massive SolarWinds application attack[4] is a reminder that the software factory is a target for cyber criminals. The 2020 Verizon Data Breach Investigations Report found that 43% of data breaches this past year were the result of a web application vulnerability—a figure that more than doubled over the previous year.[5] And the number of open-source vulnerabilities logged into the Common Vulnerabilities and Exposures (CVE) database has increased dramatically in recent years.

Another security challenge involves the increasing complexity of library use in applications. Imagine a library with several functions—A, B, and C. This library relies on numerous other libraries (called "transitive dependencies") to implement those functions. A developer wanting to use function A will inadvertently include all the libraries that support functions B and C in the application.

These complex dependency trees make developers reluctant to remove or update old libraries, fearful that doing so will have unforeseen downstream consequences. In other cases, they waste time by updating libraries that are not used by the software in any way.

INADEQUACIES OF LEGACY OPEN-SOURCE SECURITY APPROACHES

Despite these increasing complications, most organizations still employ open-source security strategies that were developed many years ago, when open-source software was less complex, comprised a smaller part of applications, and was a part of a more deliberate development process. Legacy software composition analysis (SCA) tools depend on periodic static scans of either built applications or the build files in code repositories. These scans are disruptive to modern native development processes. Worse, they show data from just a specific point in time rather than providing continuous analysis. The scans are out of date the first time there is a library change or update.

But perhaps most detrimental is legacy SCA tools' lack of visibility into which libraries and classes are actually used by the software, how they are used, and what version is in use. As a result, all vulnerabilities of each severity level are presented as equally risky, when some pose no risk. Just as false positives from application security scanning tools cause developers and security experts to waste time on items that pose no risk, a lack of visibility into software dependencies creates false positives when SCA tools identify CVEs in code that is not used by the software. Both types of false positives waste an organization's staff time and potentially can delay the remediation of vulnerabilities that truly pose risk.

As the findings of this report clearly demonstrate, full observability of the all open-source library content in each application is a necessity for ensuring the security of applications for employees, partners, and customers.

METHODOLOGY OF THIS STUDY

The data in this report is based on aggregate telemetry collected by Contrast Labs from Java, .NET, and Node applications covered by Contrast OSS and Contrast Assess. From this data, we identify and quantify five layers of risk faced by users of open-source software:

- Risk from active and inactive libraries
- Risk from active and inactive library classes
- Risk due to library age
- Risk due to open-source vulnerabilities
- Risk associated with licensing

04 | Library Counts: Indicative of Complexity, not Necessarily Risk

Many observers would be surprised at the number of third-party libraries that are included in a typical piece of software. Contrast OSS telemetry data shows that the average application contains 118 libraries. While nearly one-quarter (24%) of applications contain fewer than 25 libraries, the same percentage have more than 150. At the same time, 52% of applications contain fewer than 75 libraries (Figure 1). This highlights the varying open-source risk from application to application and speaks to the increasing complexity of software today.

FIGURE 1

Percentage of overall applications by library count.

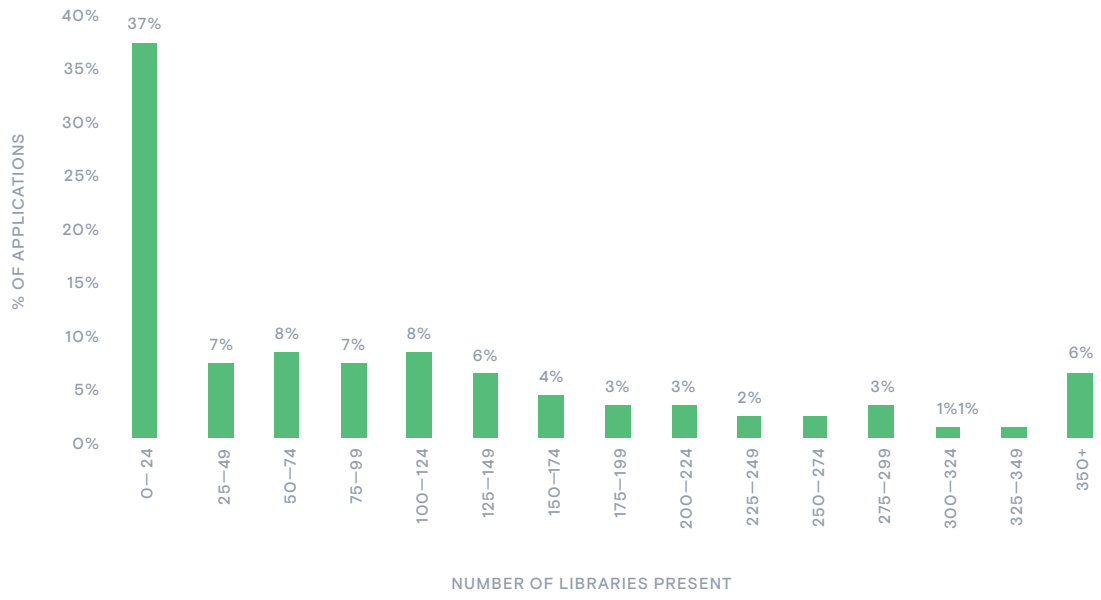
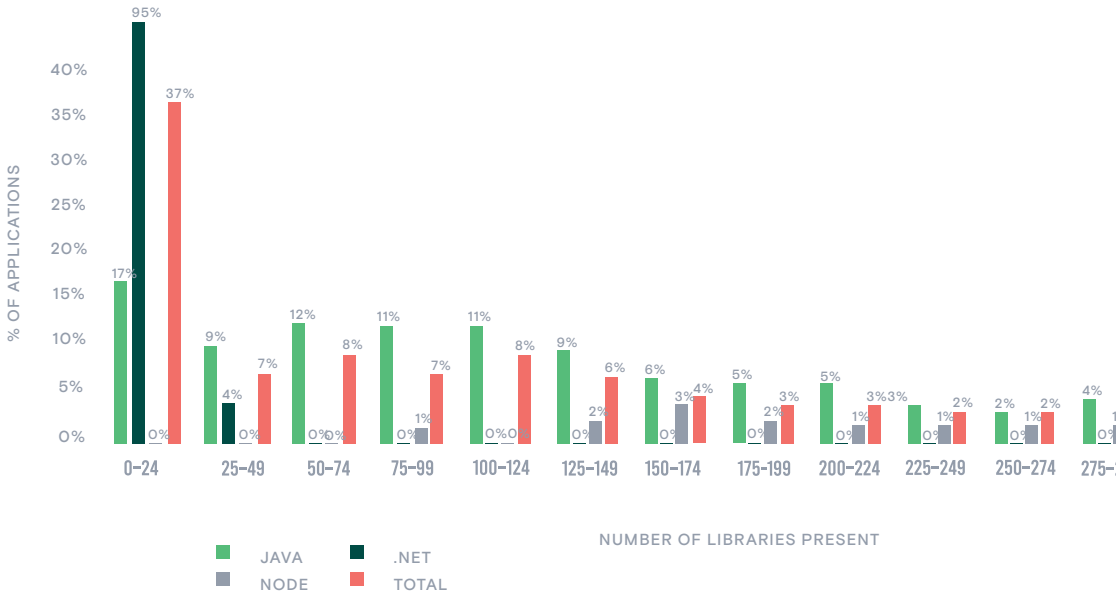


FIGURE 2

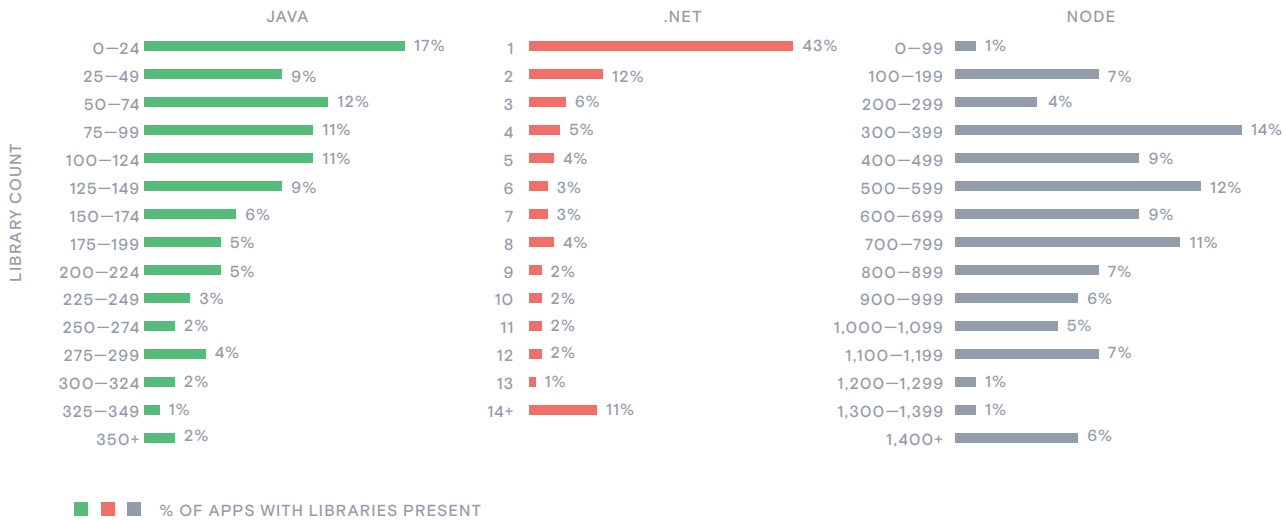
Percentage of applications containing different numbers of libraries, by language



Many observers would be surprised at the number of third-party libraries that are included in a typical piece of software. Contrast OSS telemetry data shows that the average application contains 118 libraries. While nearly one-quarter (24%) of applications contain fewer than 25 libraries, the same percentage have more than 150. At the same time, 52% of applications contain fewer than 75 libraries (Figure 1). This highlights the varying open-source risk from application to application and speaks to the increasing complexity of software today.

FIGURE 3

Percentage of applications by library count, by language.

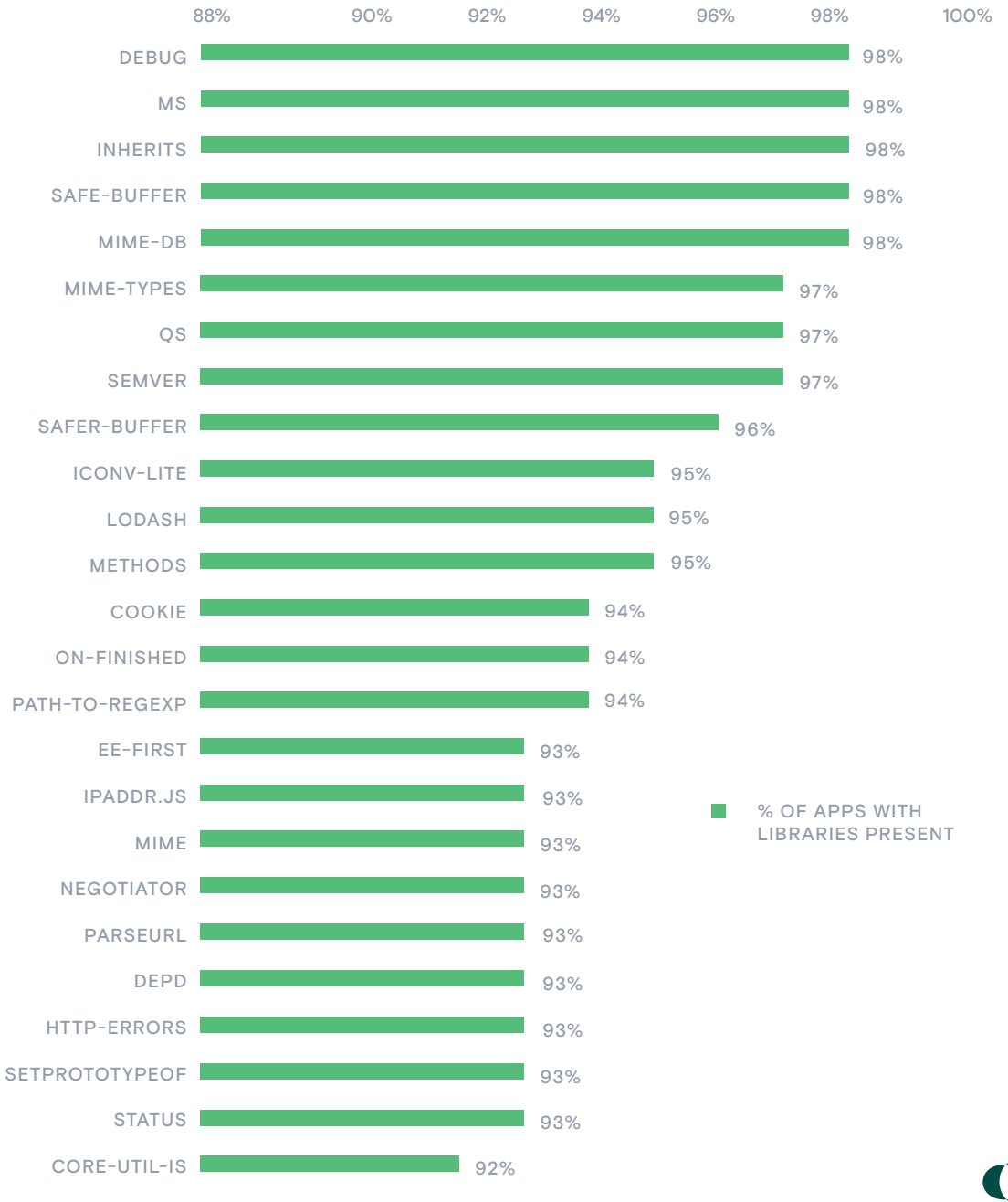


LIBRARY USAGE BY LANGUAGE

While the mean Java application contains 125 libraries, the median is 100, with 50% of applications having fewer than that number (Figure 3). Because the mean is higher than the median, the interpretive result means there are a select number of Java applications with a disproportionately high rate of library vulnerabilities. Specifically, 16% of Java applications have more than 200 libraries, and 8% have more than 250. The slf4j-api library is found in 79% of Java applications, and another 10 libraries are found in more than 70% (Figure 4). All of the top 25 libraries are found in a majority of Java applications. This means that an attacker who infiltrates a single library can potentially compromise a large percentage of the world's Java applications.

FIGURE 4

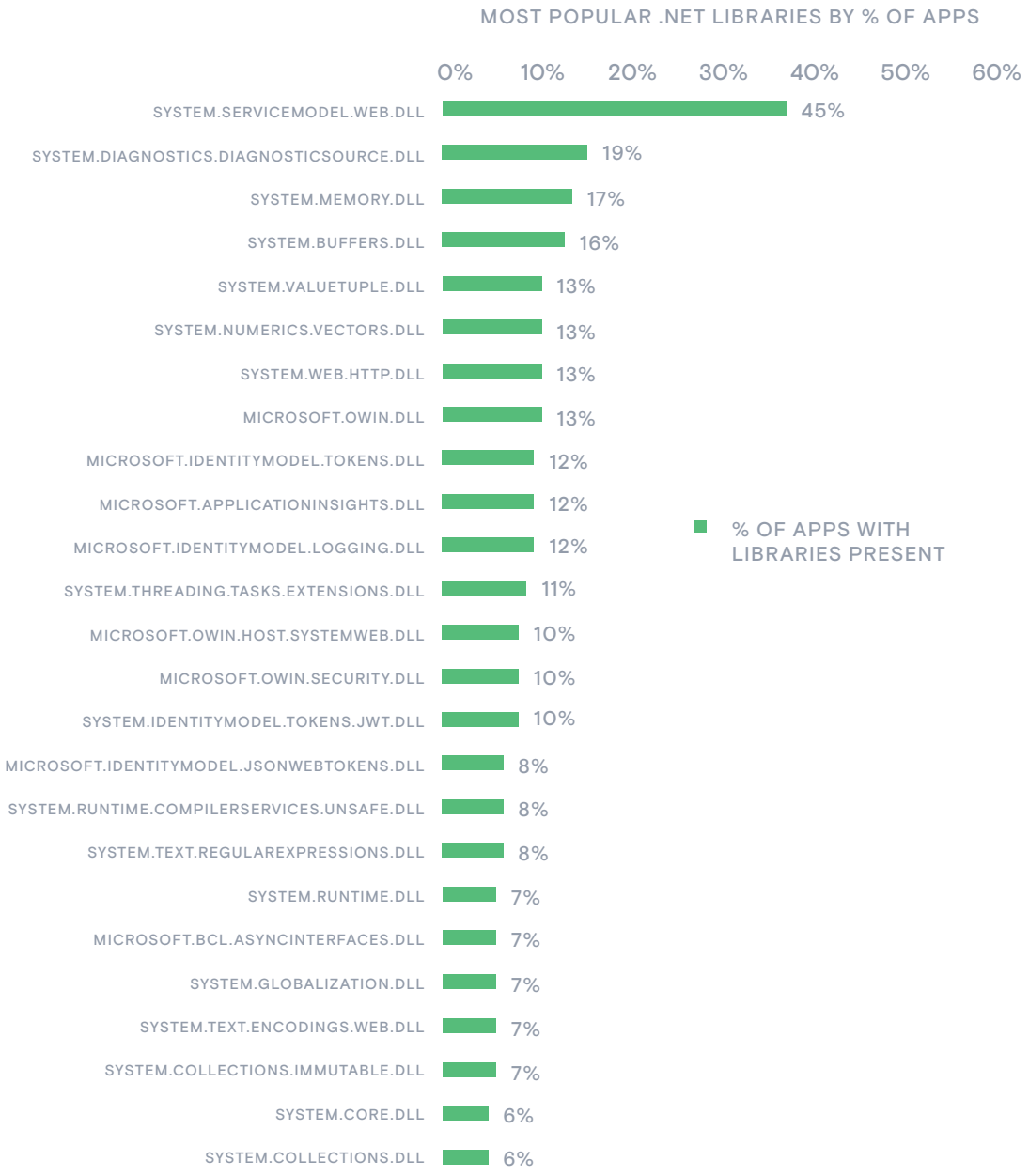
Percentage of Java applications containing the top 25 libraries.



The streamlined infrastructure supporting .NET development is readily apparent when one looks at library counts. While nearly 2 in 10 applications (18%) have 10 or more libraries, a solid majority (55%) include 2 or fewer (Figure 3). By far the most common library, System.ServiceModel.Web.dll, is present in 45% of applications. No other library is included in as many as 20% of applications (Figure 5), but all libraries present in more than 5% of .NET applications are controlled by Microsoft.

FIGURE 5

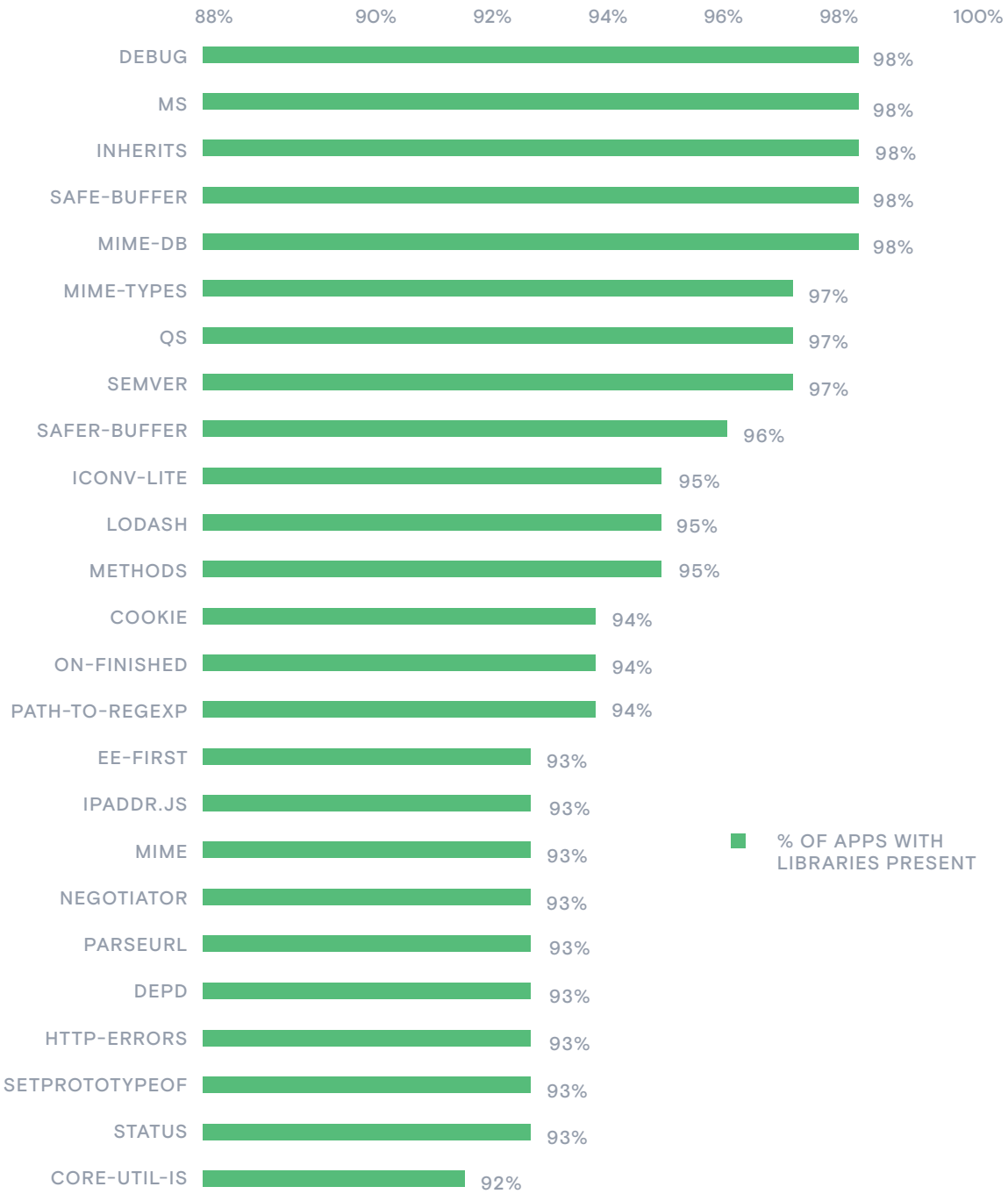
Percentage of .NET applications containing the top 25 libraries.



As noted, Node is structured in such a way that each library is smaller and more focused. As a result, 65% of Node applications have more than 500 libraries and 20% have more than 1,000 (Figure 3). The top 25 Node libraries are all present in 92% or more of Node applications (Figure 6). If any of these libraries were to be compromised, this would pose extraordinary risk to Node applications around the world (a dramatically higher risk than in the case of .NET applications).

FIGURE 6

Percentage of Node applications containing the top 25 libraries.



COMPLEXITY AS A CONTRIBUTOR TO RISK

By all accounts, the use of open-source libraries has exploded in the past several years.[6] For example, a recent study by GitHub found that 65% of all Java projects, 90% of .NET projects, and 95% of JavaScript projects (including Node) on that platform use open-source software.[7] But measuring open-source risk for a specific application is more complicated than simply counting libraries. Indeed, this entire study describes in great detail the fact that different libraries—and different parts of the same library—pose different levels of risk to an organization.

Yet while there is no direct correlation between the number of libraries and the amount of risk, the complexity that comes from a proliferation of libraries and multilayered dependency trees can increase risk. Even without cybersecurity considerations, organizations may benefit from deliberate efforts to declutter application code and practice basic hygiene on open-source libraries. The increasing focus on web applications as an attack vector for cyber criminals makes such hygiene even more important.

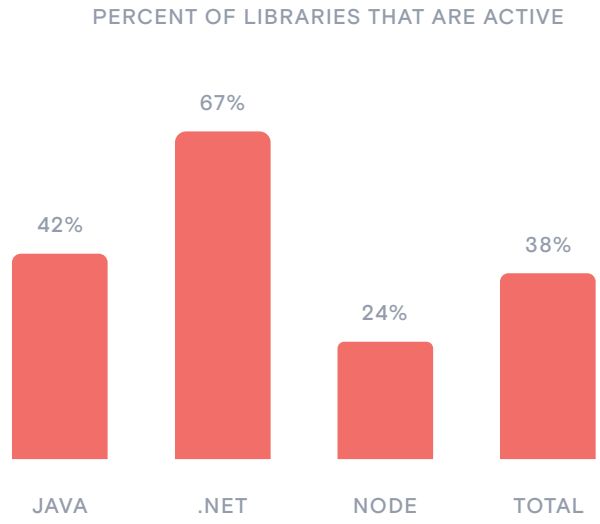
05 | Risk Layer 1: Active and Inactive Libraries

While the number of libraries is high, the percentage of those libraries that are active is the more important metric and represents the first layer of open-source risk. Overall, only 38% of libraries present in applications protected by Contrast OSS and Contrast Assess are active (Figure 7). This means that 62% of libraries found in applications are not used by the software in any way. Again, Node applications skew this average somewhat. More than three-quarters (76%) of Node libraries found in applications are inactive, while that number is 58% with Java and just 33% with .NET.

Why do applications contain so many libraries that are not used in any way? As described above, most inactive libraries in applications occur when multiple additional libraries are attached to an active library—but do not contribute to the functionality for which the library was selected. This can lead to multilayered dependency trees and increased complexity. Node packages in particular introduce many transitive dependencies. Another reason that libraries may be inactive is that later revisions to a piece of software might bypass libraries that were active in a prior version.

FIGURE 7

Percent of libraries active per application, by language.



ACTIVE AND INACTIVE LIBRARIES BY LANGUAGE

While the average Java application contains 125 libraries, 61% of Java applications have fewer than 50 active libraries (Figure 8). And while all the top 25 Java libraries are present in a majority of applications, the percent of applications where these libraries are active is much lower (Figure 9). Only 12 of the top 25 Java libraries are active in more than half of applications.

FIGURE 8

Percentage of applications by active library count, by language.

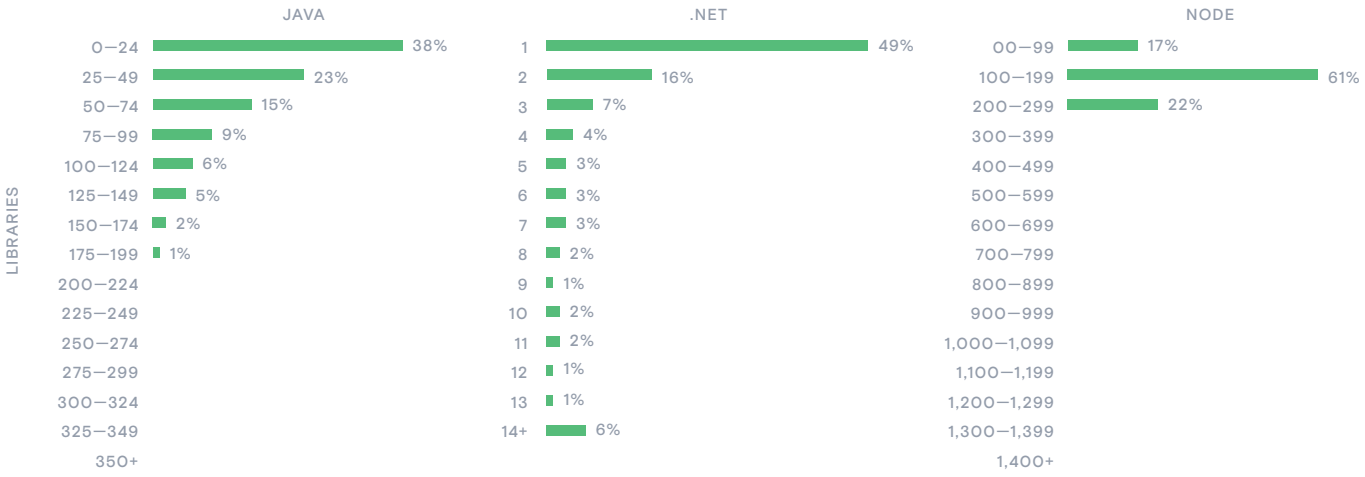
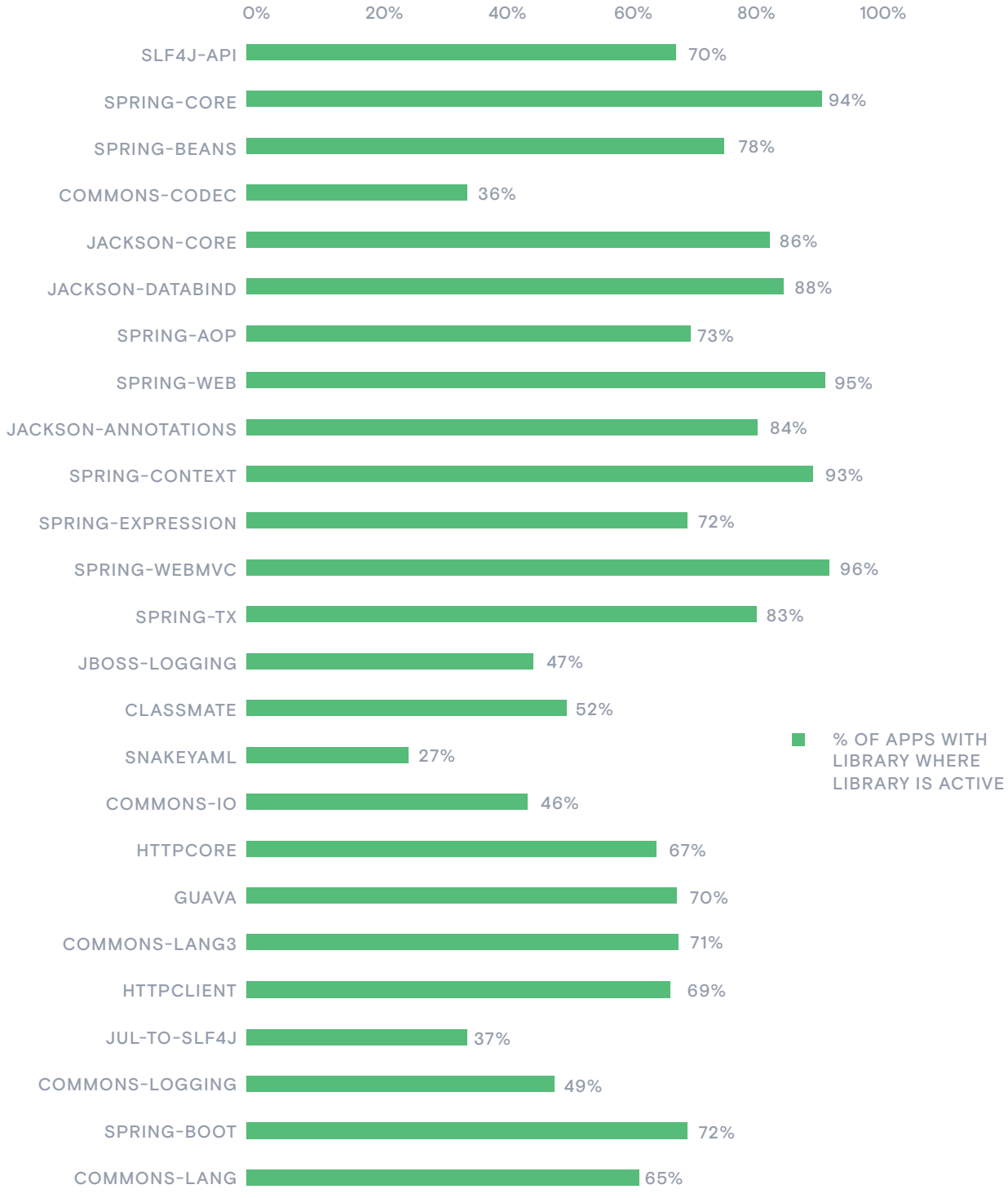


FIGURE 9

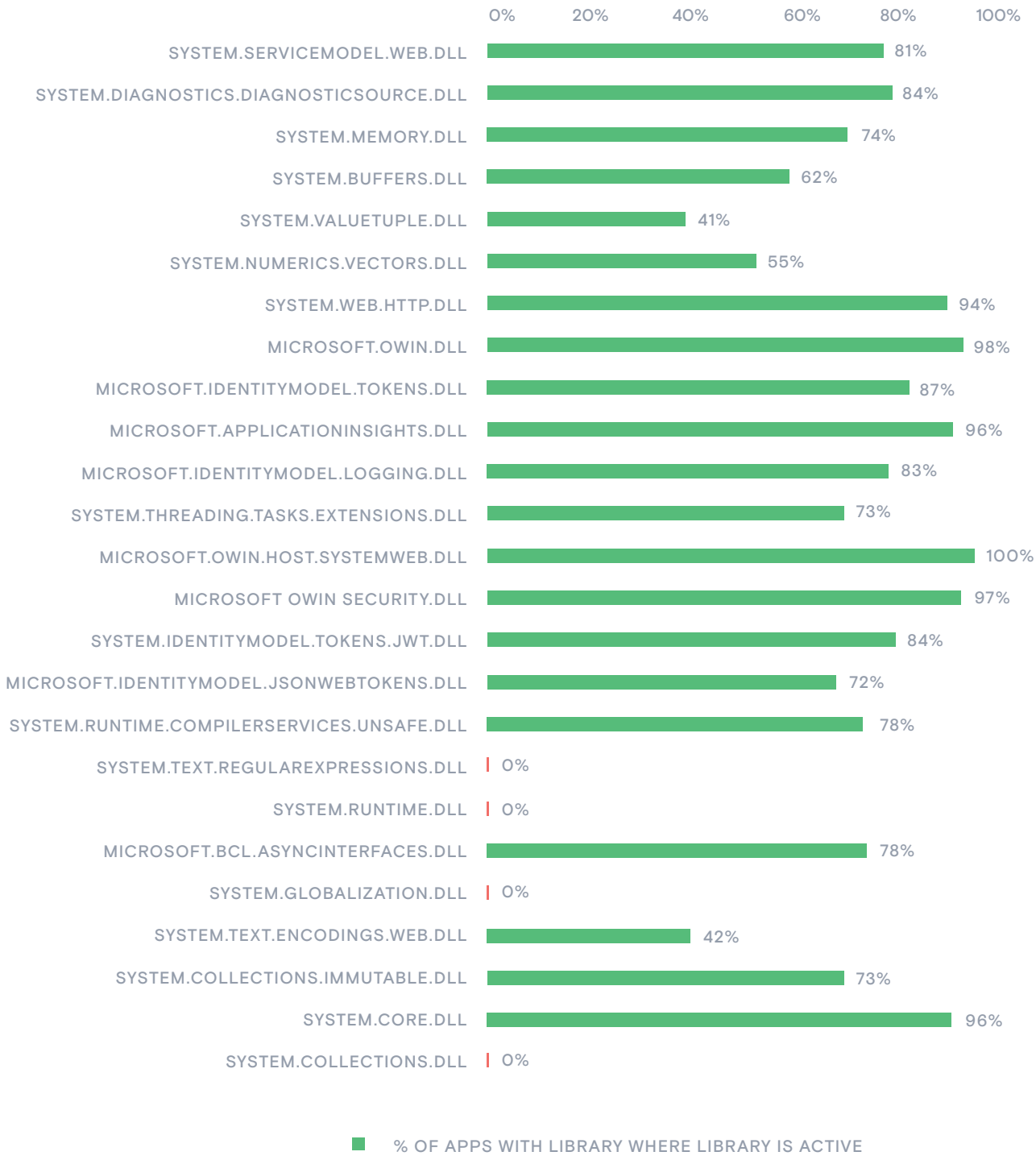
Percentage of Java applications with active libraries in the top 25, in descending popularity order.



Amazingly, 49% of .NET applications have just one active library (Figure 8). The most common library, System.ServiceModel.Web.dll, is active in 37% of applications (Figure 10). Beyond that, only one library is active in more than 15% of applications, and an additional five are active in more than 10%.

FIGURE 10

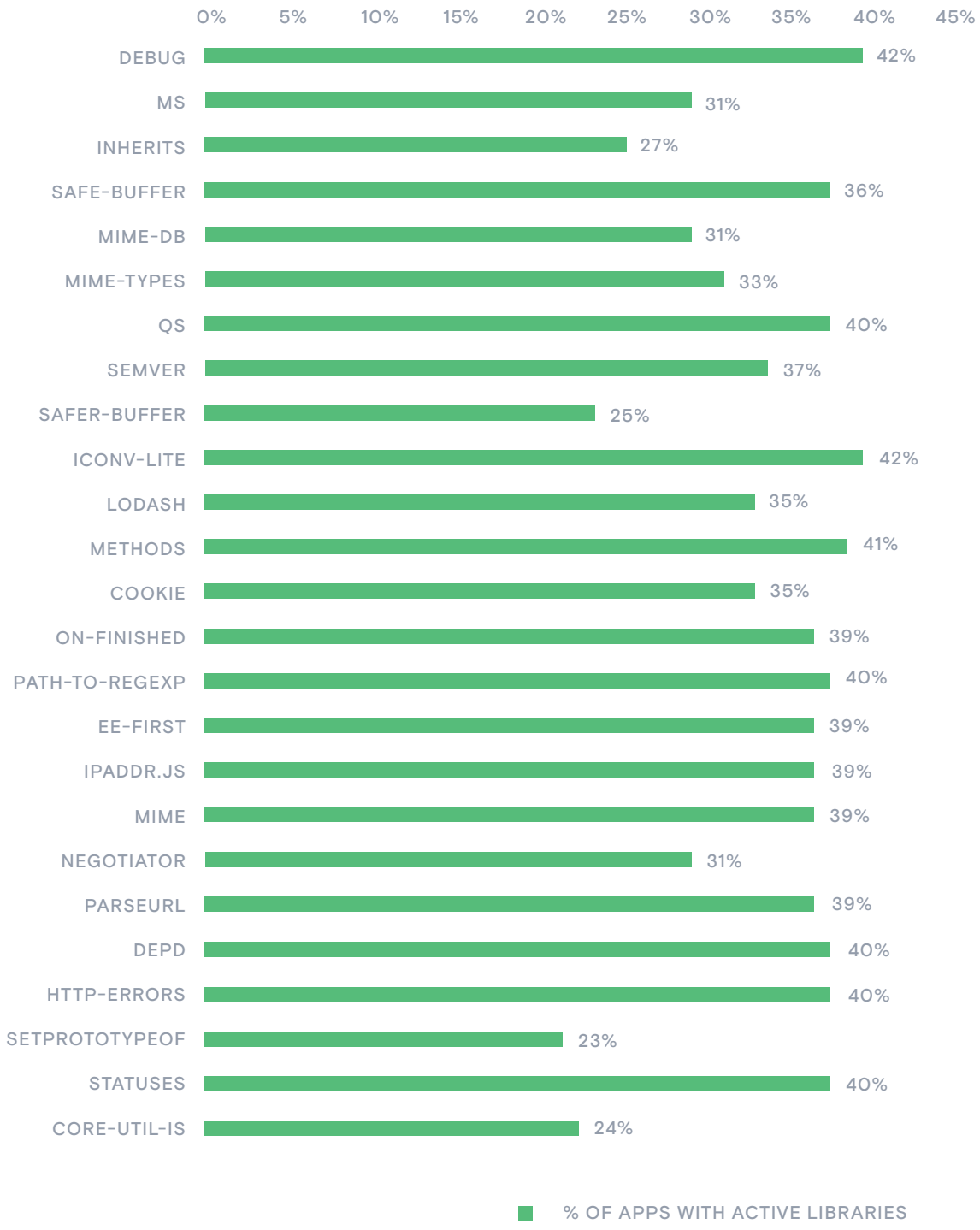
Percentage of .NET applications with active libraries in the top 25, in descending popularity order.



With Node applications, while the application count averages 537, none of the Node applications protected by Contrast OSS and Contrast Assess have more than 300 active libraries, and 78% have fewer than 200 (Figure 8). And while the top 25 libraries are present in more than 90% of applications, the most common active library is only present in 42% of applications (Figure 11). This reveals that many of the numerous Node libraries found in applications are not actually used.

FIGURE 11

Percentage of Node applications with active libraries in the top 25, in descending popularity order.



ACTIVE AND INACTIVE LIBRARIES: TWO KINDS OF RISK

Organizations face risk from both their active and their inactive libraries. The libraries actually used by the software can potentially have vulnerabilities that bring risk if they are not addressed. And while

vulnerabilities in inactive libraries pose no risk, companies can waste many hours of staff time remediating those vulnerabilities if they do not know which libraries are active. In addition to this operational inefficiency, fixing vulnerabilities that pose no risk can also delay action on vulnerabilities that can be exploited.

Another insight that can be gleaned from this data is that applications containing more libraries tend to have a lower percentage of those libraries that are active. Again, this could suggest that in these cases, legacy code needs to be cleaned up to reduce the total code surface area and reduce risk.

Of course, both of these efforts at library hygiene require visibility into which libraries are active and which are not.

05 | Risk Layer 2: Active and Inactive Library Classes

While a given library may be active in an application, only a very small part of that library is active in many cases. On average, across all languages, only 31% of classes in active libraries are invoked (Figure 12).

This state of affairs can be quantified by looking at library classes that are active in an application. Classes are logical collections of code within libraries that perform related tasks. Vulnerabilities that may exist in inactive classes in a library—even if the library itself is active—cannot be exploited successfully by cyber criminals.

The above equates to a dramatic revelation. Assuming that 80% of application code derives from third-party libraries,[8] this means 9.4% of application code is from active open-source library code.

The remaining 90+% is custom. Because of differences in the way they are structured, the number of classes varies widely depending on the language being used. On average, Java libraries contain 279 classes, .NET libraries contain 138 classes, and Node libraries contain just eight classes (Figure 13). But only 32% of Java classes, 67% of .NET classes, and an astounding 5% of classes in Node libraries are invoked by active libraries. Clearly, even in active libraries, much of the code is not used by an application—especially with Java and Node.

FIGURE 12

Percent of classes per active library, per application.

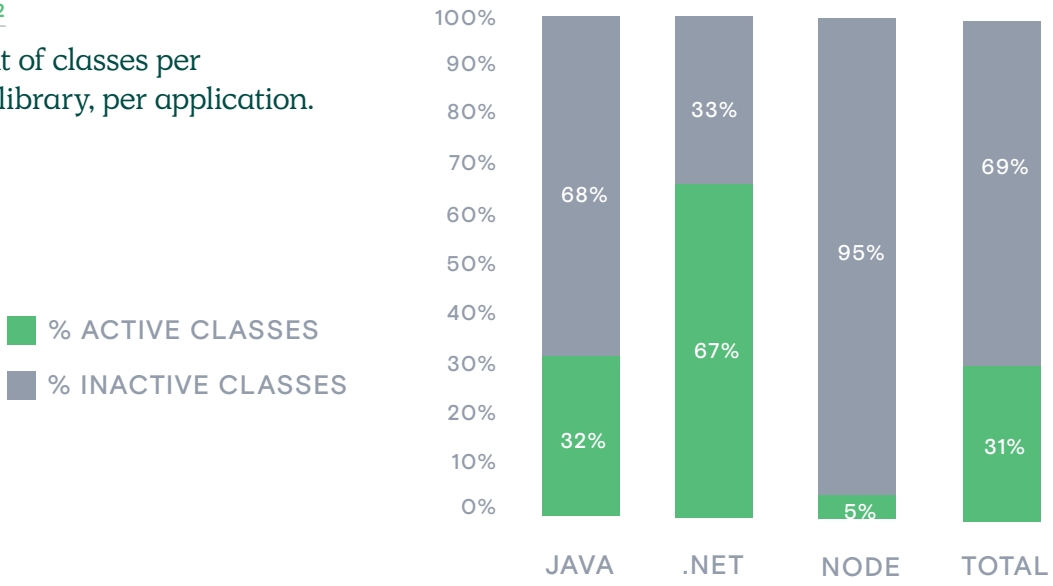
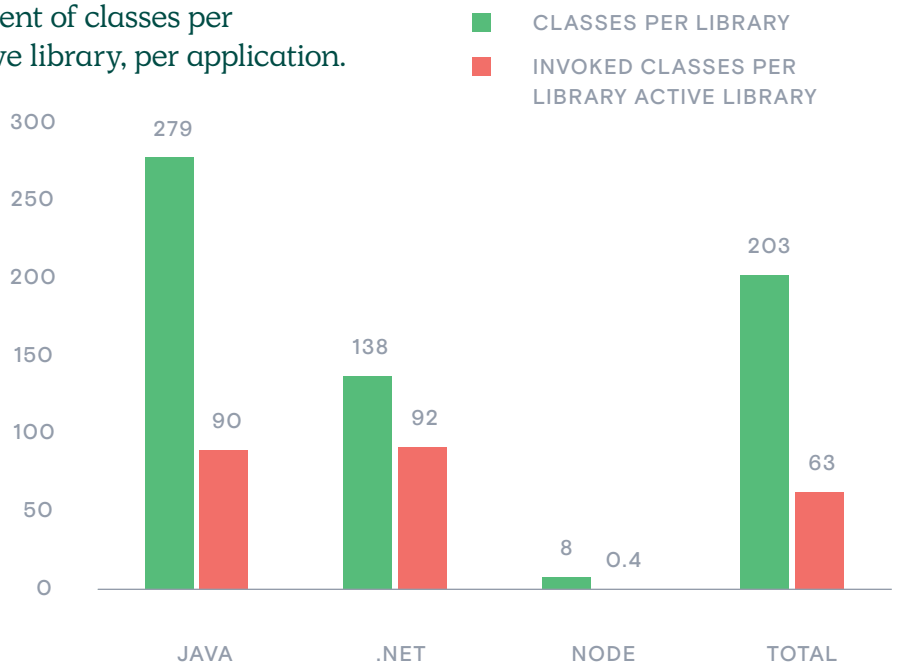


FIGURE 13

Percent of classes per active library, per application.



LIBRARY CLASSES BY LANGUAGE

For Java libraries, less than 30% of classes are active in a majority (53%) of libraries (Figure 14). Several of the top 25 Java libraries have 37% or 38% of their classes active, but others are in the single digits (Figure 15).

One piece of good news is that the above averages obscure the fact that 48% of .NET libraries and 68% of Node libraries have more than 90% of their classes active (Figure 14). However, in reality, the percentage of active classes varies widely depending on the specific library, as shown in Figures 15 and 16.

FIGURE 14

Percentage of active classes, by language.

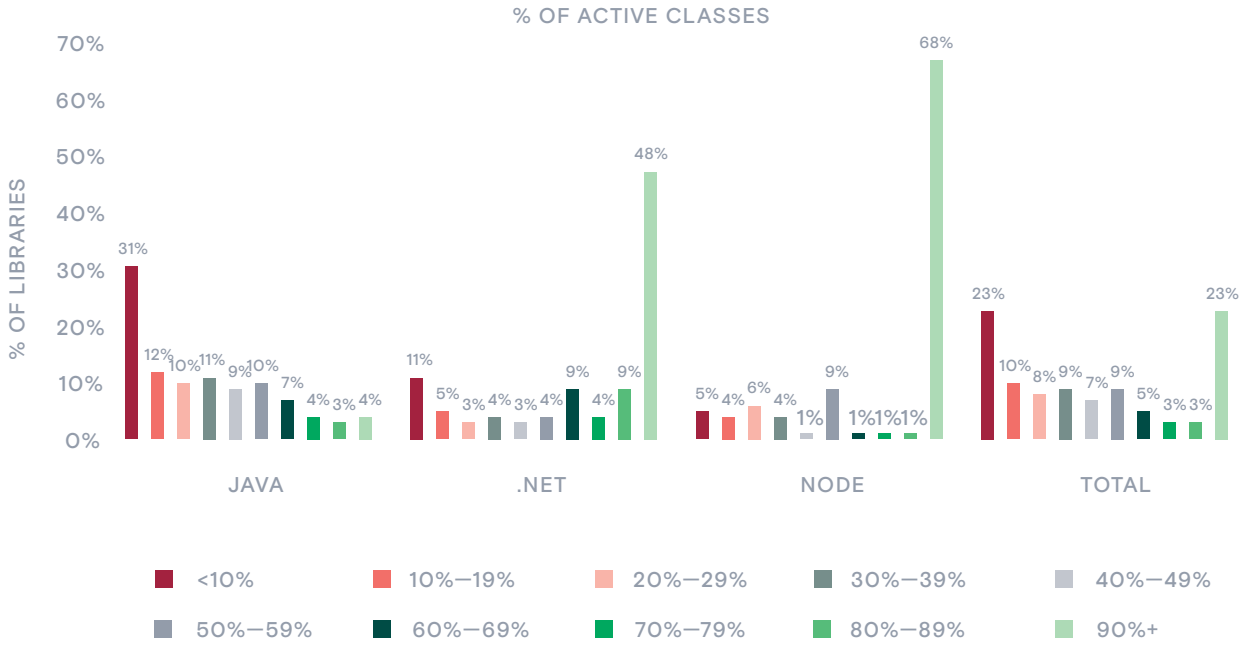


FIGURE 15

Percent of classes invoked by active libraries
for top 25 Java libraries, in descending popularity order.

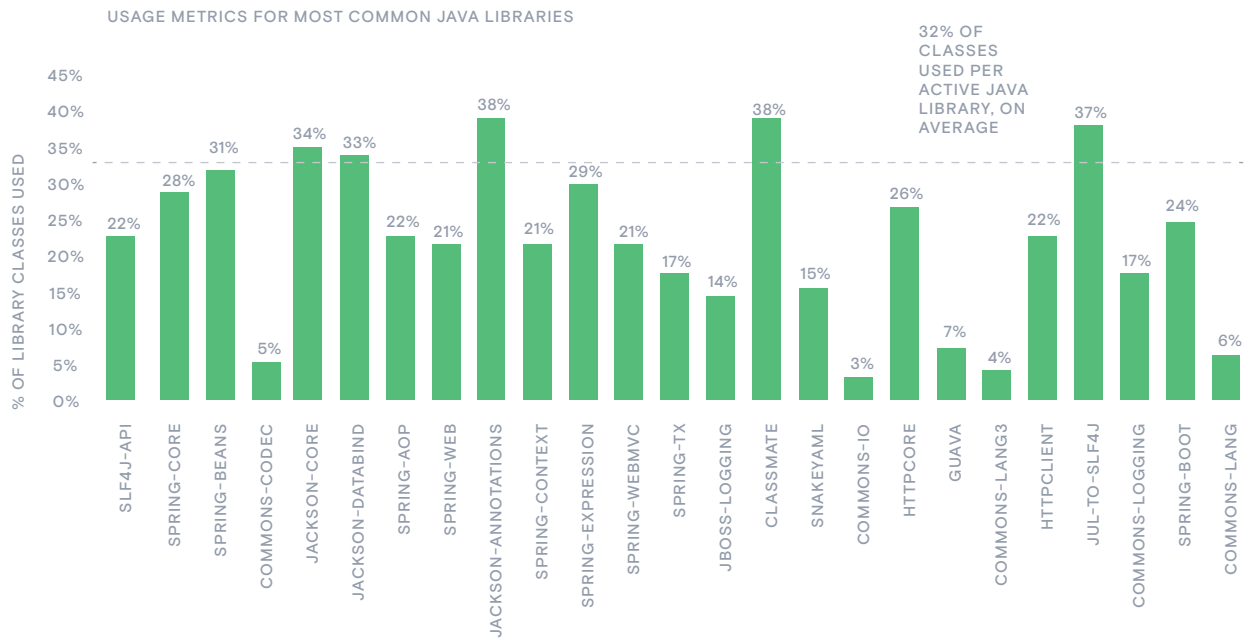


FIGURE 16

Usage metrics for most common .net libraries

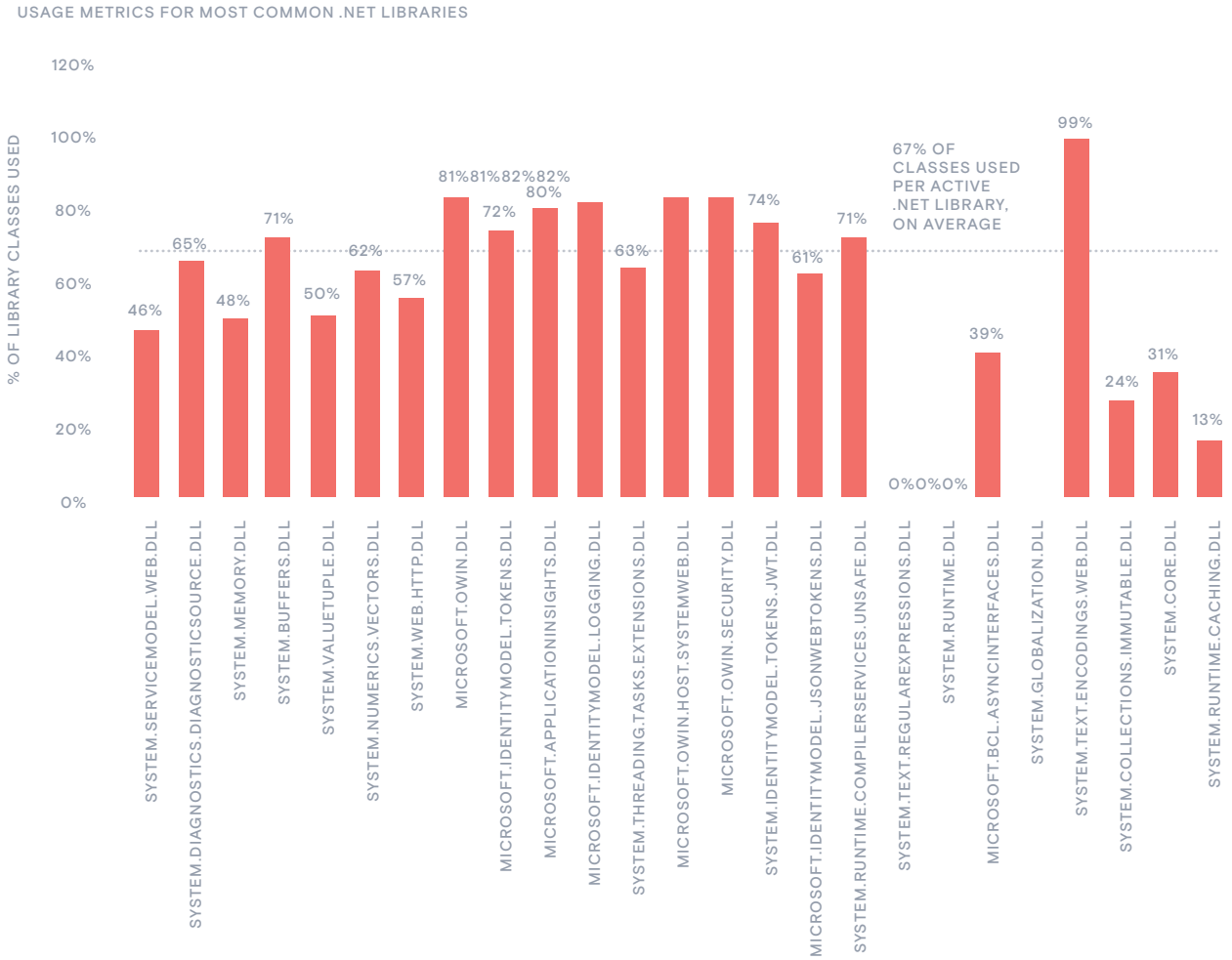
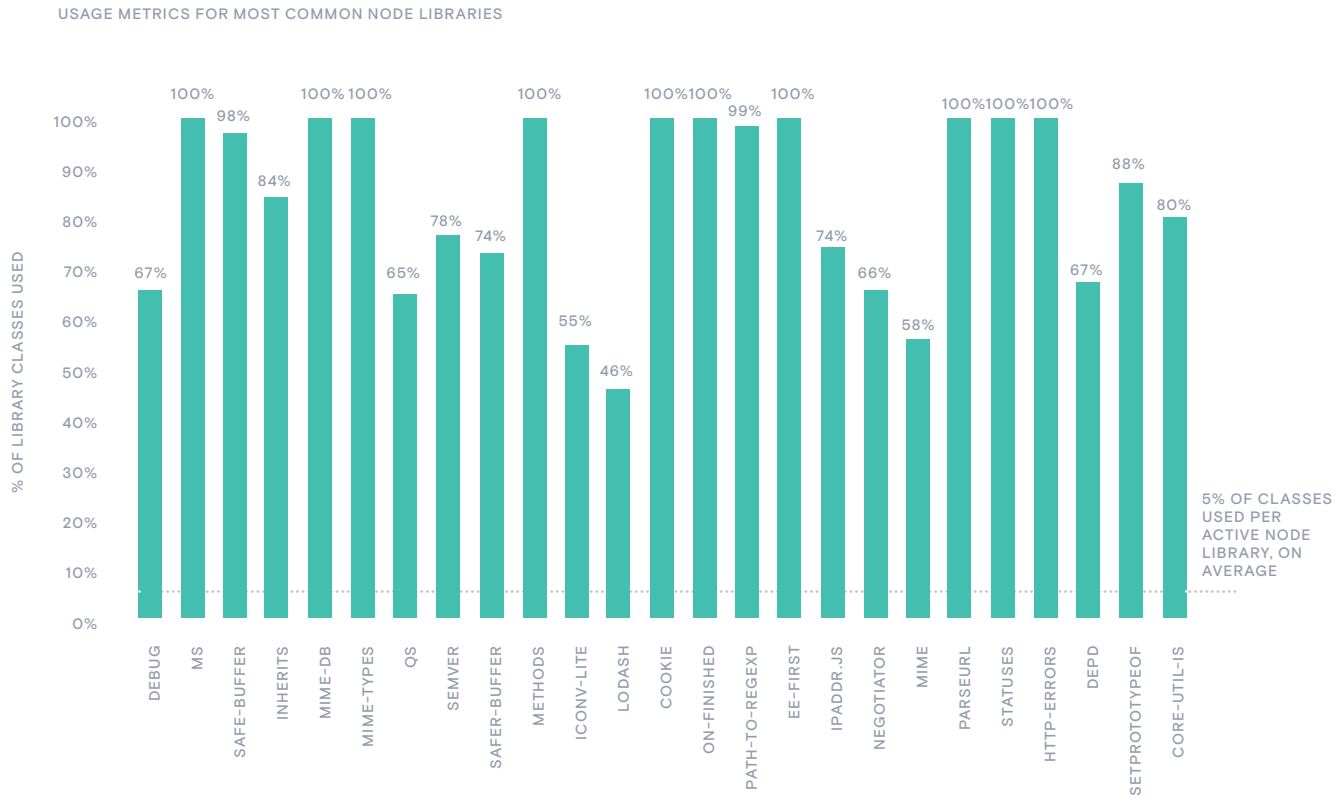


FIGURE 17

Percent of classes invoked by active libraries for top 25 Node libraries, in descending popularity order.



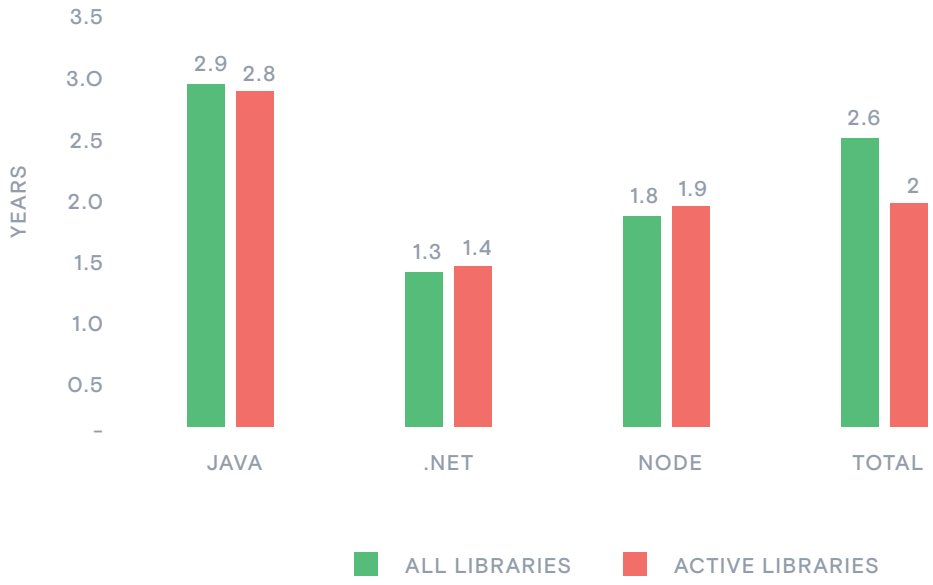
07 | Risk Layer 3: Library Age

As new vulnerabilities are discovered in libraries and added to the Common Vulnerabilities and Exposures (CVE) database, new versions of those libraries are released that remediate these issues. Ideally, organizations would immediately update the library in all applications, but there are reasons this is not advisable in some cases. Some libraries release new versions before adequate testing has been done, resulting in unstable code. In other instances, a library update might have downstream impacts on functionality that has nothing to do with the CVE being addressed. Notwithstanding, organizations are further behind on library updates than they should be.

One problem in compiling data on library age for this report is that each framework has a unique numbering system and frequency for new library releases. As a result, simply counting the number of versions that have been released since the version found in a specific application does not provide an “apples to apples” comparison across libraries. Instead, we opted to measure the chronological age of each library version—specifically, how many days ago a specific version was released.

FIGURE 18

Average years behind for libraries, by language.



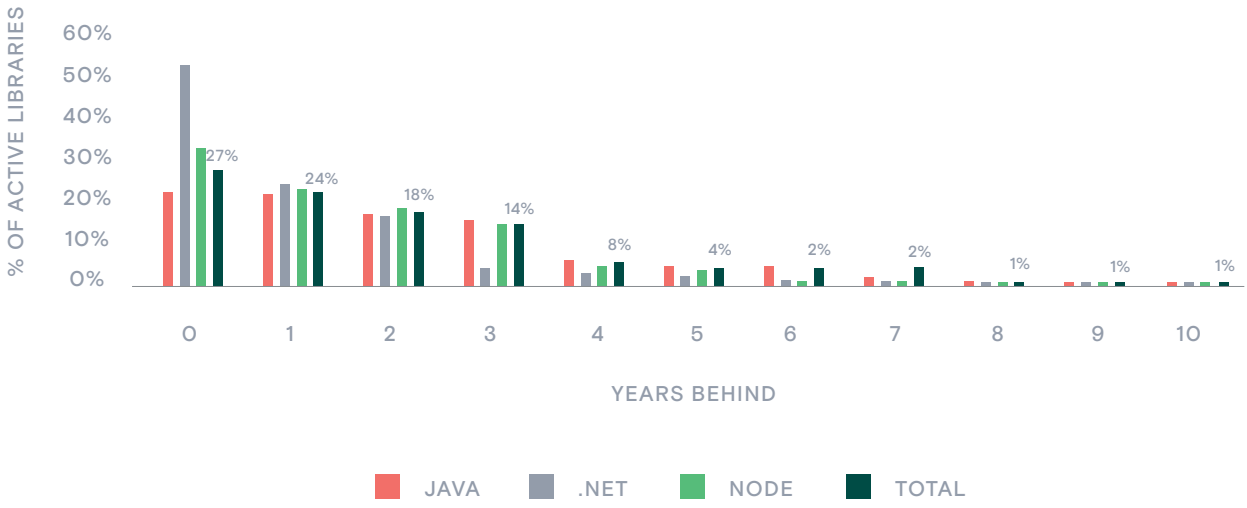
TYPICAL LIBRARIES ARE YEARS OUT OF DATE

Among all applications protected by Contrast OSS and Contrast Assess, the average library has not been updated in 937 days, approximately 2.6 years (Figure 18). Among active libraries, the news is only slightly better—892 days or 2.4 years. Further, 19% of libraries currently in use are more than three years old, with 6% more than five years old (Figure 19). Only 27% of active libraries are less than a year old.

The differences between languages are also clear in Figures 18 and 19. Java libraries are nearly three years old on average—2.9 years for all libraries and 2.8 years for active ones. And while 45% of Java libraries are less than two years old, 37% are more than three years old. .NET libraries, on the other hand, are barely 16 months old on average, and just 21% are more than one year old and 5% are more than two years old. Node libraries are in between, with the average library being just under two years behind.

FIGURE 19

Percent of active libraries by number of years behind, by language.



OLDER LIBRARIES INCREASE RISK AND REDUCE AGILITY

Keeping libraries up to date is a part of the basic hygiene that is critical for the continued health of an application. This is especially important with libraries for which a high percentage of classes is being used, and are therefore deeply integrated into an application. Needless to say, visibility into the age of each library and the percentage of classes in use is essential to conduct this basic maintenance effectively.

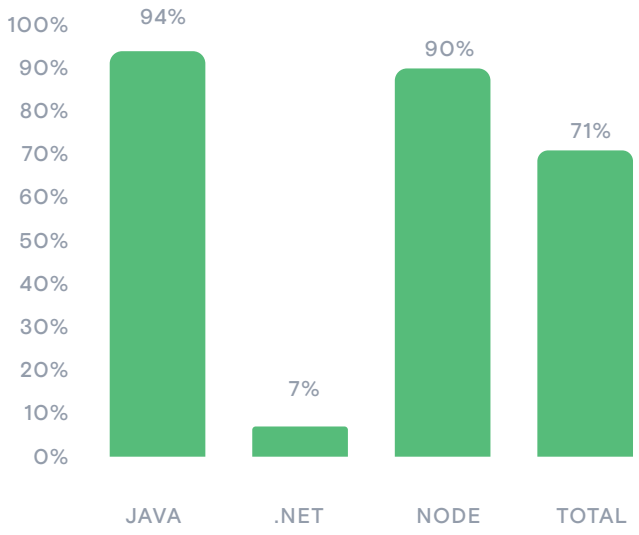
Failure to keep libraries updated over time not only increases risk to an organization but also makes library updates much more difficult and time-consuming when they are finally done. When a library stays dormant in an application for multiple years, any new vulnerability is difficult to fix because so much code has been built over it. Updating a years-old version of a library will require significant work by the development team.

08 | Risk Layer 4: Vulnerabilities in Libraries

Preventing and remediating software vulnerabilities is the whole point of application security, and it is important for organizations to have a picture of vulnerabilities present in their third-party libraries. Among applications protected by Contrast OSS and Contrast Assess, an astounding 94% of Java applications and 90% of Node applications have at least one CVE (Figure 20). The news is especially bad for Java, where 45% of applications have a Critical CVE.

FIGURE 20

Percentage of applications with at least one CVE, by language.



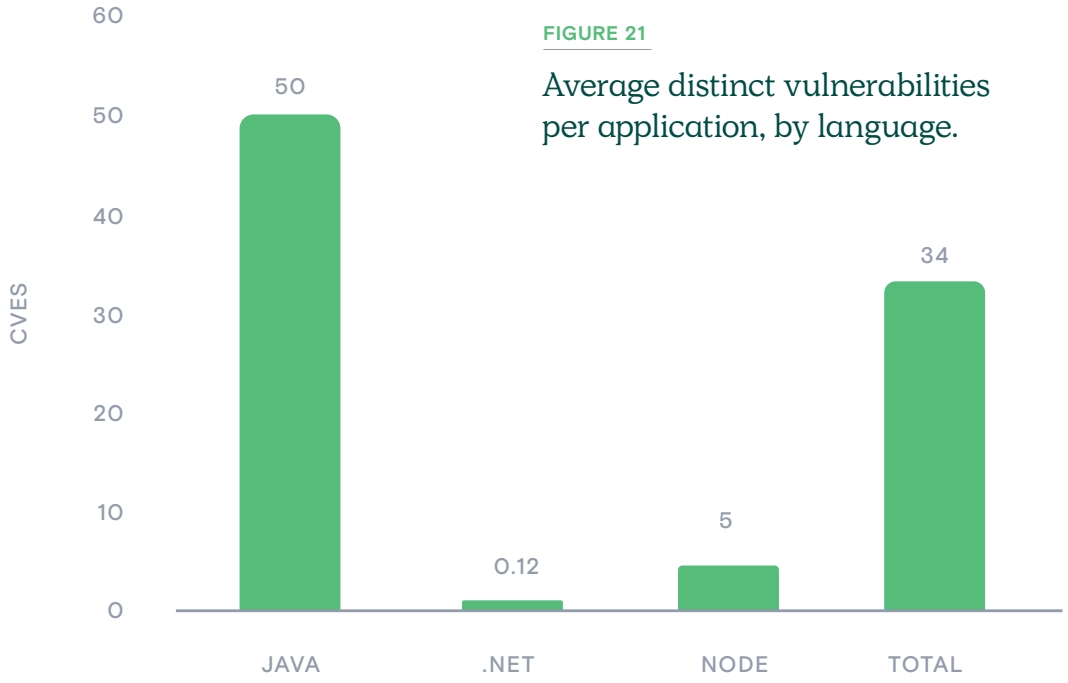


FIGURE 21

Average distinct vulnerabilities per application, by language.

Overall, the average application has 34 CVEs. However, this number is misleading because Java applications have 50 vulnerabilities on average (Figure 21). There are just five CVEs per Node application, and just one vulnerability for every eight .NET applications.

Drilling down to individual active libraries, the number of vulnerabilities also varies greatly by programming language. Nearly 1 in 12 active Java libraries (8%) contain a CVE, while just 2% of active Node libraries and 1% of active .NET libraries have one (Figure 22).

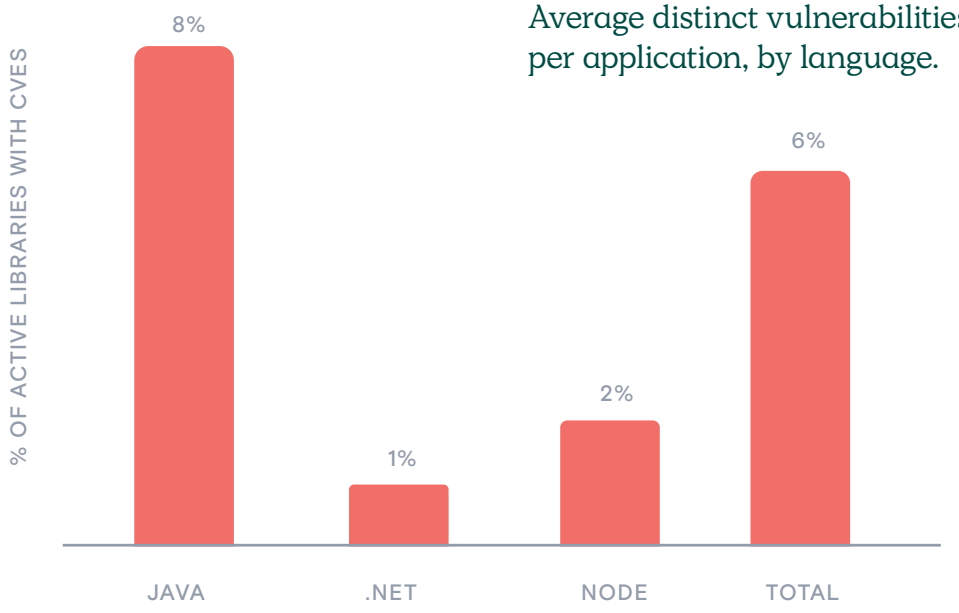


FIGURE 22

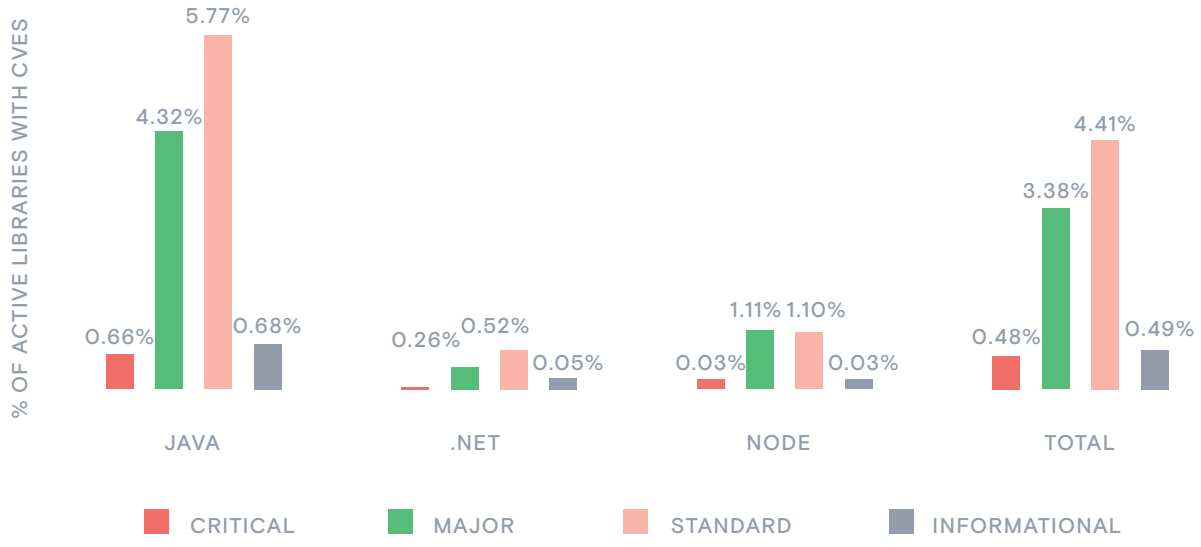
Average distinct vulnerabilities per application, by language.

BY SEVERITY

Some vulnerabilities obviously present more risk than others, with Critical and Major CVEs much more risky than Standard and Informational ones. Just under 5% of active Java libraries have Critical or Major CVEs—more than half of the total CVEs for the language (Figure 23). Critical or major CVEs are present in under 1% of active .NET libraries and just over 1% of active Node libraries.

FIGURE 23

Percentage of active libraries with CVEs, by language and severity.

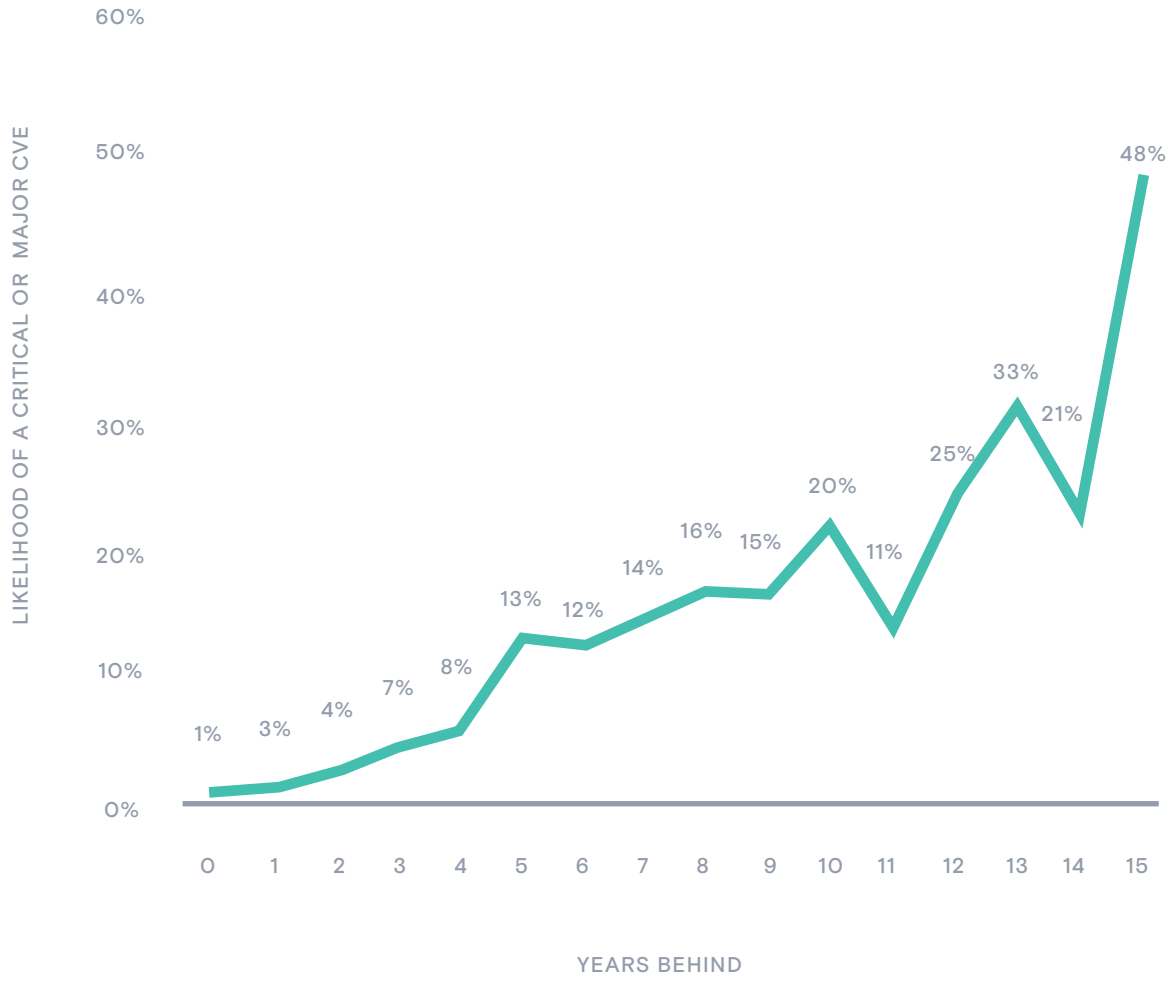


BY LIBRARY AGE

It is obvious that less up-to-date libraries contain more CVEs on average than newer versions. What may be surprising is the speed at which risk increases. Looking at Java libraries in particular, the odds of a Critical or Major CVE being present in a library increases from 1% to 3% to 4% to 7% as library age progresses to three years old, spiking at 48% when the library is 15 years old (Figure 24). Put another way, updating an old library is a quick way to significantly reduce organizational risk. It goes without saying that while CVEs in older libraries get resolved in later versions of the library, they remain unresolved in the older version where the CVE was found.

FIGURE 24

Percentage of Java libraries with critical and major CVEs, by library age.



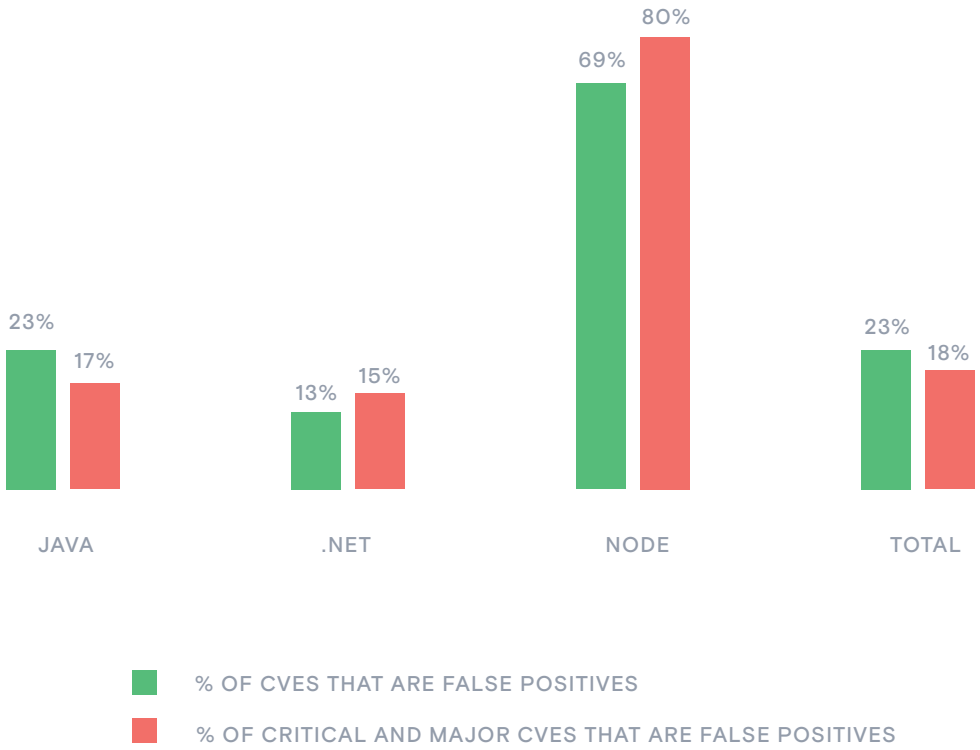
FALSE-POSITIVE RATES FOR TRADITIONAL SCA TOOLS

Owners of applications protected by Contrast OSS can easily determine which CVEs are present in inactive libraries and library classes, and therefore pose no risk to the organization. As noted, traditional SCA tools that simply return a list of CVEs present in an application produce a false positive every time they list a CVE in an unused part of the application.

The aggregate data shows that 17% of Critical and Major CVEs in Java applications, 15% in .NET applications, and 80% in Node applications are in inactive libraries or classes (Figure 25). Without this observability, organizations would spend significant time remediating vulnerabilities that introduce zero risk. For organizations with hundreds of applications, this can quickly tally into thousands of hours annually—which translates into development delays.

FIGURE 25

SCA false-positive rates for Java, .NET, and Node applications.



RESOLVING RISKY VULNERABILITIES, AND BEING READY FOR NEW ATTACKS

Taking care of vulnerabilities is what application security is all about, but not every vulnerability is created equal. Just as Critical and Major vulnerabilities rank ahead of other ones in terms of risk, CVEs in active classes that are a part of active libraries are the only ones that present risk to an organization. Again, full observability into active libraries and classes, library age, and unresolved CVEs is important to reduce risk and maximize efficiency.

That said, it should be noted that all CVEs that are logged are discovered by a very small number of volunteer security researchers—a group that is badly outnumbered by cyber criminals. In actuality, it is very likely that there are many more undiscovered vulnerabilities than discovered ones. While this report focuses on the risks we know about, it is important to remember that runtime protection is critical to prevent exploitation of unknown vulnerabilities.

09 | Risk Layer 4: Licensing Risk

Although open-source code is free to use, it is not always free to use without restriction. These restrictions are determined by the type of license associated with the library. Licenses fall into two categories: permissive and copyleft. Permissive licenses place no restriction on the use of the software and include Apache, the most common Java and .NET license, and MIT, the most common Node license (see Figure 26).

Copyleft licenses, on the other hand, claim that the code is copyrighted and can only be used if the resulting software product is released as open source. This, of course, introduces significant operational risk for organizations. Including even one library that uses a copyleft license in a library's dependency tree technically renders the entire library subject to copyleft restrictions, and libraries can potentially be mislabeled in this regard.[9]

Versions of the General Public License (GPL) are the most popular copyleft licenses, and Contrast Labs rates all versions of GPL as high risk. Other copyleft licenses bring moderate risk according to Contrast Labs, including Lesser GPL (LGPL), Mozilla Public License (MPL), and Eclipse Public License (EPL).

Because of this risk, it is concerning that 69% of Java applications and 33% of Node applications have at least one high-risk license (Figure 27a and 27b). In addition, 95% of Java applications and 70% of Node applications have at least one license of unknown or variable risk. One specific copyleft license, GPL 2.0, is present in 35% of all applications. Although the .NET language tightly controls its licenses and its applications have no high- or moderate-risk licenses, 99% of organizations represented in the dataset have at least one application containing a high-risk license.

High-risk licenses are only used with 2% of Java libraries and a tiny fraction of 1% of Node libraries. It is entirely possible that the libraries associated with these licenses are inactive, and organizations may not even be aware that these licenses are in their applications. Yet, high-risk licenses pose significant risk, and this is yet another reason that full observability of the open-source software environment is critical for all organizations.

FIGURE 26

Percentage of applications and libraries with the top 10 open-source licenses.

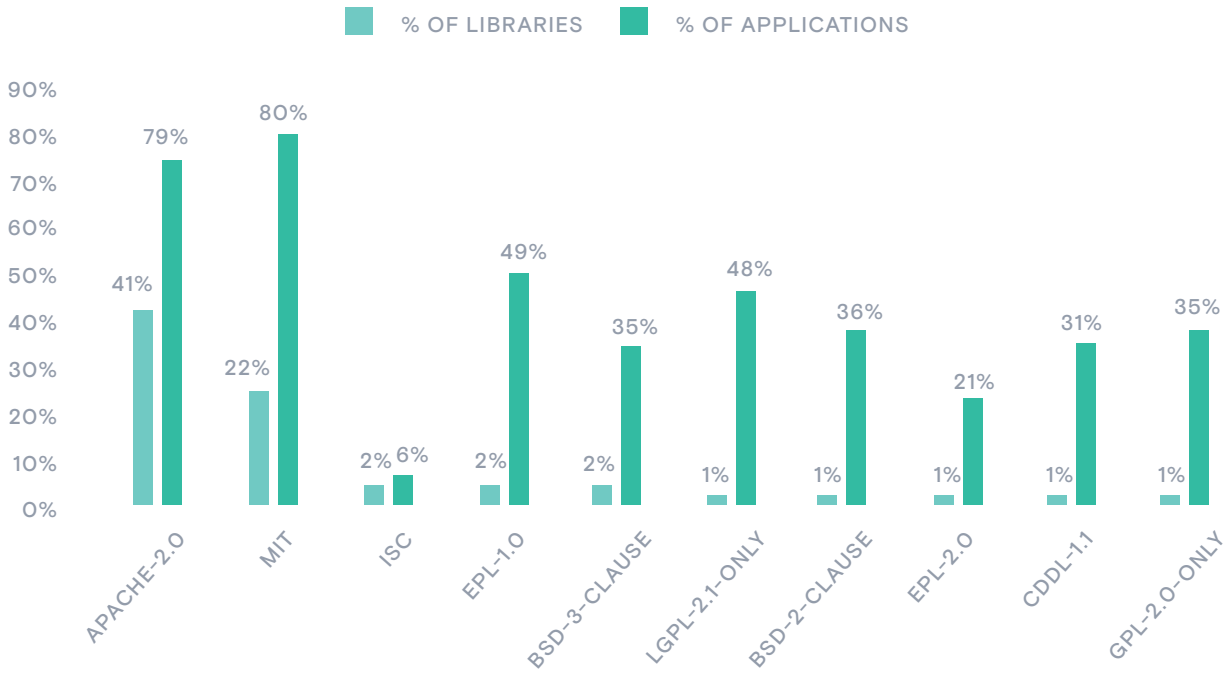
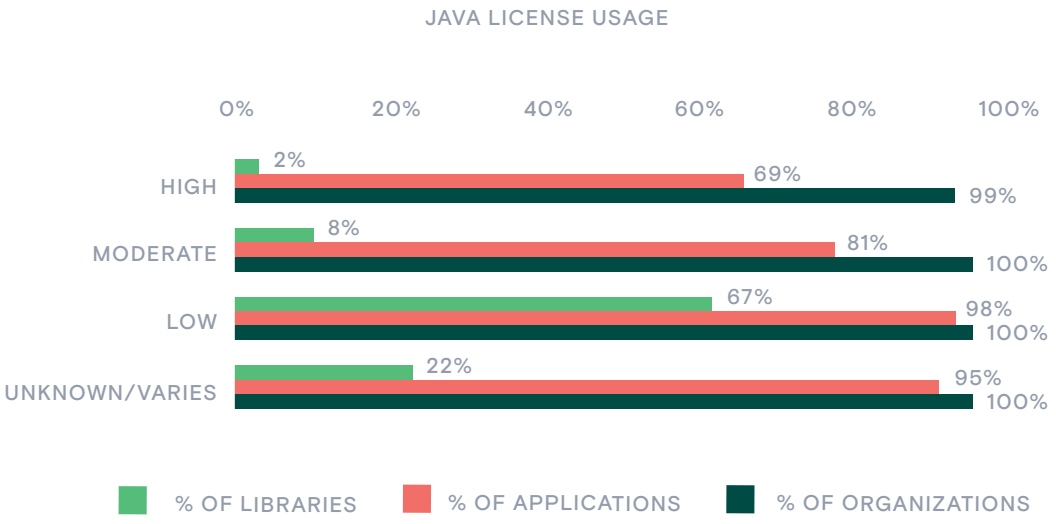


FIGURE 27A

SCA false-positive rates for Java, .NET, and Node applications.



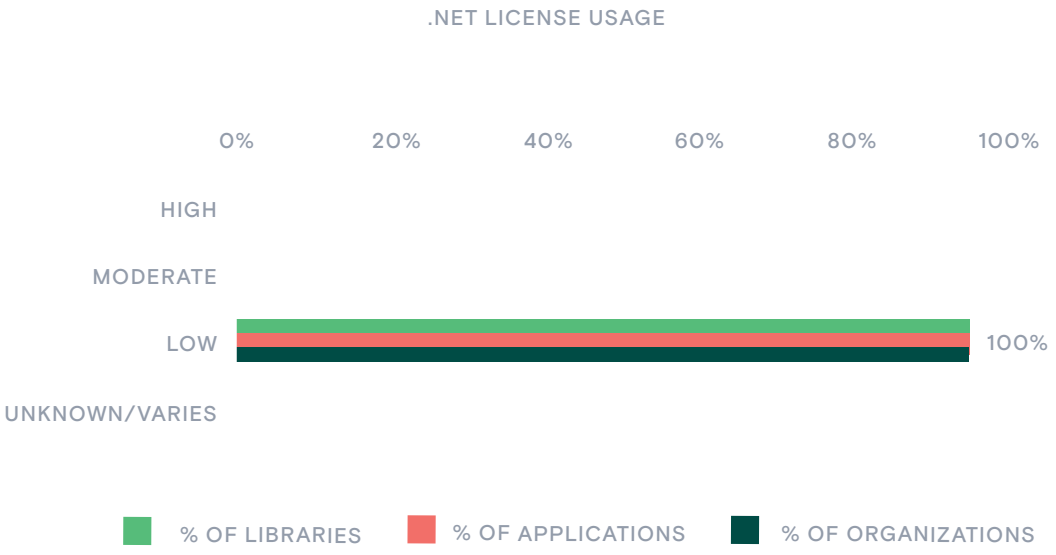
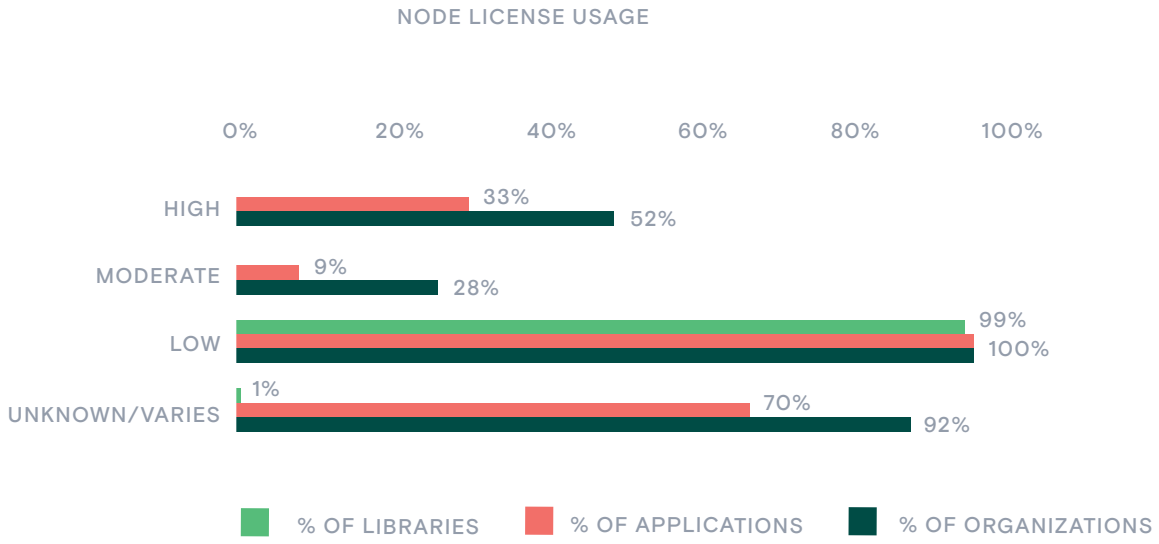
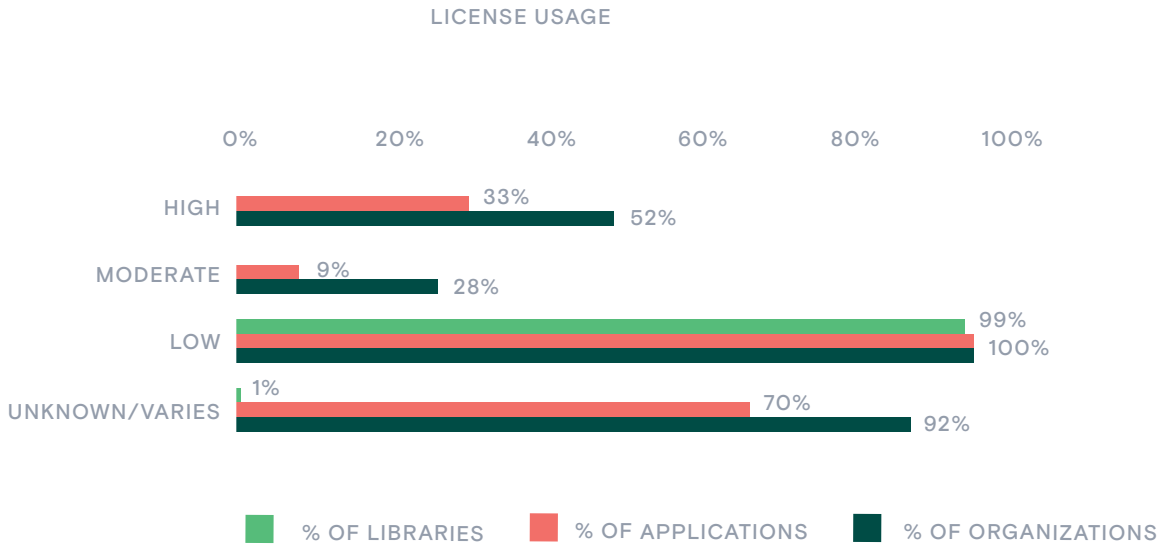


FIGURE 27B

License usage by risk level and language.





10 | Conclusion

The 2021 State of Open-source Security Report leverages data from real applications to identify trends in the quest to secure the libraries that form an integral part of most applications today. The report highlights five layers of risk faced by every organization that develops software: active and inactive libraries, active and inactive classes, library age, vulnerabilities in libraries, and licensing risk.

An increasingly efficient software factory is the engine behind the ongoing digital transformation that is remaking how companies operate and interact with their customers. This long-standing trend accelerated its pace during the COVID-19 pandemic. As many as 79% of executives who responded to one survey said that the pandemic had resulted in increased budgets for digital transformation.[10] Another survey found that consumers are three times more likely to say that 80% of their customer interactions are digital in nature than before the coronavirus.[11]

From an application security perspective, open-source libraries are one of the four elements of the software supply chain, each of which must receive equal and critical priority:

- What you write: Custom code developed in-house
- What you build with: Software development tools
- What you buy: Off-the-shelf Software-as-a-Service (SaaS) applications
- What you use: Third-party libraries

Contrast SECURITY

FIGURE 28

There are four primary components to the "assembly line" in the software factory.

WHAT YOU WRITE[12]

- 60% release code multiple times per day; 80% do so multiple times per week
- 79% still under pressure for more speed
- 55% skip security processes to meet SDLC deadlines
- Less than 50% of application security integrated with CI/CD tools

WHAT YOU BUILD WITH

- Developers have access to literally 1,000+ software development tools
- Work-from-home environments create greater security risks for thousands of pieces of software running with high privilege

WHAT YOU BUY

- SaaS market to grow 25% by 2022[13]
- 70% indicate "uninformed or misleading claims about security" in a SaaS solution were cause of dissatisfaction[14]
- 95% of businesses host sensitive data in SaaS solutions[15]

WHAT YOU USE

- 90% of applications rely on third-party libraries that comprise up to 70% of code[16]
- Applications on GitHub have an average of 200 dependencies[17]
- 73% of applications have a vulnerability traceable to third-party code

If any of these elements is missing from an organization's application security strategy, the other elements are weakened, and risk is increased. When it comes to open-source libraries, they inject code into the software factory and introduce significant risk. The fact that so many applications have a library with a high-risk license attests to the fact that too many organizations have an incomplete view of what libraries exist in their applications. This means that they are unable to provide the protection that those applications need.

KEY TAKEAWAYS

In this context, it is important that organizations take a holistic, methodical approach to open-source security. Factors they should consider include:

- Set comprehensive policies for libraries, frameworks, and licensing. Defining and enforcing limitations on the components allowed in an application can prevent libraries with GPL licenses or outdated libraries from being added. Likewise, policies for updating existing libraries can decrease the odds of vulnerabilities and save extra work in the future.

- Establish continuous observability. As we have said repeatedly, full visibility into which libraries and classes are active, how old they are, what CVEs they hold, and what licenses they require is critical for prioritization of remediation and reduction of risk. Since only active libraries and classes pose risk, this knowledge significantly narrows the scale of needed remediation.

- Embed controls in continuous integration/continuous deployment (CI/CD) processes. This can keep risky libraries and licenses from entering an application inadvertently by automating policy enforcement.

As the economy and public infrastructure become increasingly reliant on software, applications are an increasingly attractive target for cyber criminals. For the components of that software that come from open-source libraries, it is critical that organizations have the detailed data they need to make their software both secure and functional. This level of visibility and control is only available with tools from Contrast Security. Organizations that leverage these tools are increasing the efficiency of their development efforts while making them more secure.

Contributors

JEFF WILLIAMS

CTO AND CO-FOUNDER,
CONTRAST SECURITY

Jeff brings more than 20 years of security leadership experience as Co-Founder and Chief Technology Officer of Contrast. Previously, Jeff was Co-Founder and Chief Executive Officer of Aspect Security, a successful and innovative application security consulting company acquired by Ernst & Young. Jeff is also a founder and major contributor to OWASP, where he served as Global Chairman for eight years and created the OWASP Top 10, OWASP Enterprise Security API, OWASP Application Security Verification Standard, XSS Prevention Cheat Sheet, and many other widely adopted free and open projects. Jeff has a BA from the University of Virginia, an MA from George Mason, and a JD from Georgetown.

DAVID LINDNER

CHIEF INFORMATION
SECURITY OFFICER,
CONTRAST SECURITY

David is an experienced application security professional with over 20 years in cybersecurity. In addition to serving as the chief information security officer, David leads the Contrast Labs team that is focused on analyzing threat intelligence to help enterprise clients develop more proactive approaches to their application security programs. Throughout his career, David has worked within multiple disciplines in the security field—from application development, to network architecture design and support, to IT security and consulting, to security training, to application security. Over the past decade, David has specialized in all things related to mobile applications and securing them. He has worked with many clients across industry sectors, including financial, government, automobile, healthcare, and retail. David is an active participant in numerous bug bounty programs.

Contrast SECURITY

BRIAN GLAS

ASSISTANT PROFESSOR OF COMPUTER SCIENCE, UNION UNIVERSITY

Brian possesses nearly 20 years of experience in various roles in IT and over a decade in application development and security. In addition to teaching a full load of classes at Union University, Brian serves as a part-time management consultant and advisor for Contrast Labs. He worked on the Trustworthy Computing team at Microsoft and served as a project lead and active contributor for SAMM v1.1-2.0 and OWASP Top 10 2017. He is a popular speaker at numerous conferences and online events, having presented at InfoSec World, Cloud Security World, and numerous OWASP conferences and meetings. Brian is also an author of various papers and is currently researching writing a book on application security. He holds a long list of cybersecurity and IT certifications as well as a master in business administration and bachelors in computer science from Union University.

KATHARINE WATSON

SR. DATA ANALYST AND DATA SCIENTIST, CONTRAST SECURITY

Katharine is a driving force in developing and building data analytics frameworks for Contrast—including Contrast Labs—and turning data into actionable narratives and insights for internal and external customers. Katharine worked as an analyst, consultant, and project manager in both private and nonprofit organizations. Before launching a career in data science, Katharine worked for three years as a mathematics teacher in the Teach for America program. Katharine holds undergraduate and graduate degrees from The Johns Hopkins University.

PATRICK SPENCER, PH.D.

EDITOR IN CHIEF,
INSIDE APPSEC PODCAST

HEAD OF CONTENT AND
PR/COMMUNICATIONS,
CONTRAST SECURITY

Patrick founded and serves as the editor in chief for the Inside Appsec podcast and leads the content marketing and PR/communications team at Contrast. He has more than a decade and a half of experience in various senior marketing and research roles within the cybersecurity sector and is the recipient of numerous corporate and industry awards. After leaving the corporate world to start his own agency, Patrick joined Fortinet to lead content marketing and research. His many duties included serving as the editor in chief for The CISO Collective. Patrick's roots in cybersecurity go back to Symantec, where he spent nearly a decade in senior marketing roles of increasing scope and responsibility. While at Symantec, Patrick served as the editor in chief for CIO Digest, an award-winning digital and print publication containing strategies and insights for the technology executive.

MARK MULLINS

FOUNDER AND PRINCIPAL,
MHM CONTENTSOURCE

MHM ContentSource specializes in marketing research and writing projects for clients across the technology sector. Mark has 15 years of experience in research and content marketing across the technology sector, as both an employee and a consultant. He has authored numerous research reports, white papers, and magazine features and produced dozens of marketing videos and a podcast series. His work has been published by leading technology brands such as Symantec, LivePerson, PRO Unlimited, Finastra, Fortinet, Lastline, and Contrast Security, among others.

Contrast SECURITY

PAULINE LOGAN

PRODUCT MANAGER
CONTRAST SECURITY

Pauline oversees the product management strategy and execution for Contrast OSS. She spent over a decade developing applications in Java, .NET, and Node.js and has served in various team and project leadership positions throughout her career. The breadth of these experiences gives Pauline a keen understanding of the opportunities and risks that open-source code poses. She holds a computer science degree from Queens University in Belfast.

JOE COLETTA

SR. PRODUCT MARKETING MANAGER
CONTRAST SECURITY

Joe Coletta oversees product marketing strategy and execution for Contrast OSS and is focused on open-source security. Joe has worked in the application security space for over a decade and has a comprehensive view of all aspects, starting in customer success before migrating to go-to-market strategy. Joe leverages his consultative experience with application security practitioners to highlight solutions that solve key customer problems.

[1] Sage McEnery, "How much computer code has been written?" Medium, July 18, 2020.

[2] Shivam Srivastava, et al., "Developer Velocity: How software excellence fuels business performance," McKinsey & Company, April 20, 2020.

[3] Ibid.

[4] Josephine Wolff, "The SolarWinds Hack Is Unlike Anything We Have Ever Seen Before," Slate, December 18, 2020.

[5] "Data Breach Investigations Report, 2020," Verizon, April 2020.

[6] Forrester found a 40% increase in the use of open-source code in one year; see Amy DeMartine and Jennifer Adams, "Application Security Market Will Exceed $7 Billion by 2023," Forrester, updated March 29, 2019.

[7] "Securing the world's software," GitHub Octoverse, accessed March 26, 2021.

[8] See Stephen Gates, "Code Exposure: The Vulnerabilities in Your Code & Where They Originate," Security Boulevard, July 10, 2019.

[9] Thomas Claburn, "Ruby off the Rails: Code library yanked over license blunder, sparks chaos for half a million projects," The Register, March 25, 2021.

[10] John Koetsier, "97% Of Executives Say Covid-19 Sped Up Digital Transformation," Forbes, September 10, 2020.

[11] "How COVID-19 has pushed companies over the technology tipping point—and transformed business forever," McKinsey & Company, October 5, 2020.

[12] "The State of DevSecOps Report," Contrast Security, November 2020.

[13] "Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3% in 2020," Gartner, July 23, 2020.

[14] Rich Cracknell, et al., "Securing software as a service," McKinsey & Company, September 2019.

[15] Alex Powell, "The biggest mistakes in SaaS security," Cloud Security Alliance, February 8, 2021.

[16] Manolo Edge, "3rd party libraries, are they a risk?" DEV, January 15, 2020.

[17] Cathy Zhou, "The State of the Octoverse 2019," GitHub, November 6, 2019.

Contrast SECURITY

Contrast Security provides the industry's most modern and comprehensive Application Security Platform, removing security roadblocks inefficiencies and empowering enterprises to write and release secure application code faster. Embedding code analysis and attack prevention directly into software with instrumentation, the Contrast platform automatically detects vulnerabilities while developers write code, eliminates false positives, and provides context-specific how-to-fix guidance for easy and fast vulnerability remediation. Doing so enables application and development teams to collaborate more effectively and to innovate faster while accelerating digital transformation initiatives. This is why a growing number of the world's largest private and public sector organizations rely on Contrast to secure their applications in development and extend protection in production.

240 3rd Street
2nd Floor
Los Altos, CA 94022
Phone: 888.371.1333
Fax: 650.397.4133

Contrast SECURITY

contrastsecurity.com