State of Open Source Security Report

2020

snyk

Table of contents

Introduction

The use of open source software has become almost ubiquitous in the software development community. Development ecosystems have grown in their dependence on third-party libraries and packages to streamline development. Awareness of how the increased prevalence of open source software impacts the security posture of enterprise organizations appears to be growing as well. For their part, the open source community is responding. In November of 2019, GitHub (acquired in 2018 by Microsoft) announced the launch of the GitHub Security Lab with free access to its CodeQL code review tool and the opening of its Advisory database to public access.

However, the question remains—is this increased awareness translating into improved security and practices related to the use of open source software? As part of our annual report on the state of open source security—and our desire to help the development community leverage open source securely—we sought to answer many of these questions.

Once again, this year we gathered information from a few key sources including the following:

- A survey created and distributed by Snyk and our partners that was completed by over 500 developers, security practitioners, and operations technologists.
- Internal data from the Snyk vulnerability database, as well as correlated data from the hundreds of thousands of projects currently monitored and protected by Snyk.
- Research and data published by various sources that include aggregated data from scanning the millions of repositories in GitHub, GitLab, Bitbucket, and others.

Through our analysis, Snyk was able to identify some key themes and trends that shed new light on the current state of security across open source ecosystems. Let's begin by looking at some of those.

Key takeaways and trends

Open source universe

- Open source ecosystems continue to expand, led by npm which grew over 33% in 2019, now spanning over 1,300,000 packages to this date.
- The majority of open source vulnerabilities continue to be discovered in indirect dependencies:
  - npm - 86%
  - Ruby - 81%
  - Java - 74%

Container & orchestration challenges

- Official base images tagged as latest include known vulnerabilities.
- Over 30% of survey participants do not review Kubernetes manifests for insecure configurations.
- Requirements for security-related resource controls in Kubernetes are not widely implemented.

Vulnerability trends

- New vulnerabilities were down almost 20% across the most popular ecosystems in 2019.
- Cross-site scripting vulnerabilities were the most commonly reported.
- Two prevalent prototype pollution vulnerabilities resulted in an impact on over 25% of scanned projects.
- New vulnerabilities reported in common Linux distributions demonstrate the need for comprehensive monitoring for new vulnerabilities in container images.
- SQL Injection vulnerabilities, while decreasing in prevalence in most ecosystems, have increased over the last three years in PHP packages.

Security culture

- Increasingly, survey respondents feel that security for software and infrastructure should be shared among development, security, and operations roles.
- However, few organizations have programs in place to develop shared responsibility across the Dev, Sec, and Ops personnel.

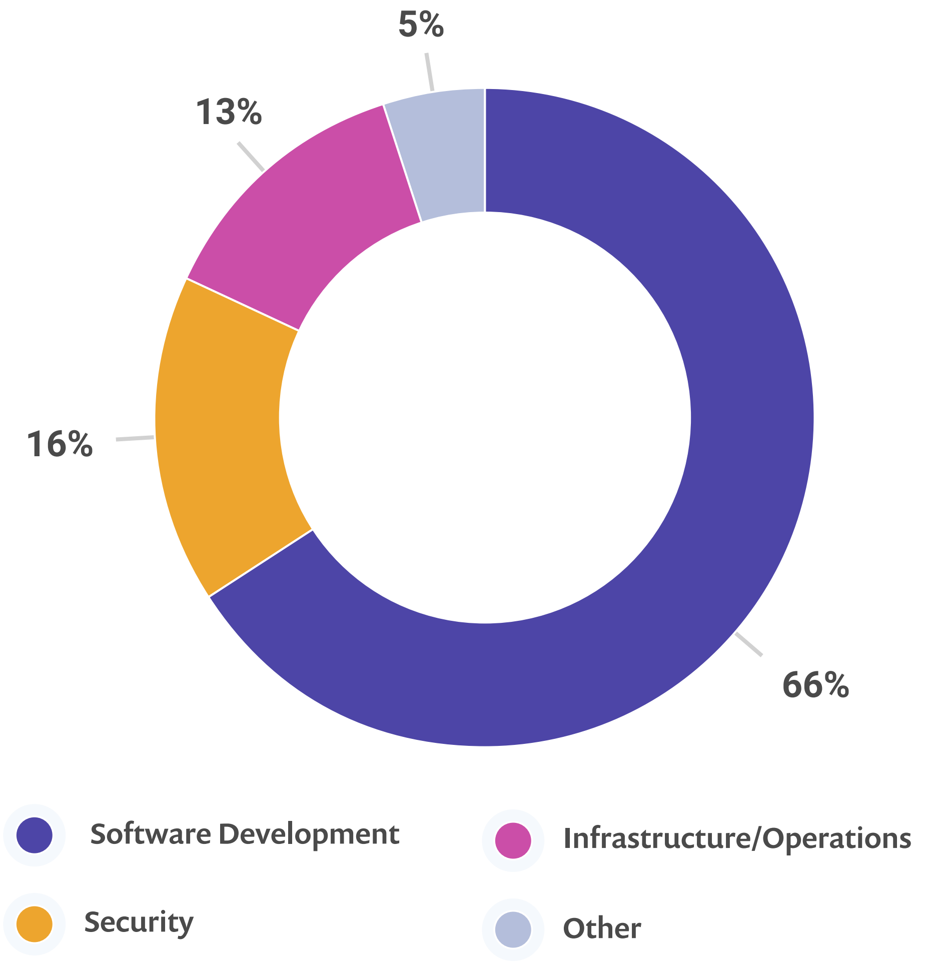
A word about our survey

In an effort to gain additional perspectives on the data we researched regarding open source software, we reached out to the community at large. Over the course of several months, we surveyed over 500 industry professionals to ask about their use and maintenance of open source packages in their software, as well as cloud native technologies. Throughout this report, we reference various findings from that survey.

However, to put those findings into perspective, it is important to understand the survey itself better. The survey was made available via social media, a number of select partners, and direct outreach to communities in software development and security. Respondents were not scientifically chosen so, there may be biases in some of the results that need to be considered.

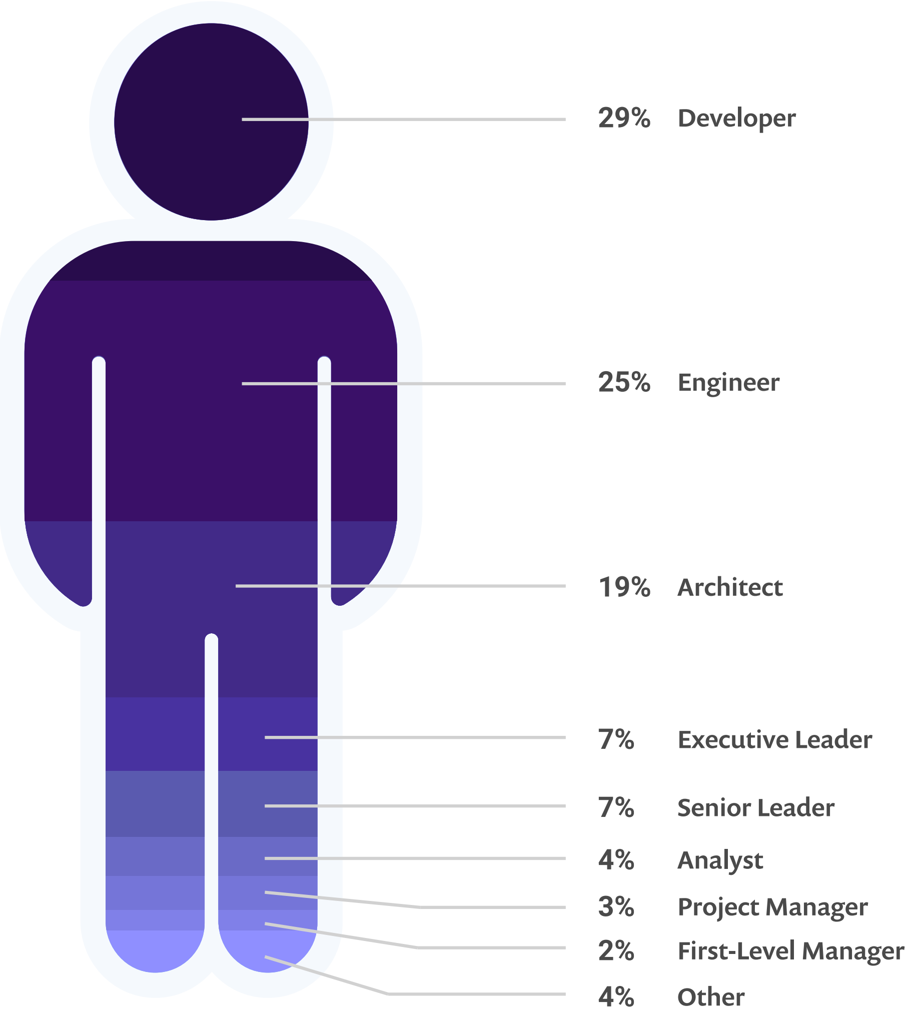
For instance, we asked each respondent what their primary role was. Maybe not too surprising, 66% work in software development. Responses from Security and Infrastructure/Operations personnel amounted to a combined 29.4%. So, while we were able to gain important security and operations perspectives, the final results will be slanted toward the view of software developers.

Survey participants job responsibilities



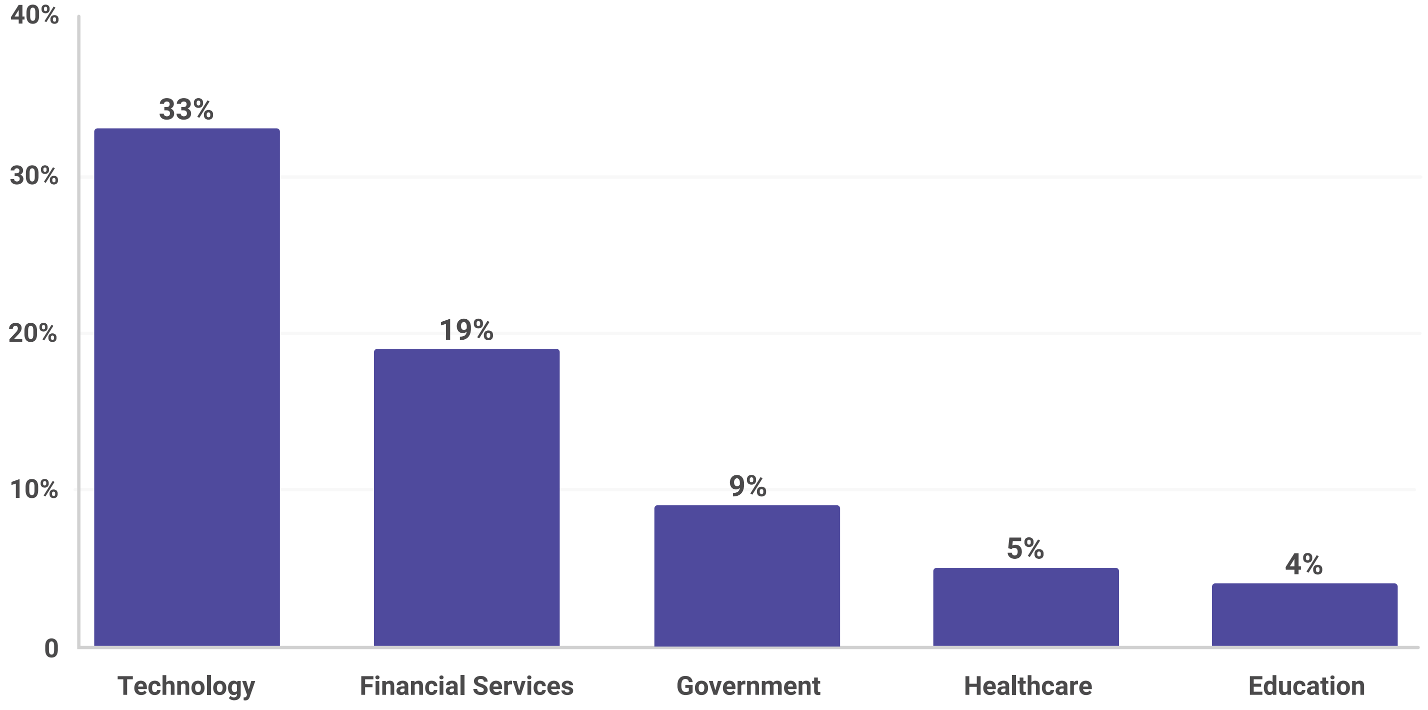
We also wanted to understand the relative position of our participants within their organization. We asked the respondents to identify what best described their role in terms of general titles. The response set was weighted more heavily toward individual contributors than leadership.

Survey participants - job role



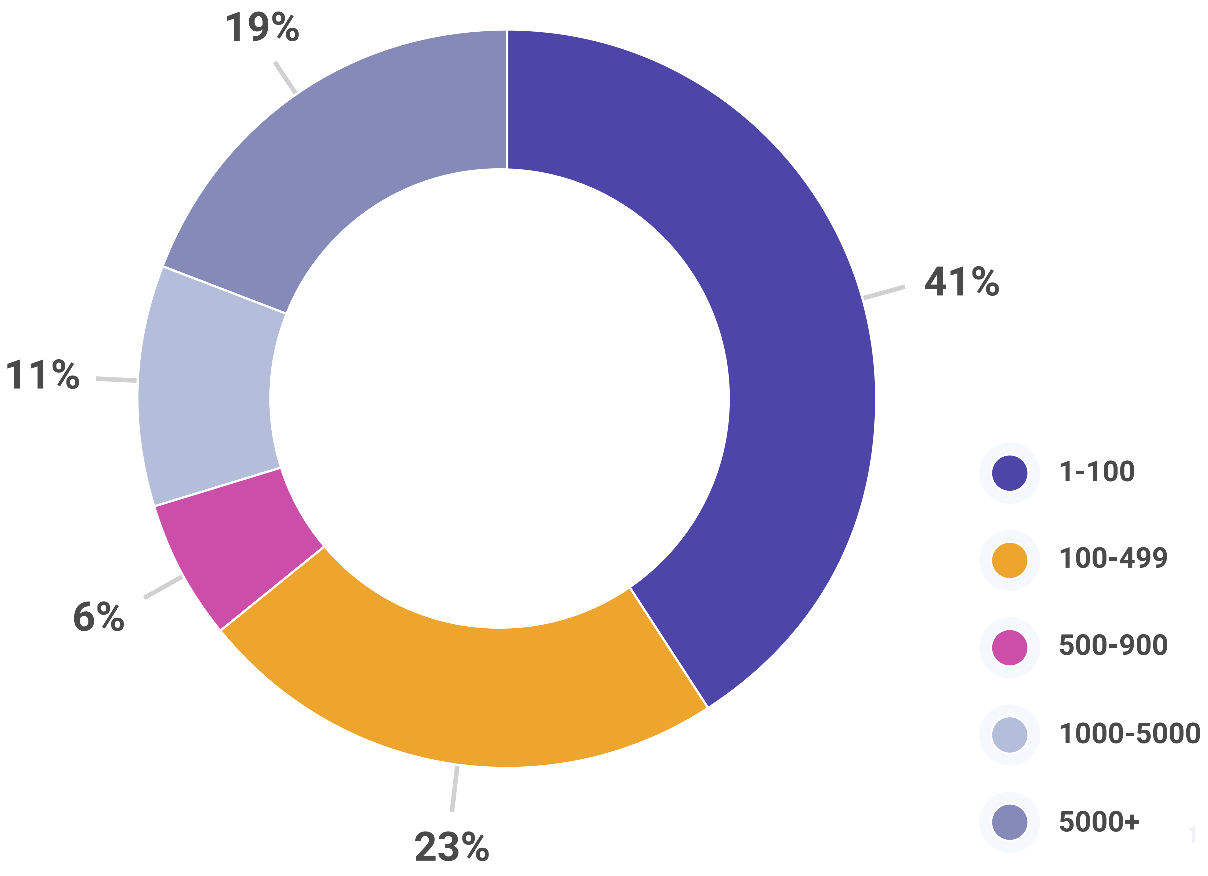
Understanding what industries are providing their perspectives is also crucial, see below the breakdown of the top five industries represented in our report. You can see that Technology and Financial Services were the most highly represented.

Response by top 5 industries



Organization size

Finally, we believe it is important to understand the size of the organizations that our participants represented. While there was significant influence from small organizations or individuals in our response set, the remaining participants provided pretty equal representation across organizational sizes.



1

The open source universe

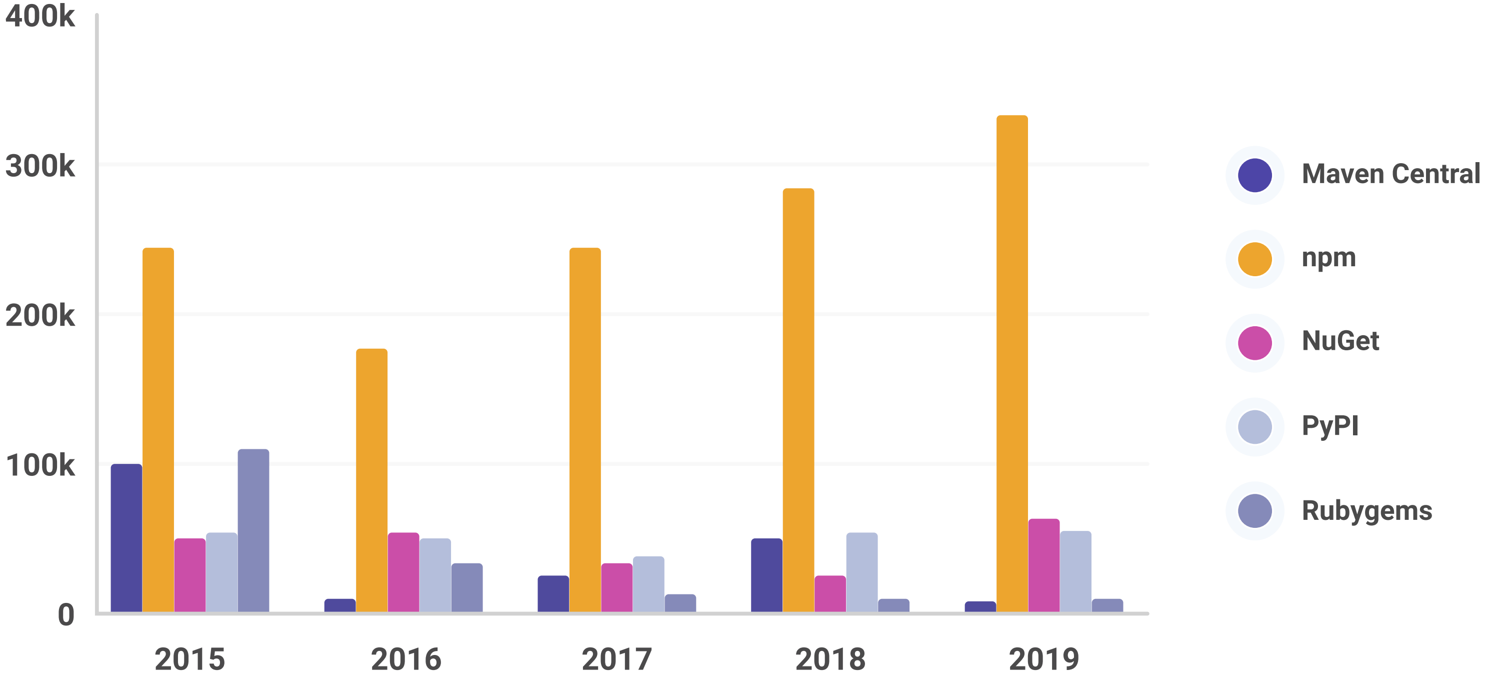
The trend of incredible growth in the use and contribution of open source software across various software development ecosystems continued in 2019. In the State of the Octoverse report, GitHub reported that over 10 million new users joined GitHub last year bringing the total number of developers on that platform to over 40 Million.

Ecosystem growth

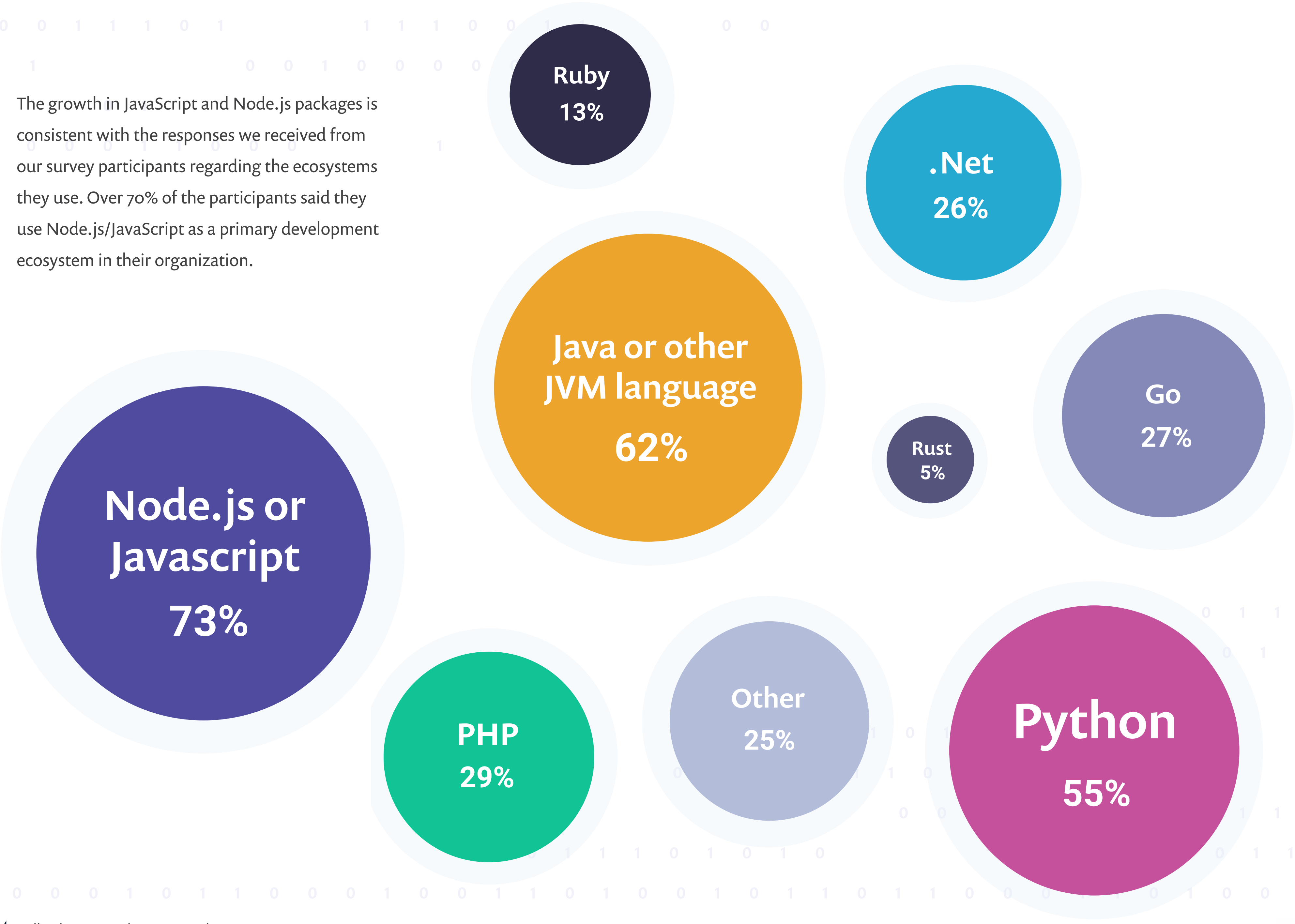
Our research across multiple ecosystems and repositories was consistent with the overall growth trends seen across the open source community. In terms of development ecosystems, we continued to track the progress of five of the most common open source ecosystems.

The growth in open source packages is driven largely by the continued popularity and growth of the JavaScript ecosystem. Conversely, fewer new packages were created for Java and Ruby than in the previous year. Overall, across these five ecosystems, the total number of packages grew significantly. In particular, npm grew by over 33% from the end of 2018 to the end of 2019.

New packages created by ecosystem per year



The growth in JavaScript and Node.js packages is consistent with the responses we received from our survey participants regarding the ecosystems they use. Over 70% of the participants said they use Node.js/JavaScript as a primary development ecosystem in their organization.



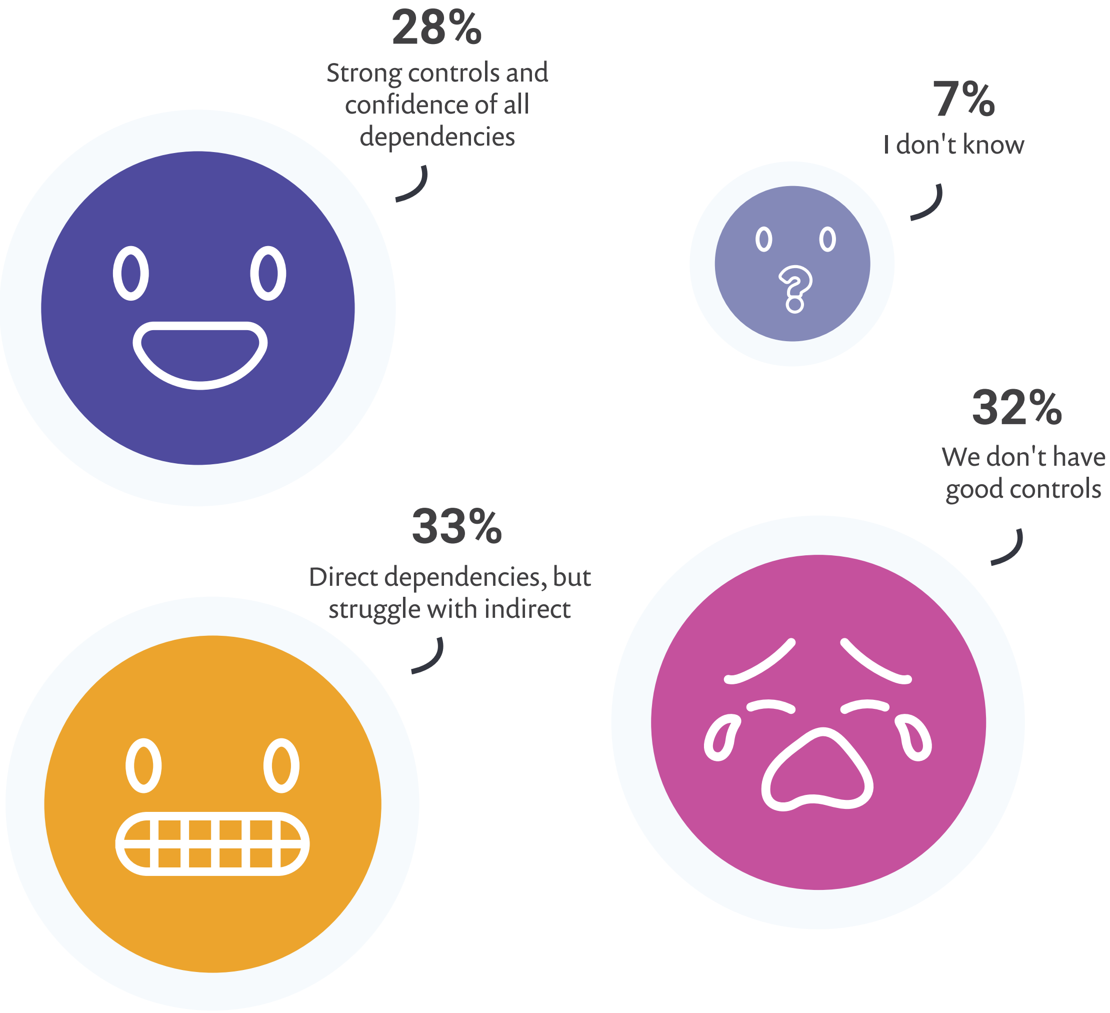
The security implications of open source development

A key risk factor when organizations consider the security of their open source software utilization, centers around the idea of maintaining a Software Bill of Materials. Organizations are challenged with understanding what open source libraries and packages are included in the software they produce. This challenge comes from the difficulty in understanding not only the direct open source dependencies defined in their code but the indirect dependencies that are introduced as a result.

60% of organizations surveyed do not fully inventory the dependency trees in their software

In our survey, we asked the participants about their ability to track open source dependencies in their software. Over 60% said they do not have a good view into the full dependency trees of their software. As a result, it would be extremely difficult to identify if a newly discovered vulnerability in an open source package affects their code or not.

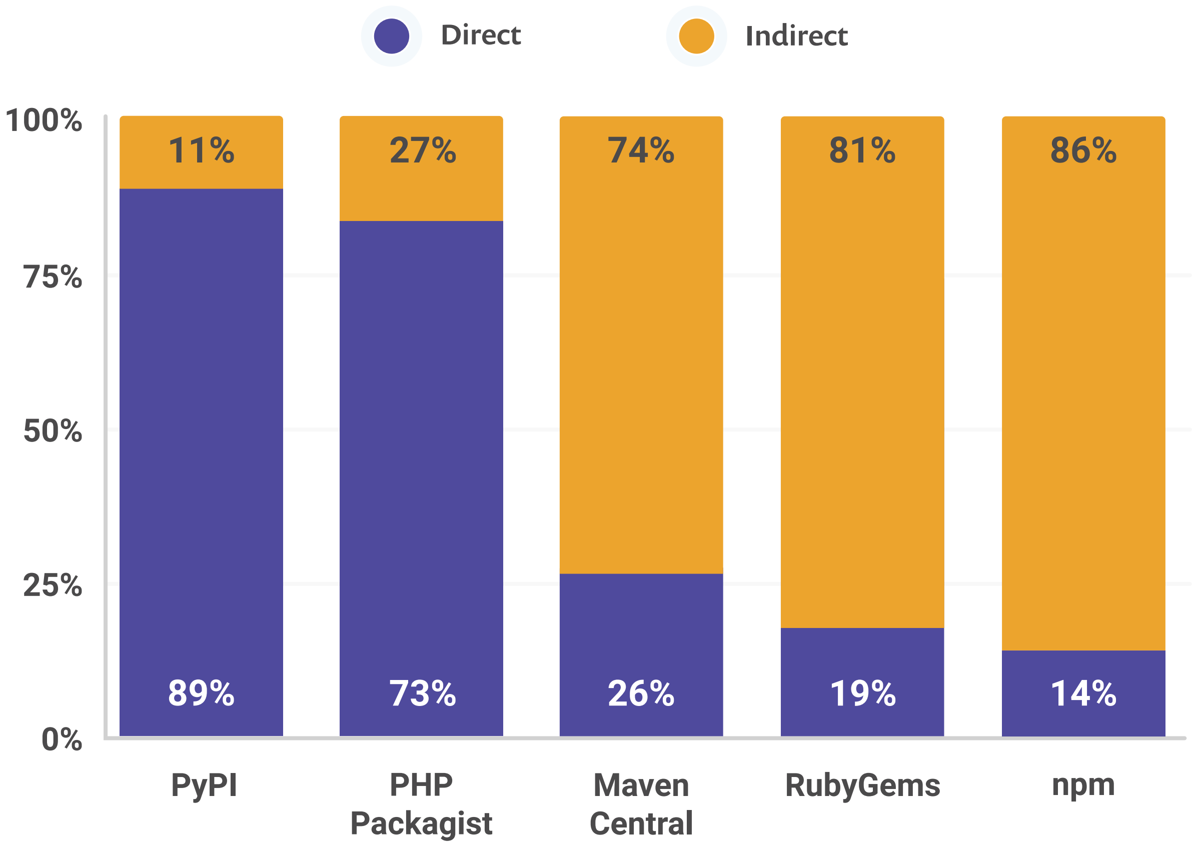
How do you track open source dependencies?



When you consider this information in light of the growth and wide-spread use of ecosystems like Node.js, the risk that open source development poses to organizations becomes all too real. Each year we analyze the vulnerabilities that Snyk has identified in hundreds of thousands of projects. We continue to find that the majority of Node.js, Java, and Ruby vulnerabilities identified are introduced via indirect dependencies.

Over 70% of vulnerabilities discovered in Node.js, Java and Ruby are found in indirect dependencies

Vulnerabilities from direct versus indirect dependencies



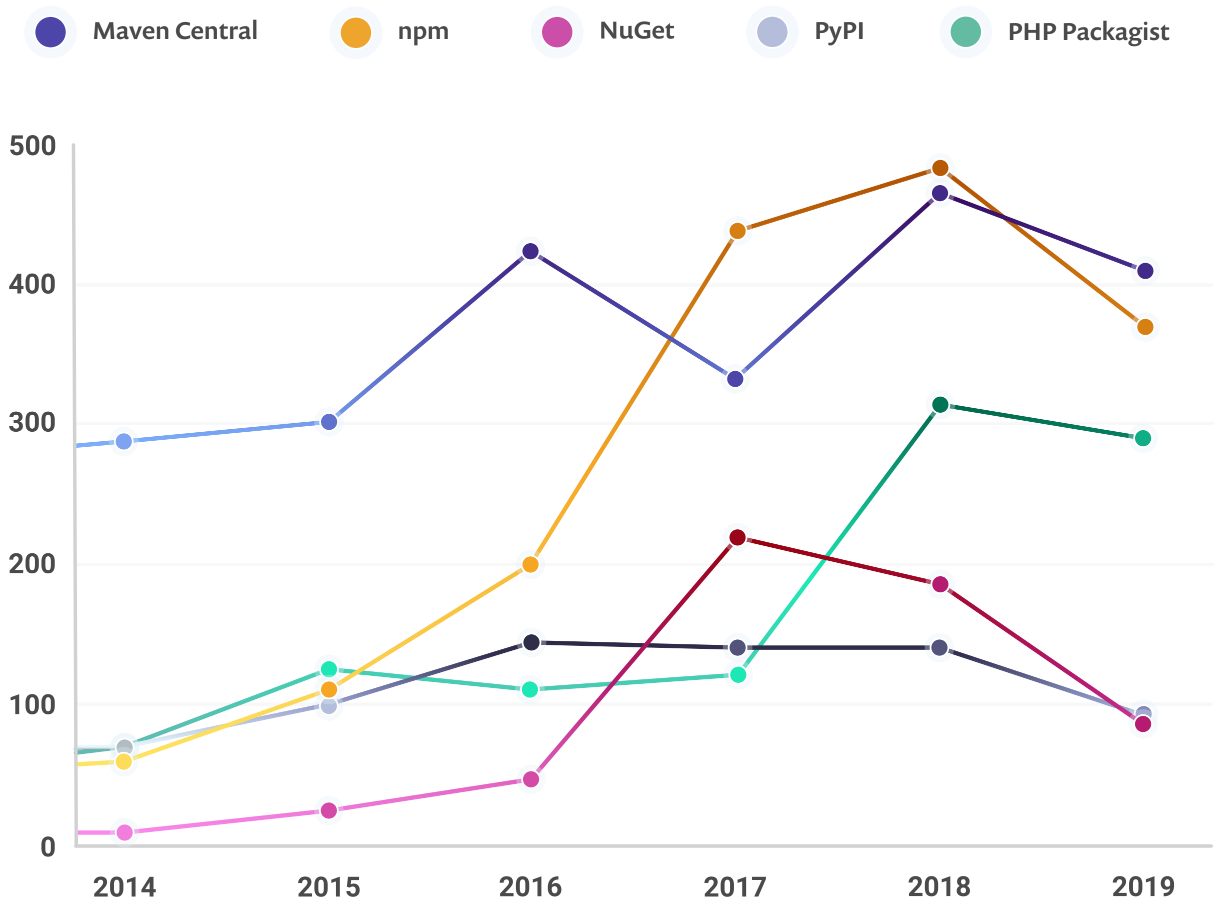
2 Detailed open source vulnerability analysis

What would any discussion of the state of open source security be without taking a look at the vulnerabilities being discovered and disclosed in open source packages and libraries? This year we are taking an even deeper look at vulnerability and ecosystem-level trends that affect the overall security posture of the open source community.

High-level vulnerability trends

In past years, we have seen that in terms of total vulnerabilities identified in open source packages across the ecosystems, Node.js and Java have traditionally shown the greatest number of new vulnerabilities each year. That trend continued in 2019, perhaps—at least to some extent—due to the relative popularity of those ecosystems. One potentially encouraging sign is that across all six popular ecosystems we looked at, there were fewer new vulnerabilities reported in 2019 than in 2018. While one year is hardly enough data to draw a significant conclusion if this trend continues, it could be a positive sign that efforts to improve the security of open source software are starting to pay off.

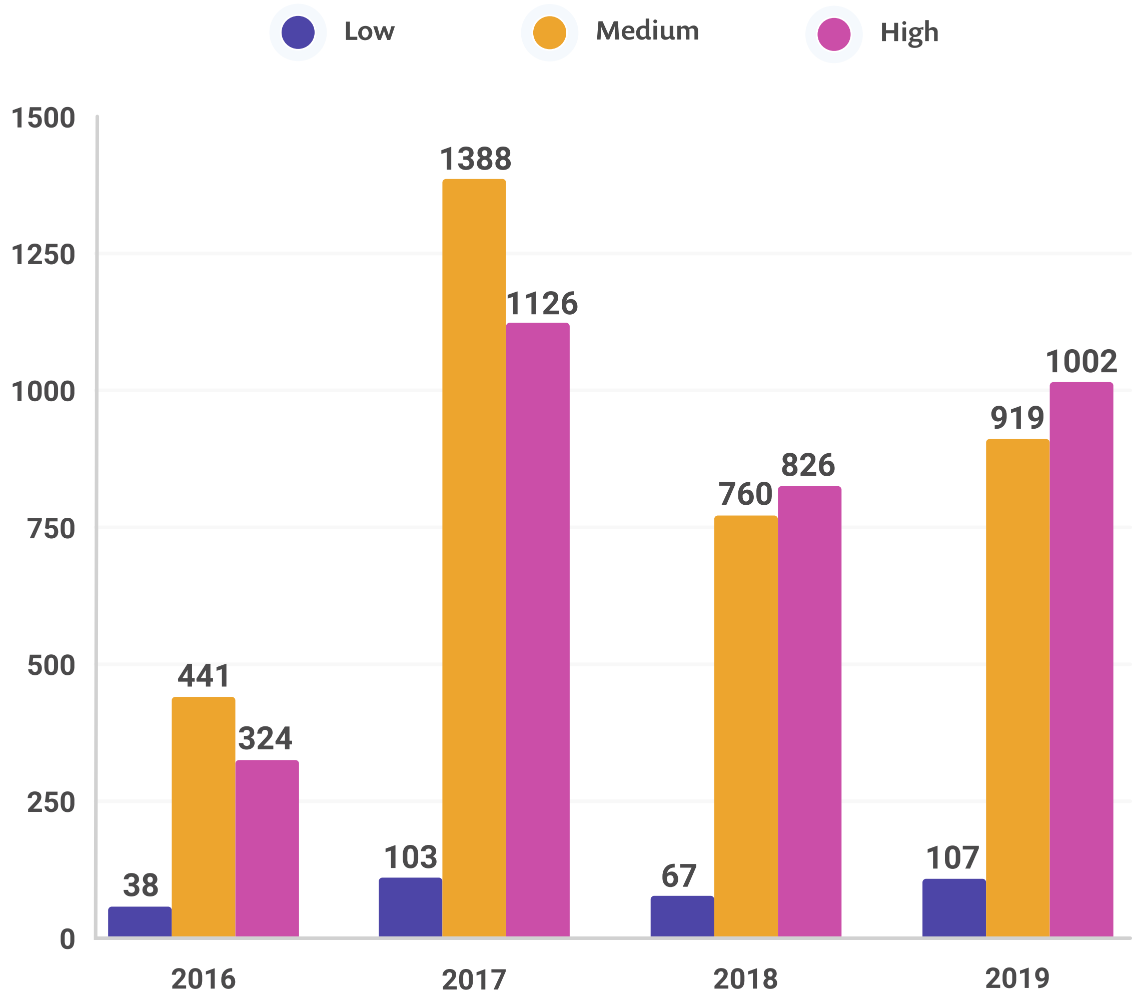
Vulnerabilities identified in ecosystems since 2014



Unfortunately, not all the high-level analysis of vulnerabilities in the open source ecosystems paints the same optimistic picture. It is good to see that the number of new vulnerabilities disclosed this year in those popular ecosystems went down. However, the overall number of vulnerabilities reported across all ecosystems increased in 2019 after having shown a decrease in 2018. Compounding that concern is that, once again in 2019, the majority of the vulnerabilities identified were considered high severity.

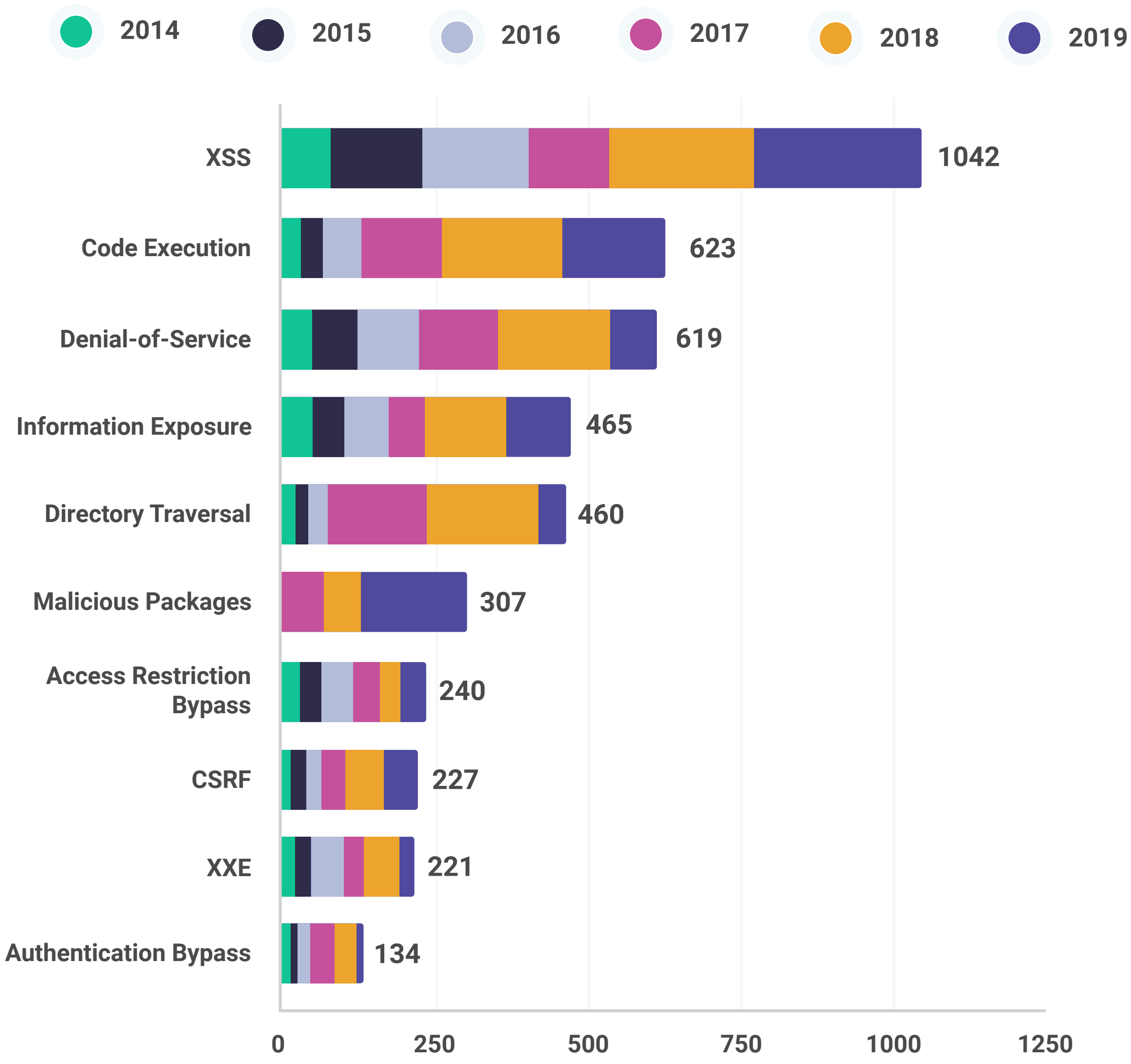
> **Note:** One may notice that our numbers for previous years do not match the numbers presented in last year's report. Those discrepancies are due to vulnerabilities that were reported but not confirmed until after the end of the year, new or additional sources of vulnerability data we included to our vulnerability database, as well as the addition of new ecosystems to our vulnerability database.

Vulnerability severities by year



Looking at the numbers of new vulnerabilities, it is easy to assume that the discovery of new attack vectors is behind the large number of new vulnerabilities reported. However, a deeper analysis of the vulnerabilities shows that is simply not the case. We analyzed the types of vulnerabilities that have been reported in open source software dating back to 2014. The data shows that well-understood vulnerabilities continue to contribute significantly to the totals. For instance, cross-site scripting — which has been a category on every OWASP Top 10 list since the very first list was created in 2003—is the most common vulnerability discovered since 2014 and, year over year, it is in the top 3 of reported vulnerabilities.

Top vulnerabilities since 2014



Given the number of controls and frameworks available in the various ecosystems to prevent exactly these types of attacks, it is somewhat concerning that these numbers continue to progress in this way. The introduction of Content Security Policies (CSP) is one of the latest attempts to thwart these types of attacks. However, based on the current vulnerability trends, it is clear that more work needs to be done. In the end, it is up to the developer to implement proper validation, while

> it is incumbent upon security professionals to help enable the discovery of potential flaws that lead to these types of attacks and assist in designing remediations.

Equally concerning is the number of code execution vulnerabilities being discovered. This category includes both remote code execution and arbitrary code execution exposures that can lead to a variety of high severity exploits. These types of vulnerabilities can often be leveraged to spread malware/ransomware. Since malware is a significant contributor to many of the high-profile breaches being reported, the prevalence of these vulnerabilities should raise some eyebrows as well.

What's old is new again

The Open Web Application Security Project (OWASP) was founded in 2001 with the goal of helping developers and organizations create secure applications that could be trusted. In 2003, OWASP published its first Top 10 list of common web application security risks. The list provided categories of common types of vulnerabilities that impact web applications. Since that time, OWASP have published periodic updates to the list, most recently releasing a version in 2017.

A number of the categories that appeared on that first list in 2003 remain on the list today, among them is cross-site scripting (XSS). Cross-site scripting has been one of the most commonly discovered vulnerabilities across applications for more than a decade. As we see in this report, that trend continues even today.

XSS is a vulnerability that can have a wide range of impacts. It allows attackers to inject malicious browser-side script that is executed when unsuspecting users visit a site. These attacks can vary from simple defacement or functionality manipulation to theft of session identifiers leading to session hijacking.

The primary strategy for preventing and/or remediating XSS vulnerabilities in applications has not changed since it first was reported on the OWASP Top10 list in 2003. Properly validating and/or sanitizing all data received from the user's browser is the most effective way to ensure that applications are free from these vulnerabilities. However, that is a solution that is much easier stated than it is implemented.

Numerous safeguards have been implemented in development ecosystems and newer browsers to help prevent successful attacks based on XSS vulnerabilities. However, the presence of XSS vulnerabilities as the most reported form of application weakness in 2019 is evidence that these controls are not being implemented consistently or not completely effective. Therefore, it is still incumbent on the developer to be aware of these attack vectors and not assume that their dependencies have enabled proper protections.

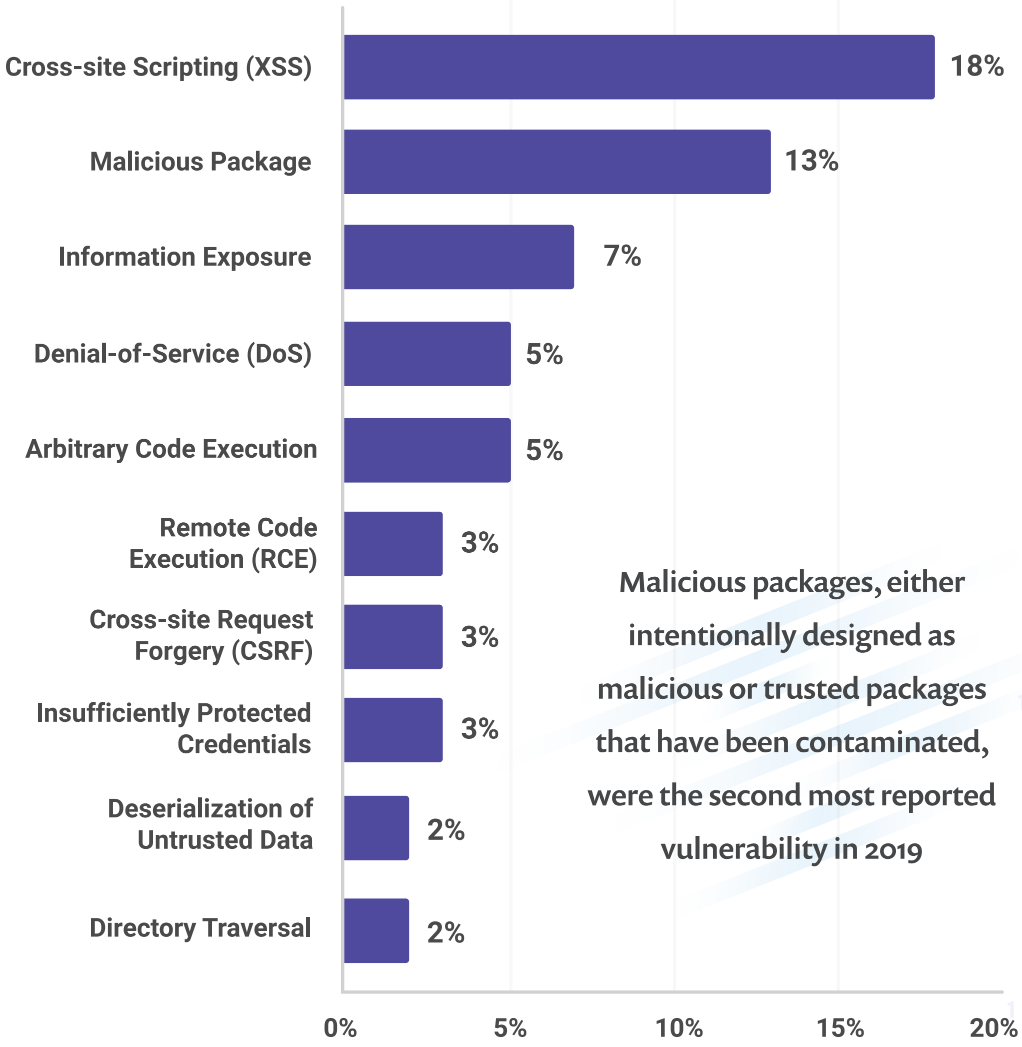
Vulnerability impacts in 2019

Understanding the prevalence of various forms of security vulnerabilities within open source packages and libraries is only one piece of the security picture. We dug deeper to look at the overall impact of vulnerabilities across the open source community and within projects that rely on open source dependencies.

Looking at vulnerabilities reported in 2019, the top ten vulnerability titles follow pretty closely to what we see in terms of the overall trends discussed in the previous section. In 2019, cross-site scripting vulnerabilities remained at the top of the list as the most commonly reported vulnerabilities.

However, what is particularly interesting to note, is that the second most common vulnerability identified were malicious packages. These are situations where a typically known and trusted package has been contaminated with an attack payload or a package intentionally designed and released with an attack payload built into it. We will talk more later about how developers and organizations attempt to understand the health and trustworthiness of their software dependencies. For now, this trend suggests that the threats are significant and efforts to improve our understanding of package health are important.

Top 10 vulnerabilities of 2019



Malicious packages, either intentionally designed as malicious or trusted packages that have been contaminated, were the second most reported vulnerability in 2019

How do these vulnerabilities translate into their impact on software projects? Analyzing this question is of particular importance as it demonstrates how widespread the attack exposures are in the software community. Vulnerabilities are less of a risk if the affected packages are only used in a handful of projects. However, when a vulnerability is reported in a highly popular package affecting thousands of projects, that creates a higher probability of that vulnerability being exploited by attackers.

To that end, we examined the number of vulnerabilities that were identified in the hundreds of thousands of projects that have been scanned and monitored by Snyk. The results were quite interesting in the story they tell. Despite the high prevalence of cross-site scripting vulnerabilities being reported, those vulnerabilities only impacted about 6.7% of the projects scanned.

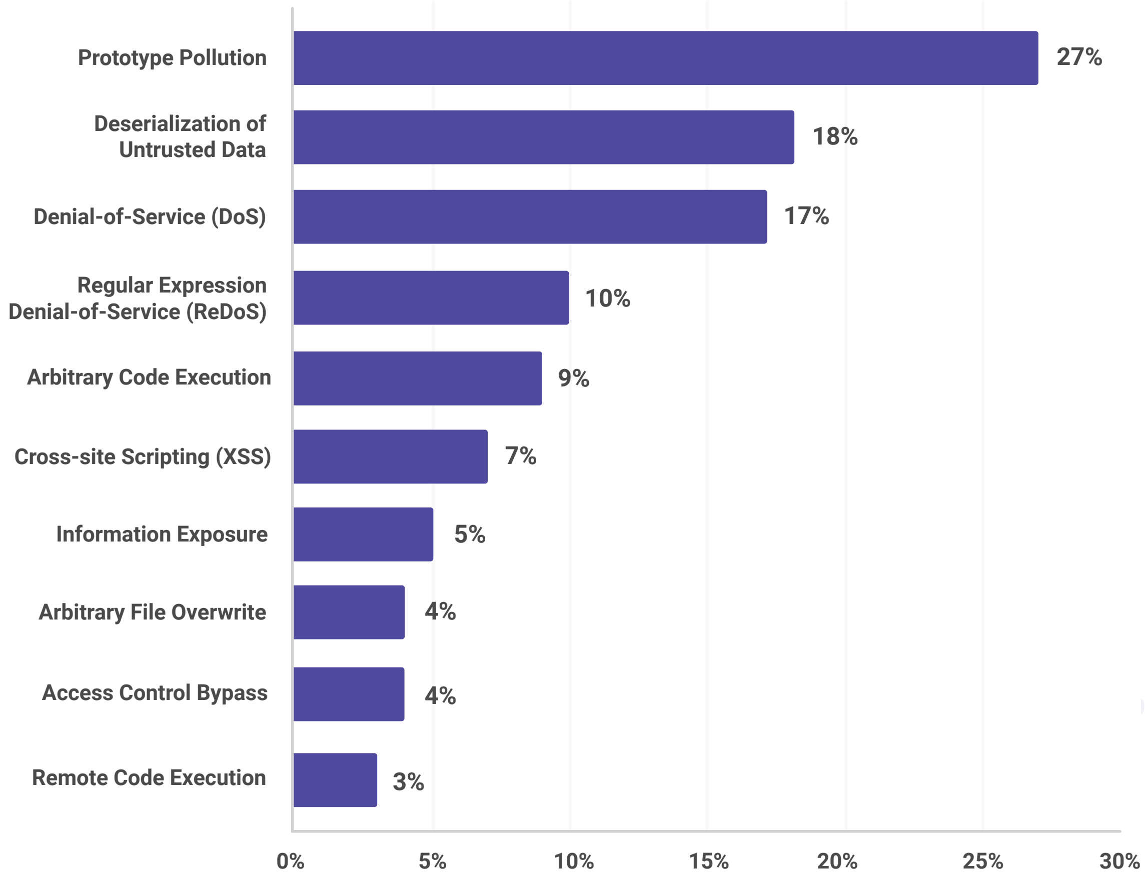
Conversely, the top vulnerability currently impacting scanned projects is prototype pollution in nearly 27% of all projects.

Prototype pollution is one potential vector through which attackers can introduce malicious code into otherwise trustworthy packages. The prevalence of prototype pollution across so many projects is likely a result of multiple high-profile vulnerabilities discovered in 2019.

The first vulnerability was discovered in jQuery and disclosed via HackerOne. Given the relative popularity of jQuery, a significant number of projects were affected.

In July of last year, Snyk researchers discovered a prototype pollution vulnerability in the extremely popular Lodash package. The vulnerability, CVE-2019-10744, affected all versions of the package at the time of discovery and as a result, its impact was very widespread resulting in a very high number of impacted projects.

Top 10 vulnerabilities of 2019 by project impact

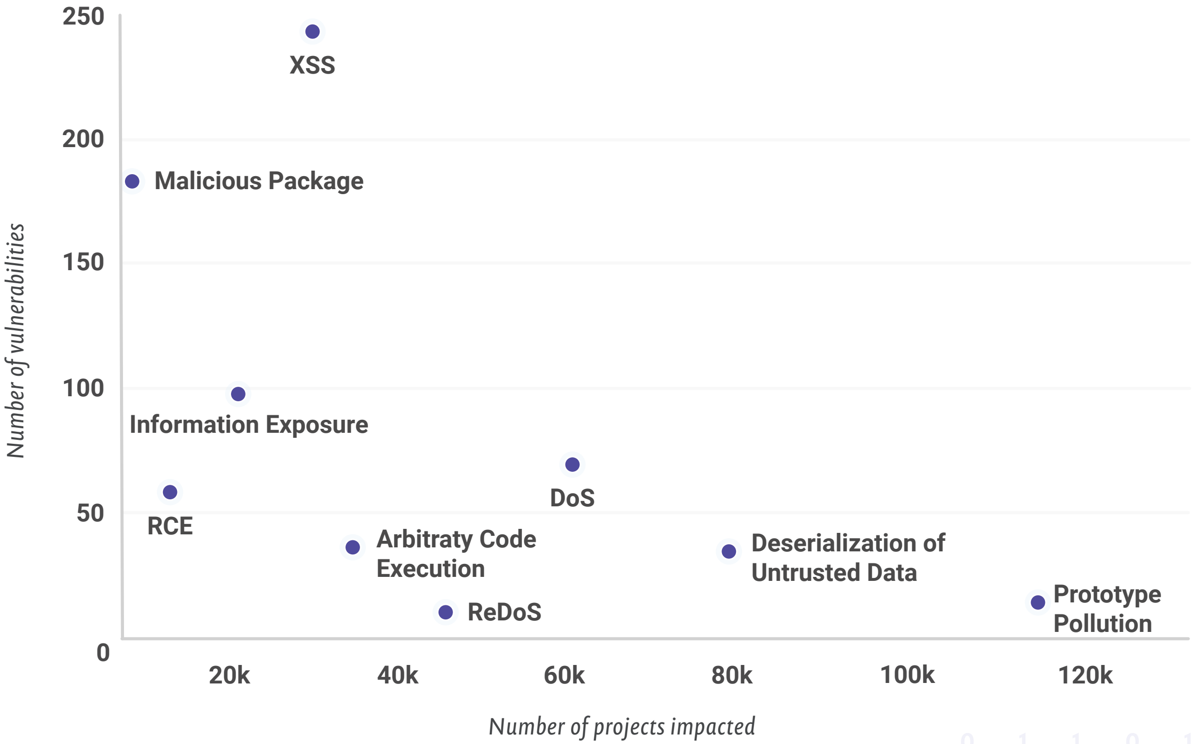


The differences between which types of vulnerabilities were reported most often and which types impacted the most projects brings up an interesting question. If we understand both the prevalence and impact of each type of vulnerability, could that help us better determine which are the biggest threats to software?

While fewer than 25 prototype pollution vulnerabilities were reported in 2019, they impacted over 115,000 projects scanned

We analyzed the top five vulnerabilities from the previous two analyses and plotted them according to both their prevalence and their impact on projects. Maybe somewhat surprising, there were no vulnerabilities that stood out as having both high prevalence and high impact. For instance, while there are a lot of malicious packages reported, very few projects were impacted by those packages. Conversely, while there are relatively few reports of deserialization vulnerabilities, the ones that have been reported affected a high number of projects.

Vulnerabilities reported vs projects impacted



Snyk discovers prototype pollution in Lodash

On July 2nd, 2019, we published a high severity prototype pollution security vulnerability (CVE-2019-10744) affecting all versions of Lodash, as the result of an on-going analysis led by the Snyk Security Research team. An updated version of Lodash (4.17.12) was subsequently released on July 9th which included fixes submitted by Snyk to remediate the vulnerability.

At the time, the popular npm library was used by 4.35 million projects on GitHub alone. The project had just shy of 40k GitHub project stars, and the library had been downloaded over 80 million times each month.

Needless to say, a high severity vulnerability in a library as popular as Lodash affects a large proportion of npm users.

The team proactively opened thousands of automatic fix pull requests for its users to remediate the vulnerability. The news of this Lodash security vulnerability came to light merely three months after a similar prototype pollution vulnerability was reported in the ever-popular jQuery JavaScript frontend library. Similar to other prototype pollution vulnerabilities, the implementation of unsafe recursive JSON merge may result in the ability to tamper with JavaScript's Object which influences other data-types through the prototype chain. The implications of such vulnerabilities can range from property injection to code injection and Denial-of-Service, depending on the affected use-case and whether this vulnerability can be exploited.

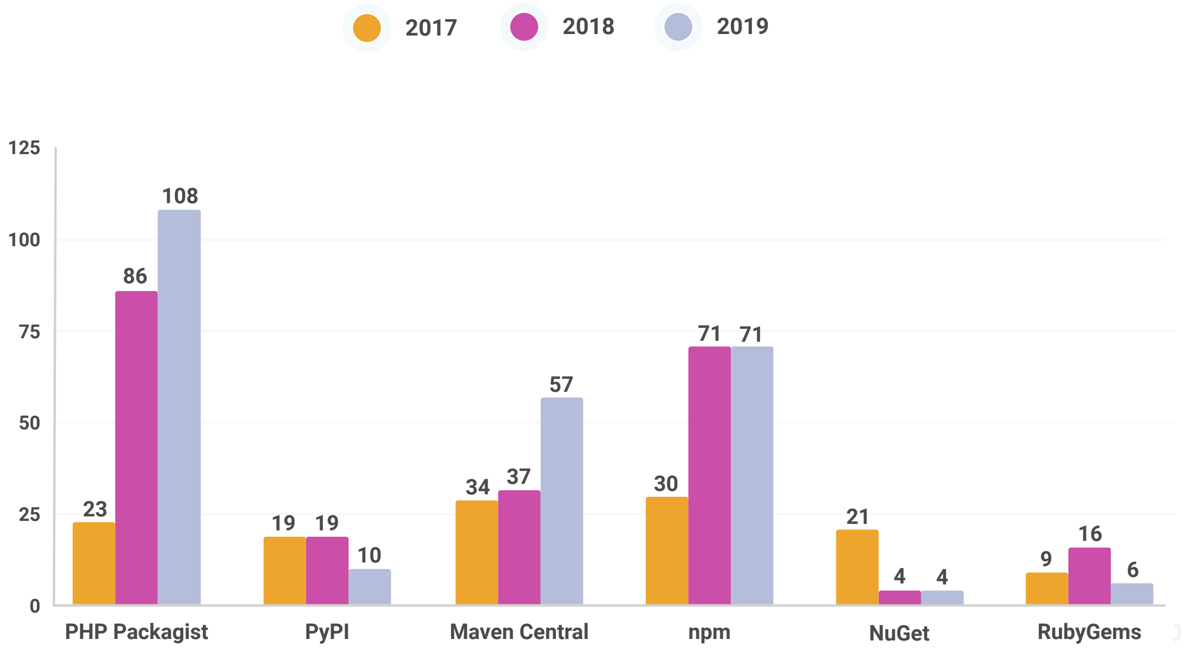
In the case of the Lodash vulnerability, the function **`defaultsDeep`** could be tricked into adding or modifying properties of **`Object.prototype`** using a constructor payload. The fix included a safety check to ensure that the global object was not being polluted. A test case was also added to ensure no future regressions occur.

This vulnerability serves as an example of how direct and indirect dependencies can amplify the impact of a single security flaw across a wide population of projects.

Ecosystem vulnerability analysis

Having established a broad understanding of vulnerabilities across the open source community, it also makes sense to understand how those top ten vulnerabilities impact individual ecosystems. We looked at the numbers for the top five vulnerability types across some of the most popular ecosystems.

New XSS vulnerabilities by year



Cross-site scripting (XSS)

Cross-site scripting vulnerabilities topped the list of newly reported vulnerabilities in 2019. How did that play out across the ecosystems? As you might expect, PHP leads the way with the newest vulnerabilities. Considering the relative lack of anti-XSS controls built into the ecosystem, it is far easier for developers to make mistakes or simply neglect to implement sufficient countermeasures.

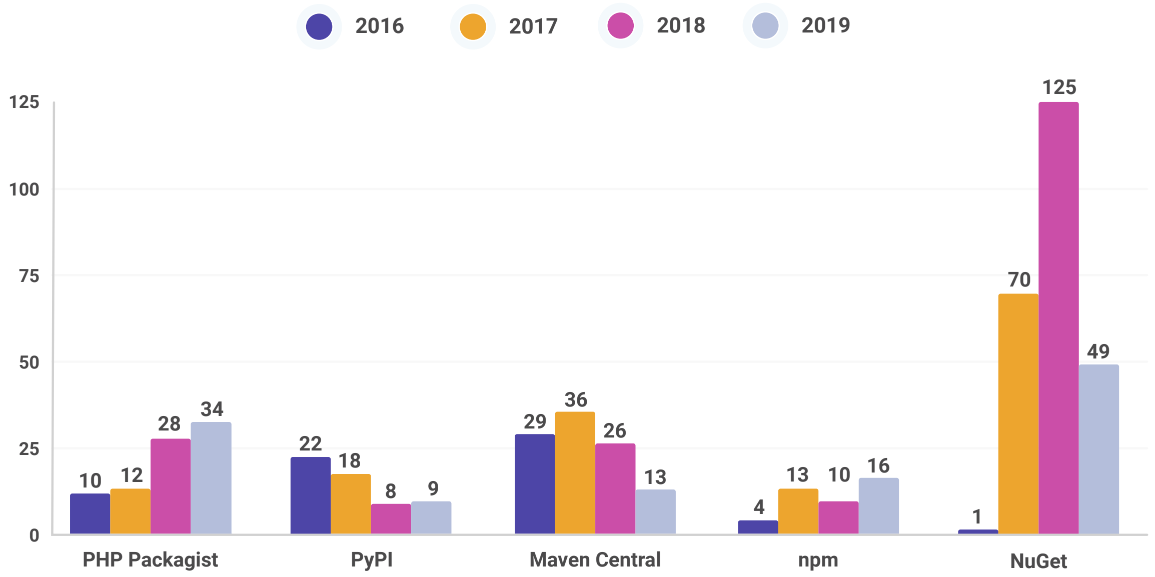
Since XSS is largely a web application attack, it is no surprise that Python and Ruby have relatively low numbers. Since they are far less commonly used to implement web applications, it simply stands to reason that there would be relatively few instances. The small number of instances identified in .NET packages available through NuGet is likely a sign of the significant work that has been done inside the .NET Framework to prevent these types of vulnerabilities. Additionally, the low use of open source dependencies in .NET is also reflected in the overall number of XSS vulnerabilities reported.

Code execution vulnerabilities

Just when one thought it might be safe to talk about the security posture of .NET applications, we bring forward the number two leading category of security vulnerabilities since 2014, code execution vulnerabilities. Vulnerability reports in 2018 were dominated heavily, especially in the .NET ecosystem, by prominent remote code execution vulnerabilities. While things improved in 2019 considerably (even in comparison to 2017 when over 75 vulnerabilities were reported) it is clear that this ecosystem has had a significant contribution in making arbitrary and remote code execution a prominent category.

Java and PHP for their part are not immune to these issues either, while Node.js and Python experienced far less activity in reports for this vulnerability type.

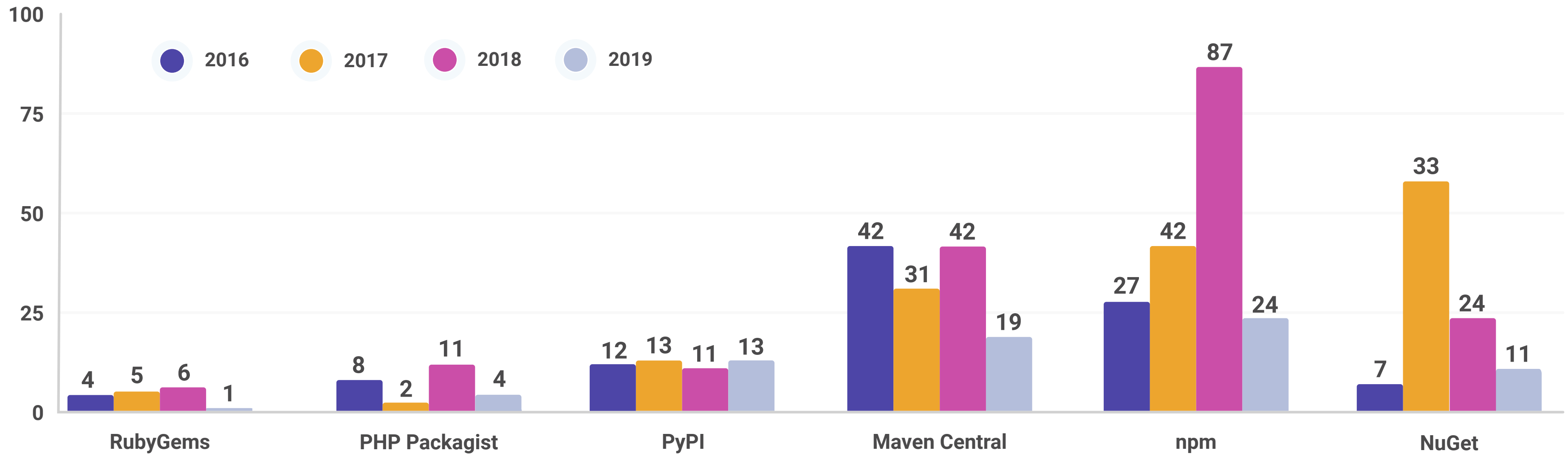
New code execution vulnerabilities by year



Denial-of-Service vulnerabilities

Denial-of-Service (DoS) attacks are one of the more frustrating risks that organizations face. They can be some of the most difficult to defend against and hardest to troubleshoot when they occur. Over the past three years and, in particular, in 2018, there were significant discoveries of Regular Expression DoS attacks in the Node.js ecosystem. Thankfully the number of new vulnerabilities in 2019 fell considerably. Java saw a similar decline after some notable flaws in 2018 and .NET has shown similar reductions as well—good news for the operations folks who are trying to keep pace with what can be an overwhelming threat landscape.

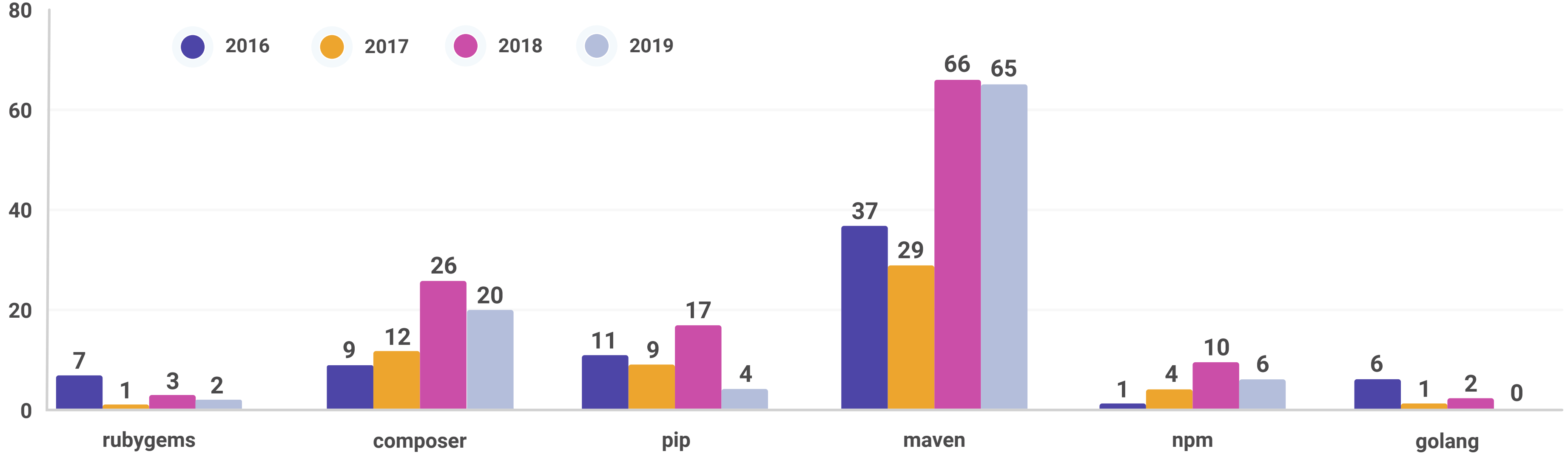
New Denial-of-Service vulnerabilities by year



Information exposure

As the name would suggest, this category describes application vulnerabilities that lead to the exposure of sensitive information. The Java ecosystem, over the last couple of years, has been hit by a number of vulnerabilities related to Jenkins plugins that have created potential exposure of sensitive information.

New information exposure vulnerabilities by year

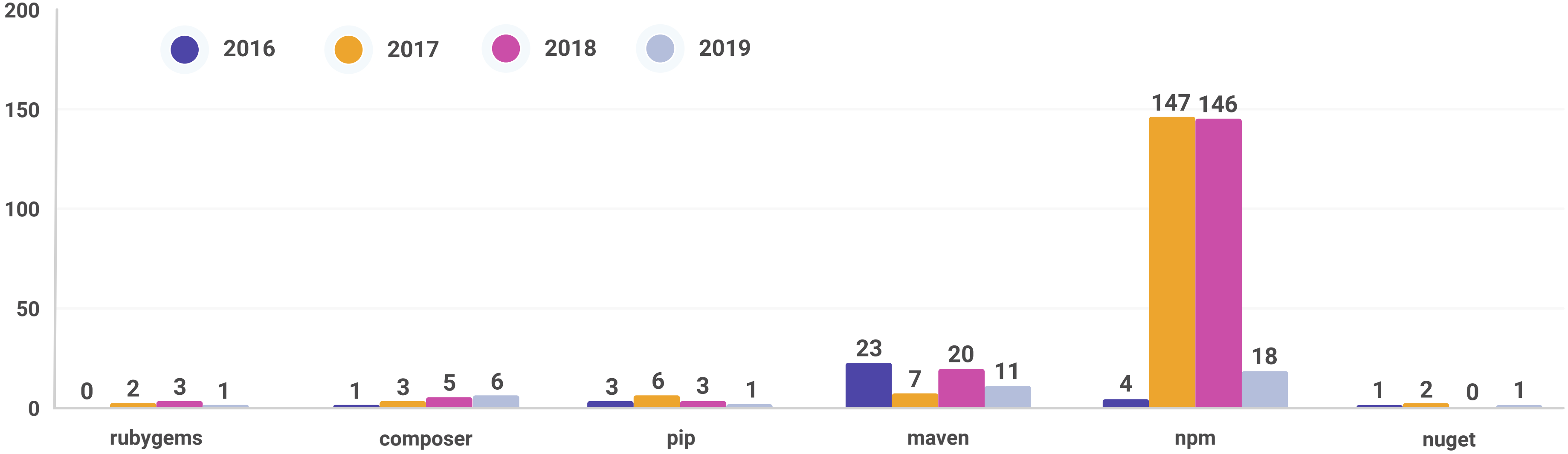


Directory traversal

Attacks that exploit directory traversal can be particularly damaging as the impact of such an attack can be varied and extensive. In some cases, such an attack simply arms the attacker with additional information about the system that they can leverage to build more complex exploits. In more serious instances, however, when an attacker is able to exploit a directory traversal vulnerability, it can lead to significant exposures—potentially including system-level credentials.

As you can see, vulnerabilities in Node.js disclosed in 2017 and 2018 account for the bulk of these vulnerabilities and why this category is prominent over the last several years. Outside of those large spikes in activity, the overall numbers of new vulnerabilities within this category are fairly low and, as a result, potentially indicative of improved countermeasures and overall developer awareness, as well.

New directory traversal vulnerabilities by year



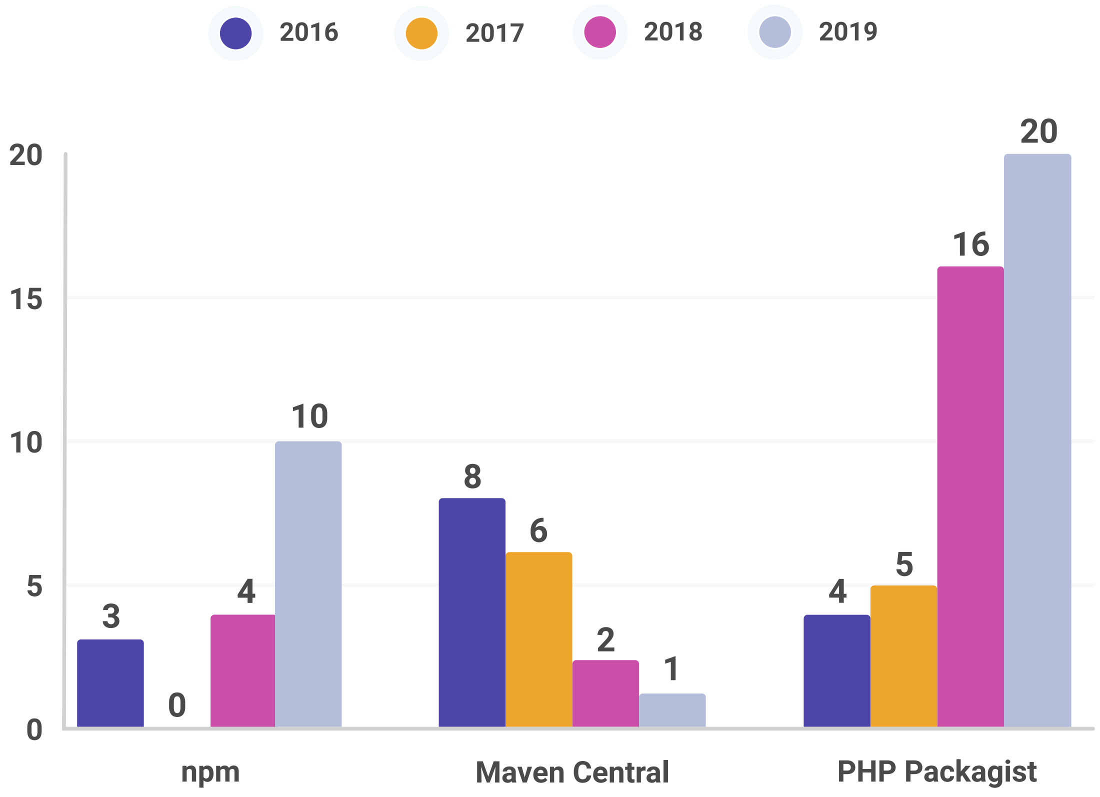
SQL injection

It is interesting to notice that SQL injection is not in the top ten (let alone the top five). However, given the potential impact of successful SQL injection exploits—and some notable past breaches that leveraged SQL injection vulnerabilities—its absence from the top ten is potentially very encouraging. For that reason, we decided to take a look quickly to understand if the data paints the same happy picture as one might expect.

The reality of the data is a mixed set. The continuing reduction of SQL injection vulnerabilities in the Java ecosystem is certainly a very good sign. However, not so when it comes to PHP where the number of new vulnerabilities is increasing.

While the overall numbers are still very low, this could point to a trend that will be worth watching in 2020. In a somewhat encouraging sign, other ecosystems we investigated did not have significant numbers of SQL injection vulnerabilities reported, and therefore, were not included in our analysis.

New SQLi vulnerabilities by year



3

Addressing infrastructure and container security risk

Containers have become almost ubiquitous in cloud environments, especially within DevSecOps pipelines. The distinct advantages of running software in dedicated, often Linux-based containers have given developers more flexibility and control over the deployment of their applications and microservices. Orchestration with Kubernetes has further enhanced those advantages by creating operational environments that are easily launched in highly resilient configurations.

Of course, as is the case with all technologies, containers have introduced their own unique challenges and threats from a security perspective. As the open source community creates and shares images and Kubernetes configurations, vulnerabilities that exist within them become a part of our operational environments. Last year we highlighted some key concerns in this space, so for this year, we revisited those findings to see how things have changed.

Security in Linux distributions

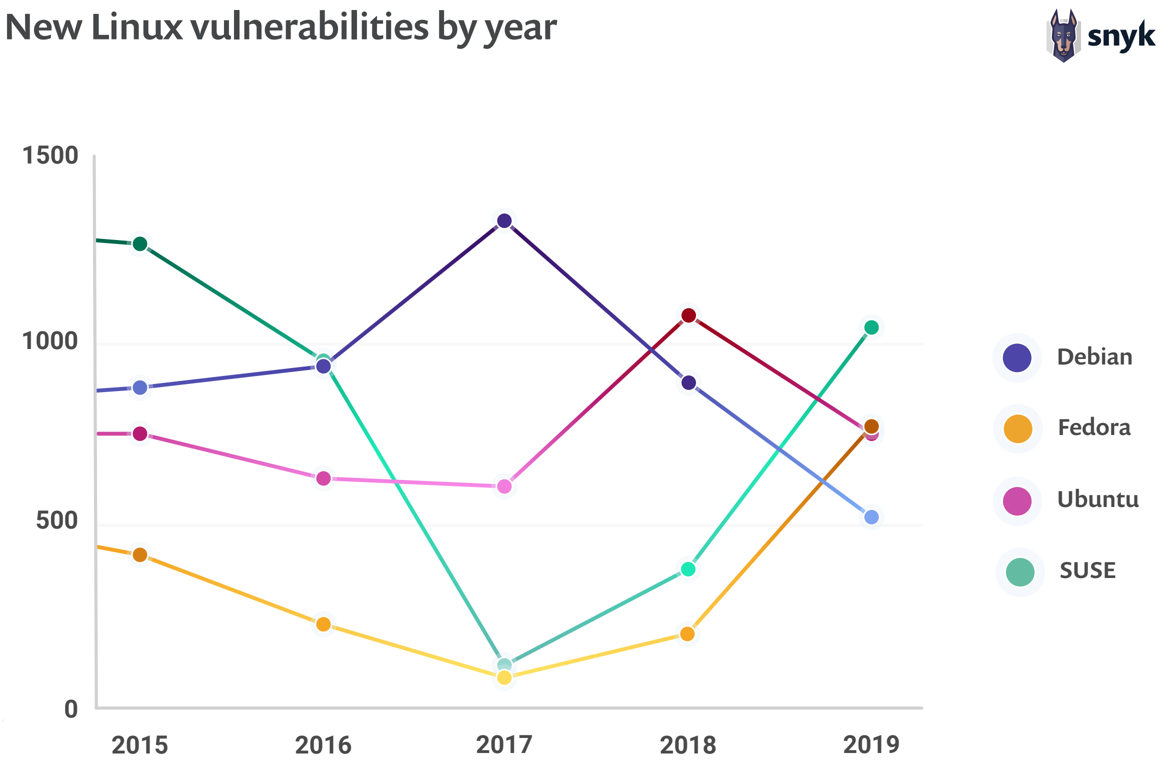
Most popular container images are based on some flavor of Linux meaning that vulnerabilities that impact popular Linux distributions have a direct impact on the security posture of the containers that leverage them. For this year's report, we turned to the MITRE CVE Database and looked at the number of reported vulnerabilities that impacted four of the more popular Linux distributions. Unfortunately, we had to omit RedHat from this particular analysis as a result of limitations of the data that we could retrieve from the CVE database.

New vulnerabilities impacting Debian have shown a significant decline over the last two years, after peaking at over 1,300 in 2017. Where the reverse is true for Fedora and SUSE whose count of new vulnerabilities has grown over the past couple years. The important lesson from these numbers, however, is not really in the trends.

The overall number of vulnerabilities being reported each year serves as a reminder of how diligent operations and security teams need to be in monitoring what Linux distributions are in use across their environment.

Obviously, with developers pulling, customizing, or in some cases building container images, this is an increasingly difficult task. This highlights the need for tooling and processes that can comprehensively identify and monitor for new vulnerabilities in all installed libraries within each container image.

New Linux vulnerabilities by year



The security cost of open source

Guest analysis by Vincent Danen, Senior Director, Product Security at Red Hat

Red Hat

As one of the largest commercial vendors of open source software, Red Hat is aware of the many security challenges around consuming and using open software. In 2019, there were over 1400 vulnerabilities reported in Red Hat Enterprise Linux alone [1]; for reference that is versions 5 through 8, and version 8 ships with over 2300 packages itself[1]. This is why Red Hat puts considerable effort into the security tracking, triage and assessment of those packages shipped in our products. It narrows the aperture for Red Hat users to focus on those third-party components someone might install on top of our platforms, meaning there is less to track, triage, and understand.

Open source software doesn't sit idle. Every day there is new software created that can be used to extend or enable new functionality. This is the beautiful thing about open source—it continues to grow and improve, allowing consumers to build more useful and sophisticated technologies with reduced overhead in development. There is a cost though, and that is diligence. With the rate of change and improvement to open source software, that new code being created every day has the potential to introduce new vulnerabilities.

You see that in communities that explode in popularity, such as npm for example, or other communities that undertake audits or begin to report more diligently on security issues. This is where you will often see surges or spikes in vulnerabilities in a particular ecosystem. It doesn't mean the software is bad or insecure, typically it means it's active as discovered vulnerabilities are often the sign of a healthy ecosystem.

[1] https://www.redhat.com/cms/managed-files/rh-2019-risk-report-overview-f21332wg-202003-en_0.pdf

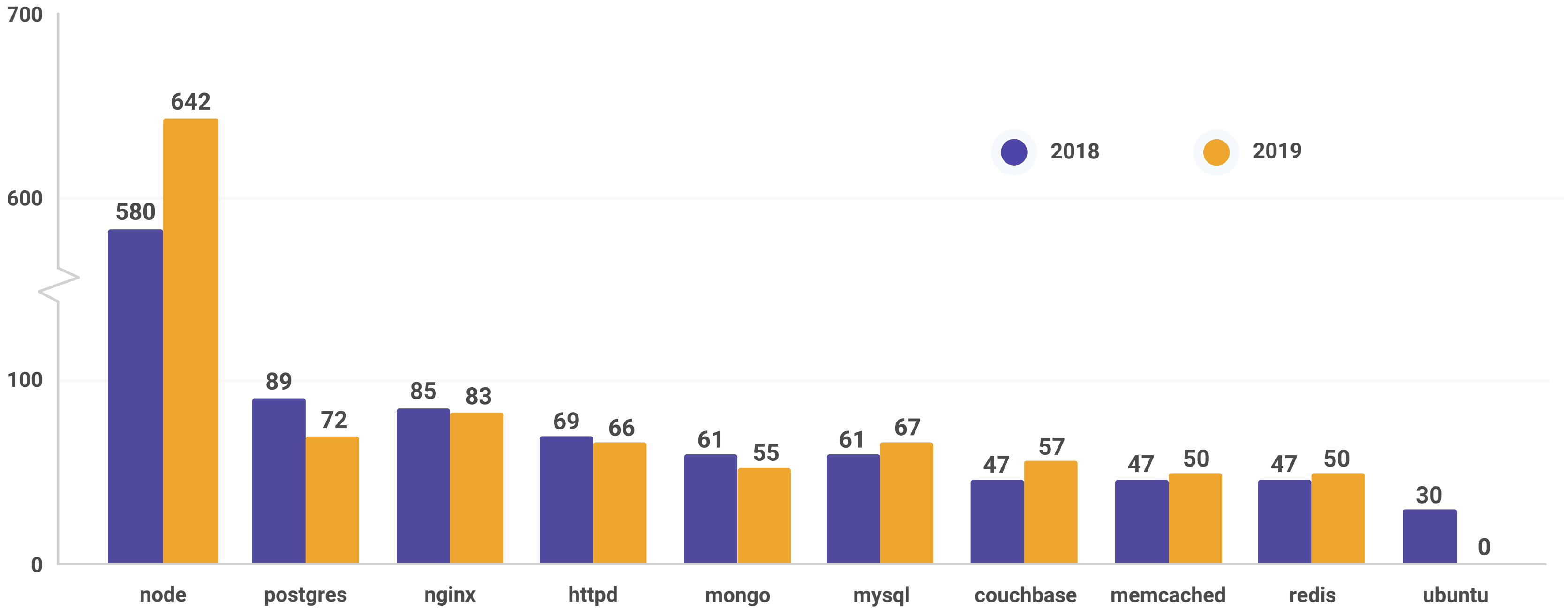
Docker images

Exploring deeper into the open source container community, we looked at the security posture of 10 of the most popular container images on Docker Hub. These official images are some of the most often pulled, and provide developers with a quick and easy way to run those applications.

The most popular images tend to be those tagged "latest" and these images are often designed to handle the broadest set of use cases for a particular language or application. With that in mind, they often come with a range of packages and development tools that make it very easy to install and run your code and all its dependencies.

Last year's analysis showed that these popular base images had many vulnerabilities, and so, in revisiting the analysis this year, we wanted to make the comparison to see how things had changed. We pulled the image tagged "latest" for each of these containers and scanned them using Snyk.

Vulnerabilities in official container images



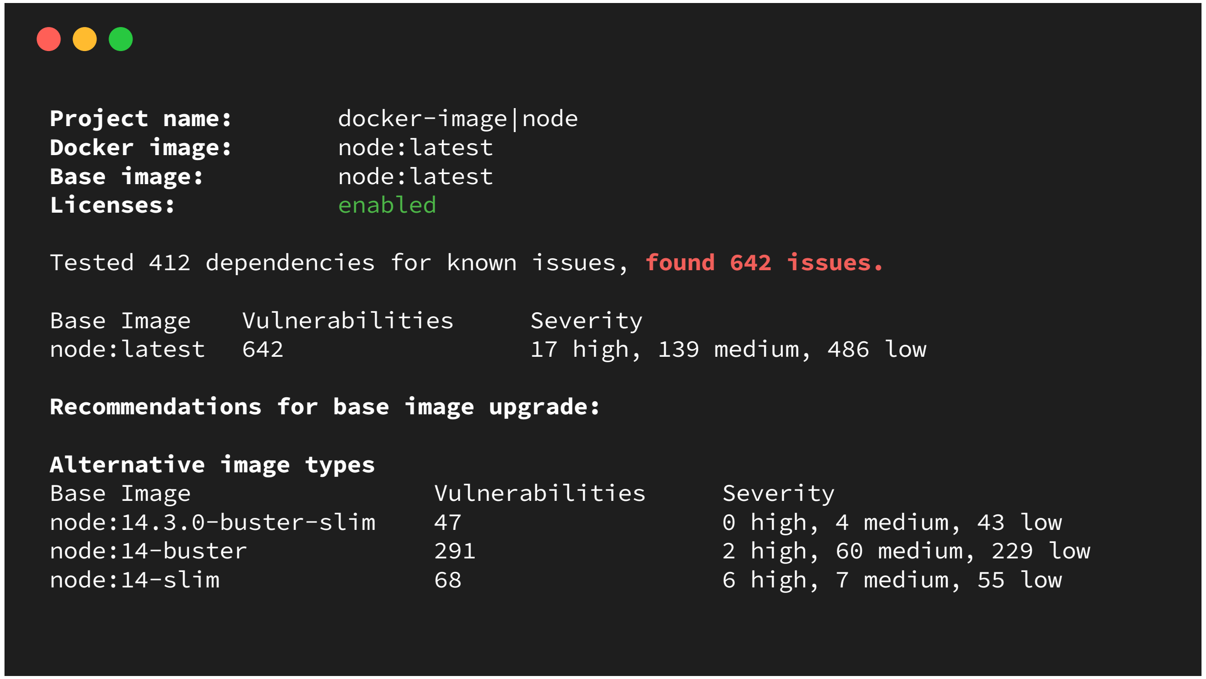
The results closely matched what we discovered last year and, in some cases, the results showed even more vulnerabilities this year. The latest node image (14.3.0-buster at the time of our analysis), for instance, has 642 known vulnerabilities.

Breaking it down further, the base image contains 17 high and 139 medium severity vulnerabilities. The high severity vulnerabilities stem, in large part, from the version of Debian on which the image is based and span many different libraries, including image processing and database connectors. All of these included packages likely make the "latest" image easy to use, but may not make it the best to use.

Further analysis shows that slimmer base images that include fewer libraries also have fewer overall vulnerabilities. For instance, the node base image 14.3.0-buster-slim has only 47 vulnerabilities, of which 0 are high severity and only 4 are medium severity since it does not include many of the vulnerable libraries that are in the 14.3.0-buster image.

These findings mimic what we saw in last year's report. For developers, the implications are pretty clear.. It is important that developers are aware of the risks that are inherent to using images retrieved from public repositories, even when they are listed as official images.

And developers should follow container best practices, which include using slimmer base images; ensuring you rebuild your images when the official images get updated so you get the latest security fixes; and using processes like multi-stage builds that can help separate development packages from production packages and slim down the final image in an automated fashion.



```
Project name:       docker-image|node
Docker image:       node:latest
Base image:         node:latest
Licenses:           enabled

Tested 412 dependencies for known issues, found 642 issues.

Base Image    Vulnerabilities     Severity
node:latest   642                 17 high, 139 medium, 486 low

Recommendations for base image upgrade:

Alternative image types
Base Image                 Vulnerabilities     Severity
node:14.3.0-buster-slim    47                  0 high, 4 medium, 43 low
node:14-buster             291                 2 high, 60 medium, 229 low
node:14-slim               68                  6 high, 7 medium, 55 low
```

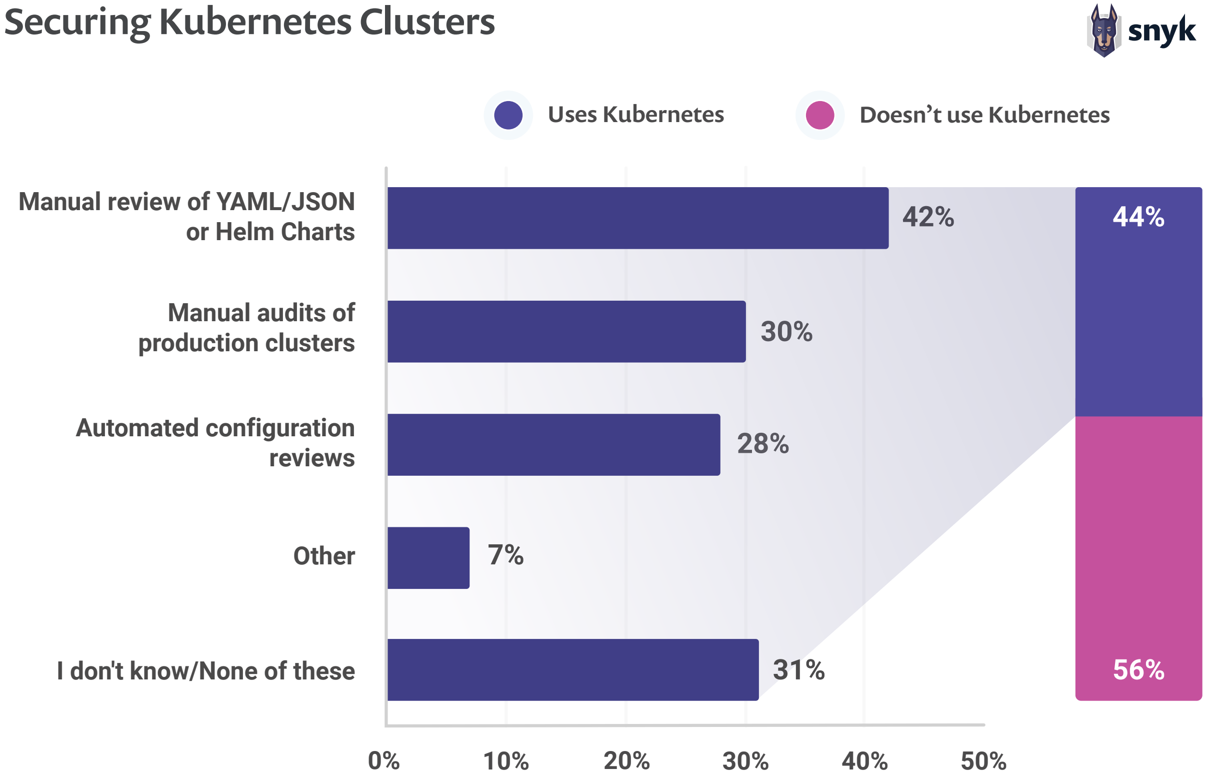
Kubernetes security

As many organizations look to leverage container-based infrastructure solutions, using Kubernetes to orchestrate and manage those containers is a likely next step. In our survey, over 44% of the participants indicated that they are currently using Kubernetes to orchestrate their containers.

But how are they ensuring the security of their Kubernetes manifests—we wondered. So we asked! Only 28% of those that use Kubernetes stated they have any form of automated tooling to review the configurations of their Kubernetes clusters. Worse, over 32% said they didn't know or don't have any practices in place. Manual configuration reviews of their YAML/JSON manifests or Helm Charts was the most common practice.

44% of survey participants indicated that they use Kubernetes to orchestrate their containers.

Securing Kubernetes Clusters



There are numerous key configuration decisions that can be made when defining a Kubernetes cluster that have a direct impact on the security of that cluster. The impact of failing to implement key controls can also be felt in terms of the costs associated with cloud environments in which these clusters are hosted. Validation that these configurations are in place can be easily achieved by automated reviews of the configuration files before deployment in a production environment. Additionally, tooling is available that can analyze active clusters for insecure configurations as well. Given the potential impact of insecure configurations, the importance of those reviews cannot be overstated.

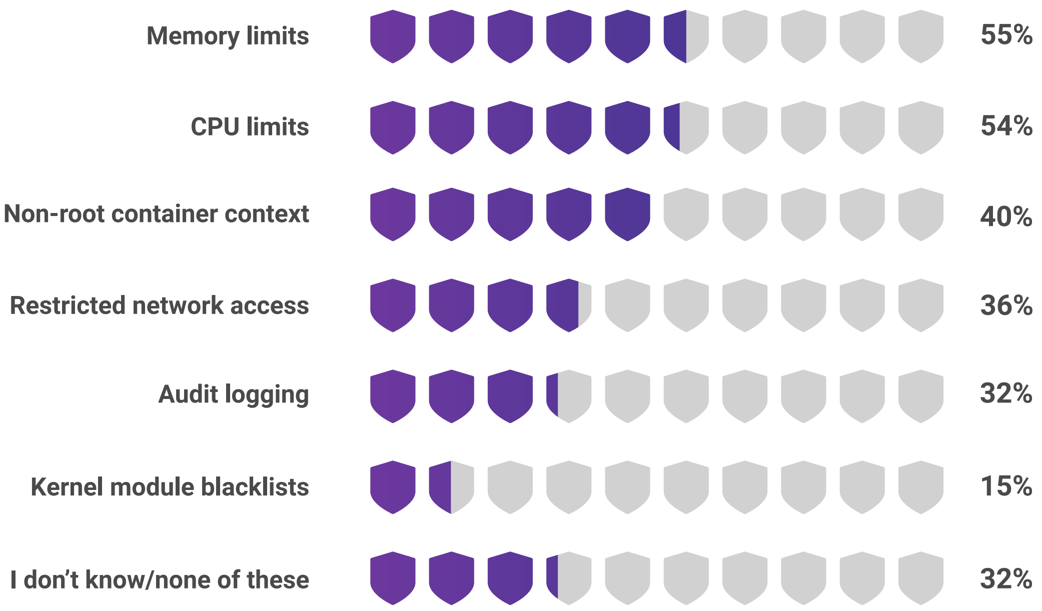
In our survey, we asked the participants about some of the more common configuration strategies that can be employed to help secure containers managed via Kubernetes. Of those that reported they use Kubernetes over 50% reported using both memory and CPU limits to manage their containers. However, restrictions on the use of vulnerable or unnecessary kernel modules were not very common.

Audit logging was also not particularly common. Nearly 32% reported they were not sure or did not leverage any of the possible controls we asked about.

Less than 40% of survey respondents verify common security-related Kubernetes configuration options.

The results seem to indicate a greater focus on the aspects of the configuration that affect availability and capacity while the more security-related features receive less attention. This would mirror a common progression of behaviors in new technology when the security of the technology is not made a primary concern early in the adoption of that technology.

Kubernetes resource controls



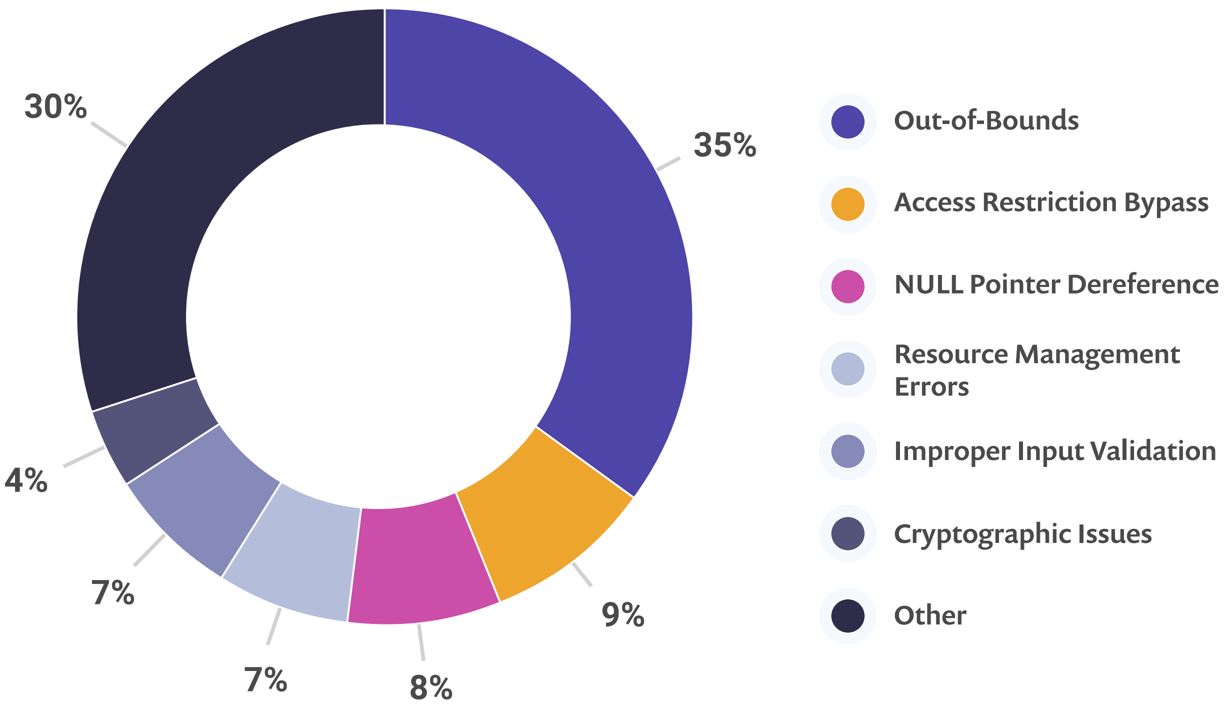
Multiple responses allowed.

Helm security

Helm is the most popular package manager for Kubernetes. As part of moving to Kubernetes, many organizations use Helm as the tool for deploying in-house or third-party applications. In our survey, over 40% of the respondents who said they use Kubernetes also indicated that they leverage Helm.

But what are the security implications of using Helm? A common risk that must be considered when installing a third-party Helm chart is the potential of introducing a vulnerable image in your cluster. In our 2019 Helm report, we found that 68% of stable Helm Charts contain an image with a high severity vulnerability. Here are the different types of vulnerabilities we found:

Helm vulnerability types



4 Security and vulnerability management practices

As development ecosystems evolve and the use of open source packages becomes increasingly prevalent in software development, organizations need to employ key strategies for how they become aware of and react to vulnerabilities. Analysis of the state of open source security would be incomplete if we did not consider how organizations are addressing the threats that the use of open source introduces.

Security as a culture

Software development has evolved considerably over the past decade. The adoption of DevOps/DevSecOps software delivery pipelines has forced software development organizations to think very differently about the development lifecycle. In particular, security practitioners have been challenged with ensuring secure practices while not inhibiting the improved efficiency of software delivery that sits at the very core of the DevOps model.

In DevOps/DevSecOps software delivery we have multiple motions. Developers pushing further right into the delivery pipeline, owning tasks from development through deployment. The ability to define the infrastructure on which their software will run via containers and other code defined infrastructure has enabled this transition. Meanwhile, as has been the case for decades, security continues pushing further left. Bringing security considerations to the development process early is recognized for its ability to, not only reduce risks but costs as well.

Finally, the operations are being driven to move up higher in the infrastructure stack as a result of increased use of virtualization technologies, software-defined infrastructure, and orchestration platforms.

As technology changes and development accelerates, delivering secure software requires a culture that emphasizes shared responsibility for the security, stability, and efficiency of the software. In last year's report, we asked survey participants to identify who is responsible for security in their organization. As one might expect, 81% identified developers as having a responsibility for security. Surprisingly, however, only 28% identified the security team as having responsibility, while only 23% said the responsibility lies with operations. Perhaps cynically, 12% of the participants said the responsibility to security belongs to nobody.

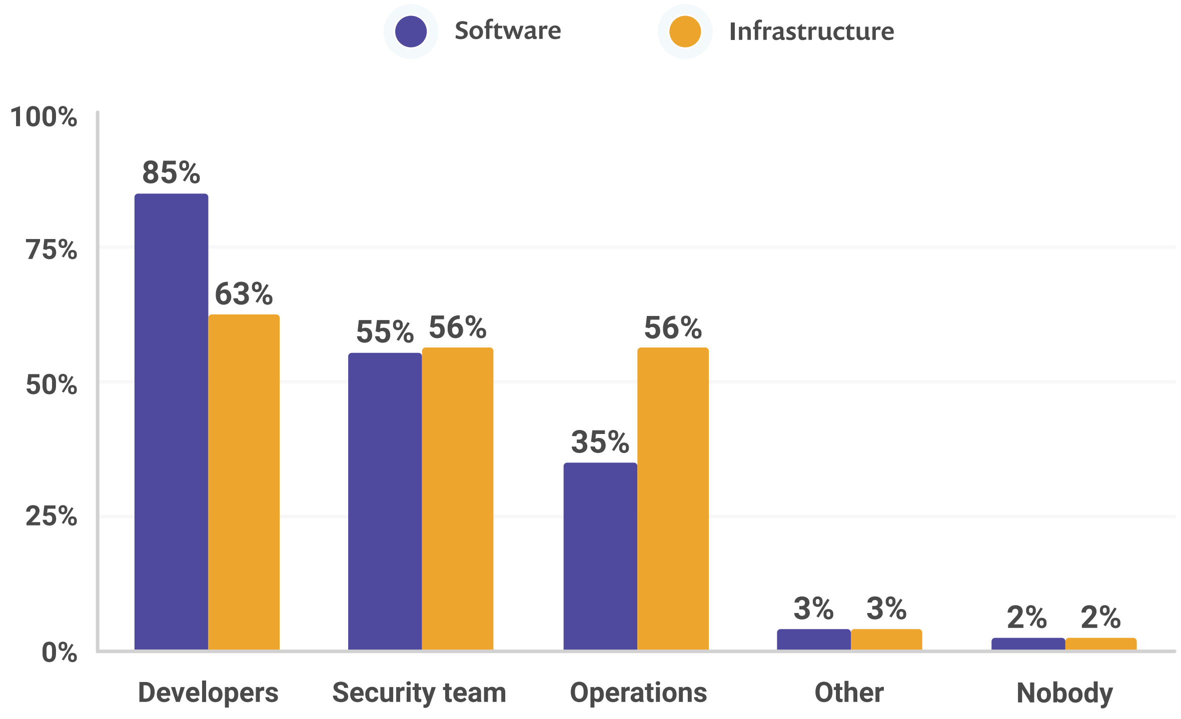
This year we decided to dig a little deeper into how organizations are doing in driving a culture shift that embodies security as a core responsibility for all members of the organization. We asked our survey participants who they felt should be responsible for designing and implementing security controls in their software, as a multi-answer question. The results were more encouraging this time around but curiously a lot of weight continues to be put on the developers' shoulders.

Not only is it not fair to ask developers to shoulder so much responsibility for software security, it is also counterproductive. Developers need to be enabled to do their part, but security teams need to drive that enablement, and operations needs to be a part of actively monitoring it as well. So, while this year's results are encouraging and do show growth, there is clearly more work to be done in this area.

When we asked the participants about the security of their infrastructure, however, we found that the results were much more balanced—Operations teams were commonly identified but so were Developer and Security teams in almost equal numbers.

However, the fact on this multi-answer question the responses were all less than 65% still indicates that respondents did not typically identify all three groups as being responsible. Again, this indicates that more work can be done in shifting toward a shared-responsibility culture.

Who should be responsible for security?



Multiple responses allowed.

Securing the pipeline

Guest analysis by Ron Powell, Technical Content Marketing Manager at CircleCI

circleci

Continuous integration (CI) is the foundation of software development. CI defines the automated steps for building, testing, and deploying software. It is essential for developing software in a time when constant change is the norm.
CI provides the ability to automate in minutes what was historically a process requiring manual approval steps that could take months to execute.
Teams using CI measure the success of their development process by four key metrics:

- **lead time for changes**
- **deployment frequency**
- **mean time to recovery**
- **change fail percentage**

All of these metrics reflect an emphasis on speed. Speed is incredibly valuable in software delivery, but should it come at any cost? Speed without reliable, consistent quality is not helpful. And speed without security is even worse.

We noticed two things from Snyk's State of Open Source Security Report 2020 that stood out to us. The first is that, increasingly, survey respondents feel that security for software and infrastructure should be shared among development, security, and operations teams. This cultural shift in the ownership of security is represented in the shift from DevOps to DevSecOps and CI is the centralized place where these teams are able to come together. By adding security to DevOps, teams are able to reap the benefit of speed that comes with automation, and they don't have to sacrifice security in the process.

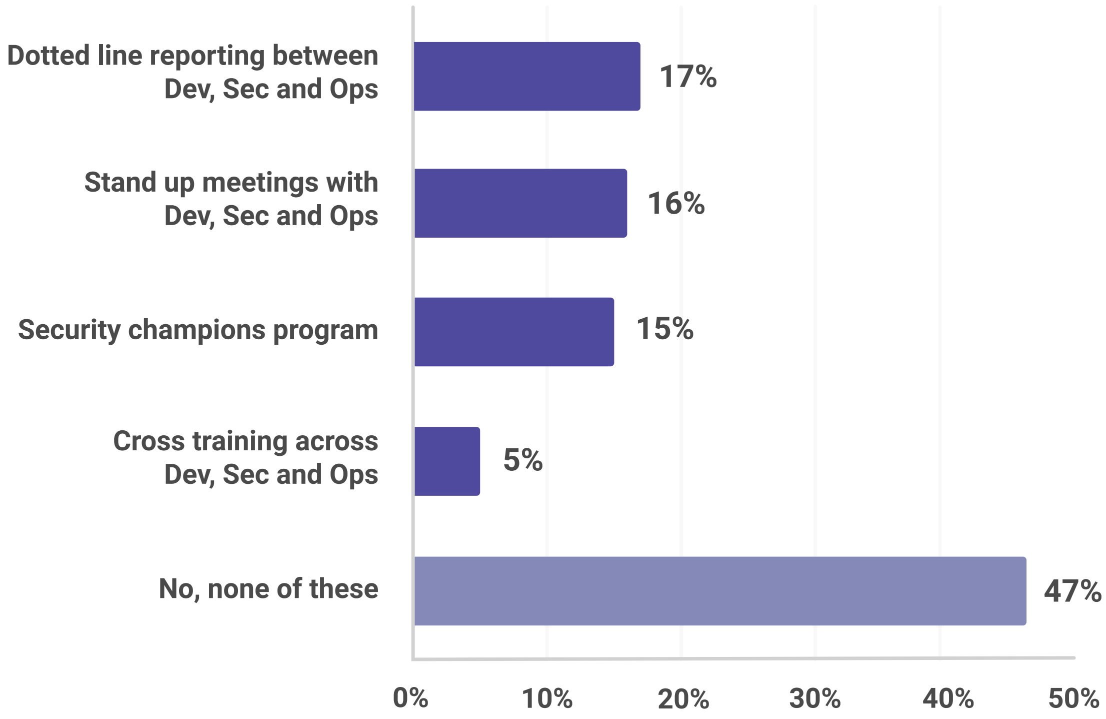
The second thing that stood out to us was the number of vulnerabilities in official base images. Containers are a central technology for CI. In a CI pipeline, when a codebase is updated, the applications are run in clean containers that use images that contain all the tools and packages needed for the app. To benefit from CI, managing vulnerabilities in images is an absolute necessity. Even for organizations that create their own custom images, Snyk's State of Open Source Security Report 2020 has identified that official base images—the image from which custom images are created—have many vulnerabilities. The official Node image with the latest tag has almost 700 known vulnerabilities!

While speed is certainly a valuable metric to consider when developing software, consistent quality and security are also necessary for ensuring that the software we develop meets the expectations we set for ourselves and the expectations of our users.

There are many different approaches that organizations may use to create a culture of shared responsibility. One of the more commonly discussed approaches is establishing a Security Champions program. The goal of such a program is to bring security expertise to the development organization. These programs establish roles in the development team to create a community of security-minded developers. Security Champions programs are even referenced as security practices at the base maturity level in the OWASP SAMM model.

We asked our survey participants about some of the more common approaches to establishing this type of culture and bringing security into the development conversation. Despite the increasing adoption of DevOps/DevSecOps software delivery, a staggering 47% of organizations indicated that they have not implemented any of these practices.

Programs to drive shared responsibility culture



Multiple responses allowed.

Creating a culture of shared security responsibility at Segment

Segment

On a recent episode of The Secure Developer podcast, Leif Dreizler and Eric Ellett talked about the importance that customer data platform provider, Segment, places on the collaboration between development, security, and operations resources. Segment does not do sprints across their organization—instead, teams operate independently. However, through a consultative model, the security team is still integrated early in the development process to provide threat model and design review capabilities.

At Segment, the idea of establishing empathy is baked into the culture of the security team. Within the team, Segment has employed a concept of "Walk a mile in the developer's shoes". Ellett explained that the security team goes to great effort to understand how their security processes impact other areas of the organization. For instance, when they sought to roll out Multi-Factor Authentication, Dreizler spent a quarter embedded with the development team. This provides the security team with invaluable context on the challenges that the development team faces in terms of what they are trying to protect.

However, the collaboration focus doesn't end there. Ellett explained that there is the intention that similar initiatives would happen in the other direction. The plan is to bring people from other areas of the organization to sit with the security team and understand their world as well. Dreizler stated, "I think that this is what the goal of DevSecOps should be. Similar to DevOps, where you have operations people learning how to code and now everything at Segment's infrastructure is code."

Segment also firmly believes in creating the "paved road". A guiding principle that the Segment security team operates under is "would this tool be used by the developer?". In other words, there is a keen focus on ensuring that the adoption of security controls is enabled by the ease of use. The ultimate focus, according to Dreizler, is "just make it as easy as possible for people to do what the right thing is."

Through this cooperative and empathetic approach, Segment has been able to grow a strong culture of collaboration in their organization—an example of how the promise of DevSecOps can be realized by ensuring all functions are aligned in their goal to do what is right for the organization.

Evaluating package health

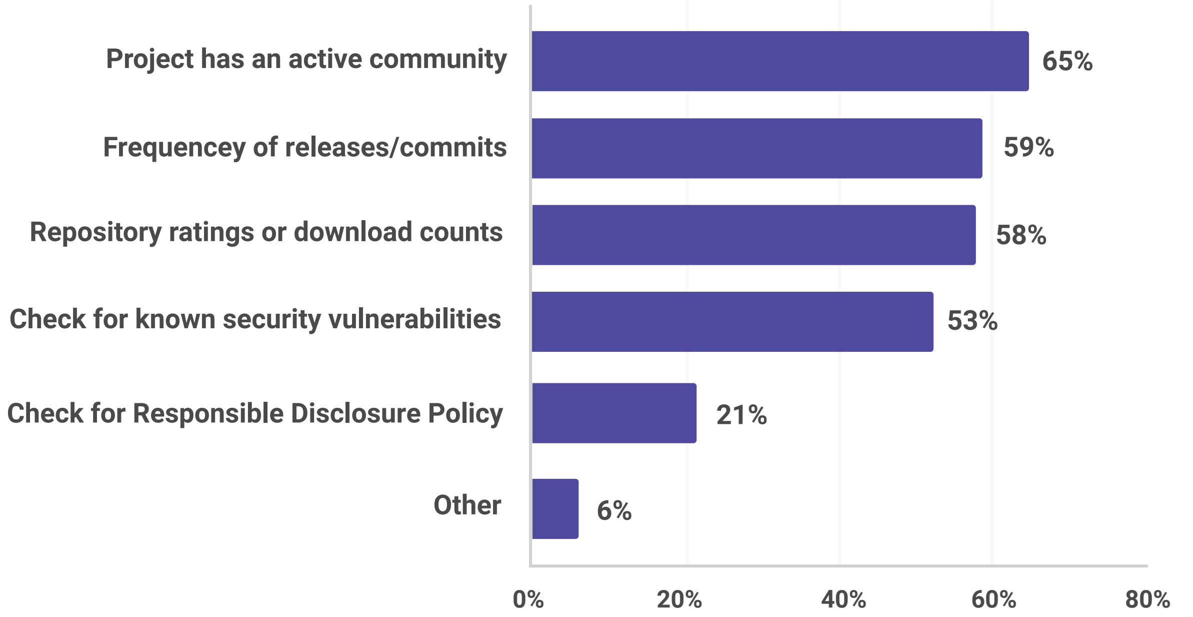
A topic of growing conversation across the open source community is how to determine the health of packages. Understanding how actively and attentively a package is being maintained and updated timely with security fixes sounds like an easy task. However, when the possible metrics for measuring health are considered and analyzed, it becomes clear that this is a harder question to answer than it would seem.

The various code repositories have attempted to help with this and do offer some useful tools that can provide some indication of which packages are trustworthy. Publishing things like issue counts, revision histories, pull request details, and even user feedback mechanisms all provide some level of reassurance. However, none of these by themselves tells a reliable story. For instance, while frequent updates to a package can be an indication of an actively maintained package (which is a good thing), excessively frequent package updates could have a negative impact. Maintenance of applications that are dependent on that package could be complicated by having to initiate more frequent changes to apply the latest versions of the updated package. This is of particular concern where the updates introduce security-related fixes that need to be applied in a timely fashion.

In our survey, we explored what factors are commonly used to evaluate and ultimately select open source packages.

Our results were also consistent with the idea that there is no single generally accepted answer to this dilemma. It is, however, encouraging to see that numerous practices are being adopted and most participants indicated that they use more than just one factor on which to base their decisions.

How do you vet open source packages?



Multiple responses allowed.

Container image health

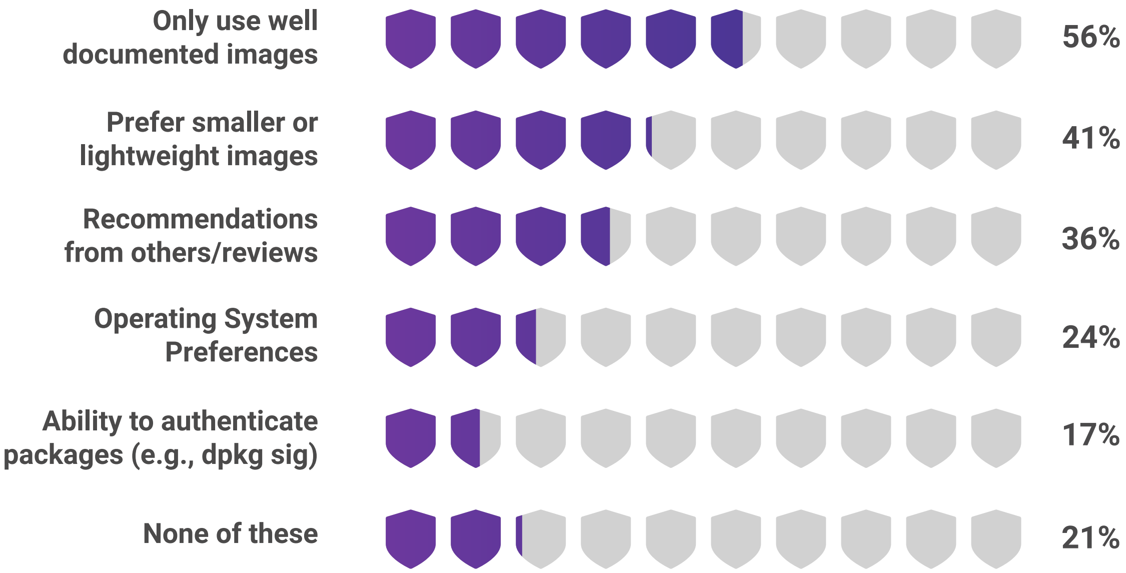
Earlier we discussed some of the vulnerabilities found in the most popular base images on Docker Hub. So, with more and more organizations turning to container technology, how are organizations going about ensuring the base images they select are secure?

Many of the same complexities and considerations that affect decisions about package health also come into play with container health. As we saw earlier, simply being labeled as an "official image" on Docker Hub does not mean there are no vulnerabilities. What about feedback from others? Do ratings offer reliable measures? All of these issues and more need to be considered when attempting to select a secure image for your next deployment.

We asked the respondents to our survey about the potential criteria they use when selecting base images. The use of well-documented images and smaller lightweight images were the two most common practices.

Good documentation assures that developers can understand what components are included in the image which in turn enables the selection of an image with only the appropriate modules necessary to support the code that will run in the container.

How do you evaluate container security?



Multiple responses allowed.

Security practices

Almost every software development organization understands that some level of security focus needs to be included in the development process before code is deployed. There is a wide variety of practices that security practitioners recommend and different organizations may implement one or many of them.

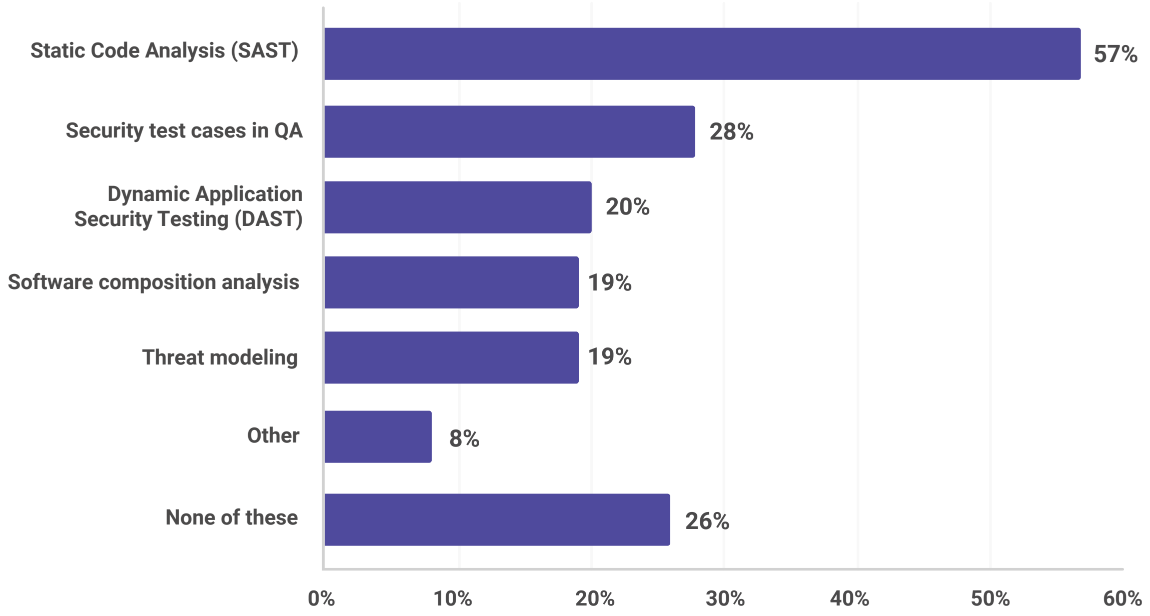
From a security perspective, each practice serves a different purpose and so the recommendation is to implement all of them. However, the reality is that the decision to build a particular security practice into the pipeline is complex and involves not only organizational maturity and risk tolerance, but also the ability of the delivery model to adapt to these activities without introducing obstacles to development.

The mantra of pushing left—that has been a part of the application security vocabulary for many years—often focuses on the ability to reduce friction and cost by identifying vulnerabilities early in the development process. To that end, it isn't too surprising to see that the use of Static Application Security Tools (SAST) is the most common activity across our survey participants.

Over 1 in 4 responses indicated no common security practices are in place.

There are two concerning numbers in these results though. First is that almost 26% of the participants indicated that they do not have any of the listed security practices implemented in their delivery pipeline. With security continuing to be a top-of-mind concern for executives across most industries, it is surprising that over 1 in 4 responses indicated no common security practices are in place. Second is the relatively low use of Software Composition Analysis (SCA). Earlier we detailed the risks that open source dependencies, and in particular indirect dependencies, post to software development. SCA is a powerful way to understand the dependencies of an application, identify if there are vulnerabilities in those dependencies, and enable monitoring of future vulnerabilities in an organization's software.

Security practices in the delivery pipleline



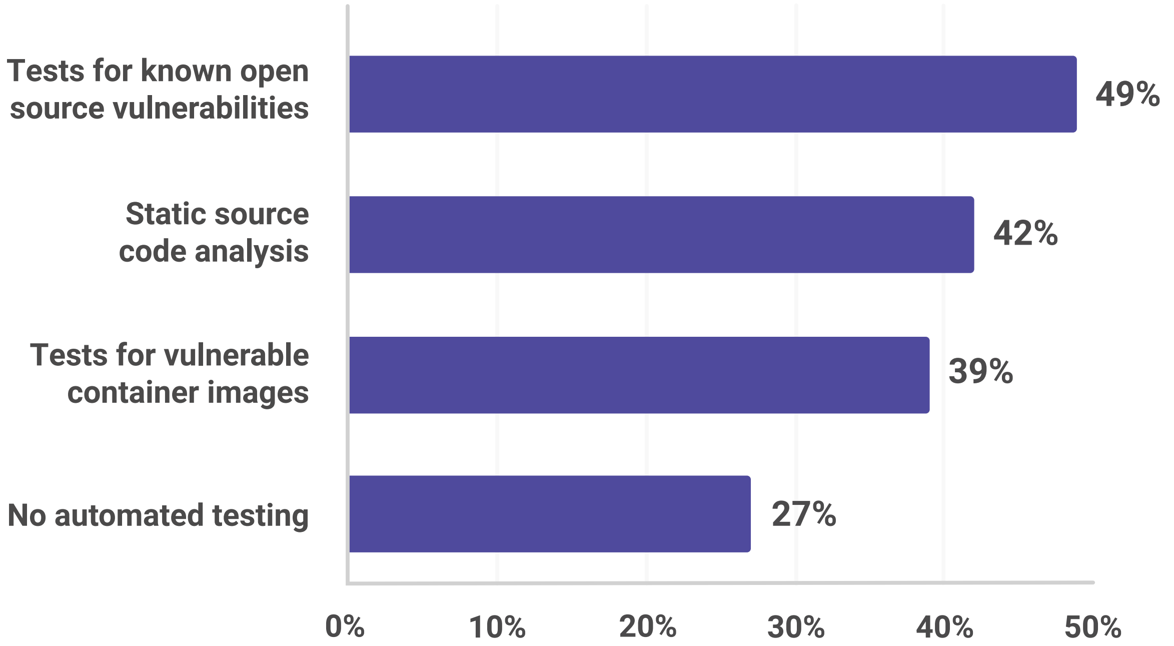
Multiple responses allowed.

As organizations adopt DevSecOps, automation and tooling become key topics to help enable efficient delivery. In particular, when attempting to shift security from being a gate between delivery stages to being integrated into those stages, automation becomes crucial. Automated tools help eliminate slow feedback cycles that create friction for development and fail in adoption because they are simply too inhibitive of efficient development.

In our survey, we asked our respondents whether they had enabled any automated security testing capabilities in their delivery pipelines. The results are encouraging in that many identified multiple capabilities. However, at the same time, over 38% indicated they have no automated capabilities. In light of the 26% who stated they have no security practices in place, this number isn't too shocking.

38% of survey participants have no automated security capabilities. 26% have no security practices in their pipelines at all.

Automated security testing capabilities



Multiple responses allowed.

Vulnerability remediation

It seems obvious, but it is not simply enough to identify vulnerabilities in software—timely remediation of vulnerabilities plays a key role in reducing overall security risk. When it comes to open source, however, remediation can be a more complex issue.

When vulnerabilities occur in the software you've developed, the answer to remediation is simply to fix the code. However, when the vulnerability is in a direct open source dependency, the answer may be more complex. There may be an updated version of that package in which the vulnerability has been remediated that you can include in your code—one-line change in your dependencies could be all it takes. However, if there's not an updated package with a fix, the options are a little more complicated. One option is to fix the code yourself. Hopefully, you would also submit the changes back in a pull request to the maintainer, thus making the package more secure for all. Another option would be to open an issue with the maintainer and wait for them to make a fix.

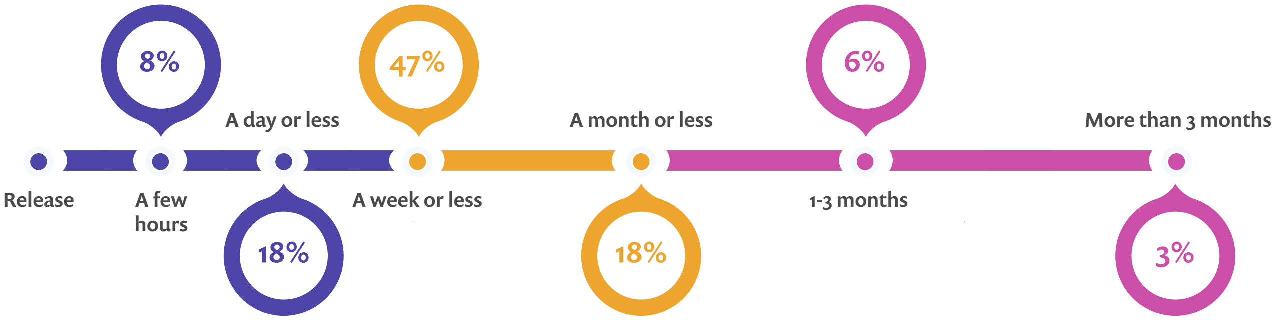
Of course, these decisions can get even more complicated when the vulnerability is in an indirect dependency.

The point of this discussion is not to bemoan the complexity and risk of open source security, but rather to understand that these various factors and approaches to remediation can have a direct impact on how quickly organizations are able to respond to a vulnerability.

Open source vulnerability expectations

In our survey, we asked participants to share with us what their expectations are for package maintainers to address security vulnerabilities within their packages. Most said they would expect a fix in a week or less from the time the vulnerability is reported.

Expectation for open source vulnerability fixes



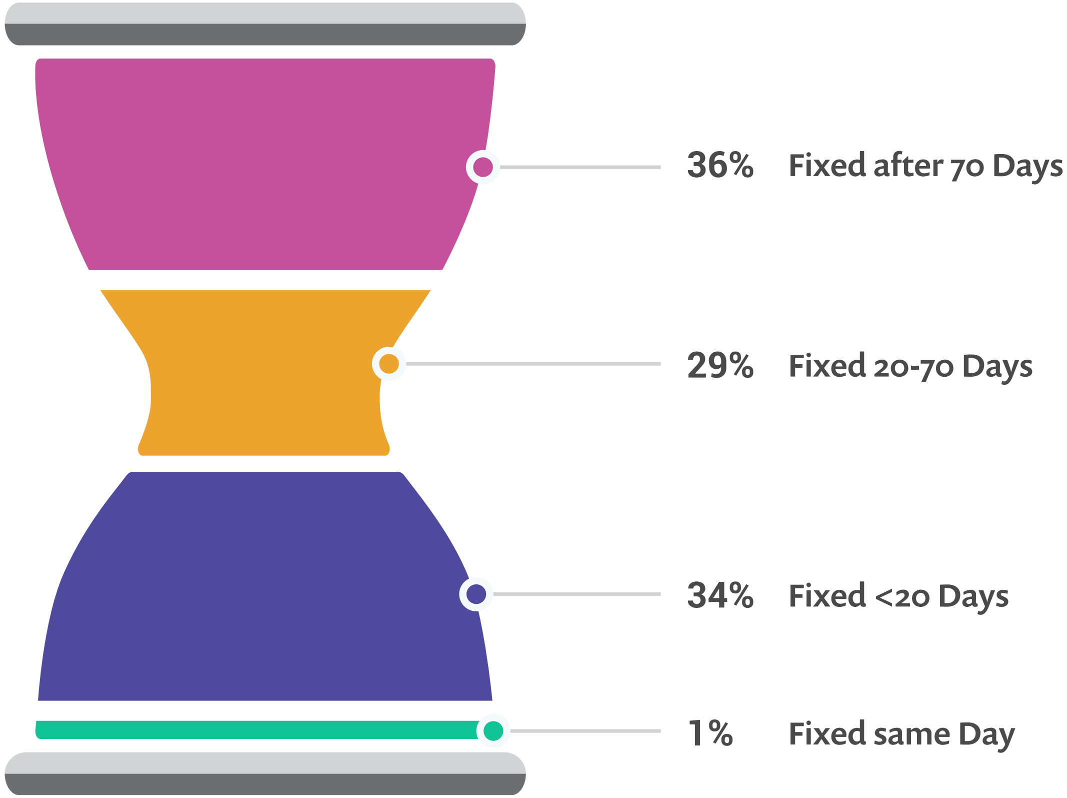
Remediation performance

The speed at which vulnerabilities get remediated once they are discovered is a constant concern of any mature vulnerability management program. Shorter vulnerability remediation timelines equate to reduced risk for the organization.

A little more than one-third of vulnerabilities are fixed within 20 days of being discovered

We investigated the monitoring of vulnerabilities that were discovered in projects scanned by Snyk to determine just how long it takes organizations to respond and remediate the issues once they're identified. Of course, as discussed previously, the speed at which a development team can remediate a vulnerability in an open source dependency is reliant on a number of factors, some of which are outside the control of the organization. Still, it is important to understand just how well organizations are able to react when a vulnerability notification comes through.

Vulnerabilities fixed in projects scanned



What is particularly interesting about the results is the percentage of vulnerabilities that are successfully remediated in under 20 days. For most vulnerability management programs, that is a very high success rate. It's also interesting to note that some vulnerabilities are even remediated on the same day. On average, however, most vulnerabilities take over two months to be remediated, suggesting that there is more work to be done. Seeing that some have gone as long 16 months before they are remediated, confirms that we can improve as a community in how we address security vulnerabilities.

However, it is important to note, when considering these numbers, that the severity of the vulnerability is not taken into account in this analysis. A common practice in vulnerability management programs is to set remediation timelines based on the severity of the identified vulnerabilities. Therefore, one can hope that those vulnerabilities that took significantly longer to be remediated are lower severity, and were therefore not prioritized for remediation as quickly as higher severity items. However, a much deeper analysis would be needed to confirm if that is the case and such analysis was beyond the scope of this study.

Maintainer security performance

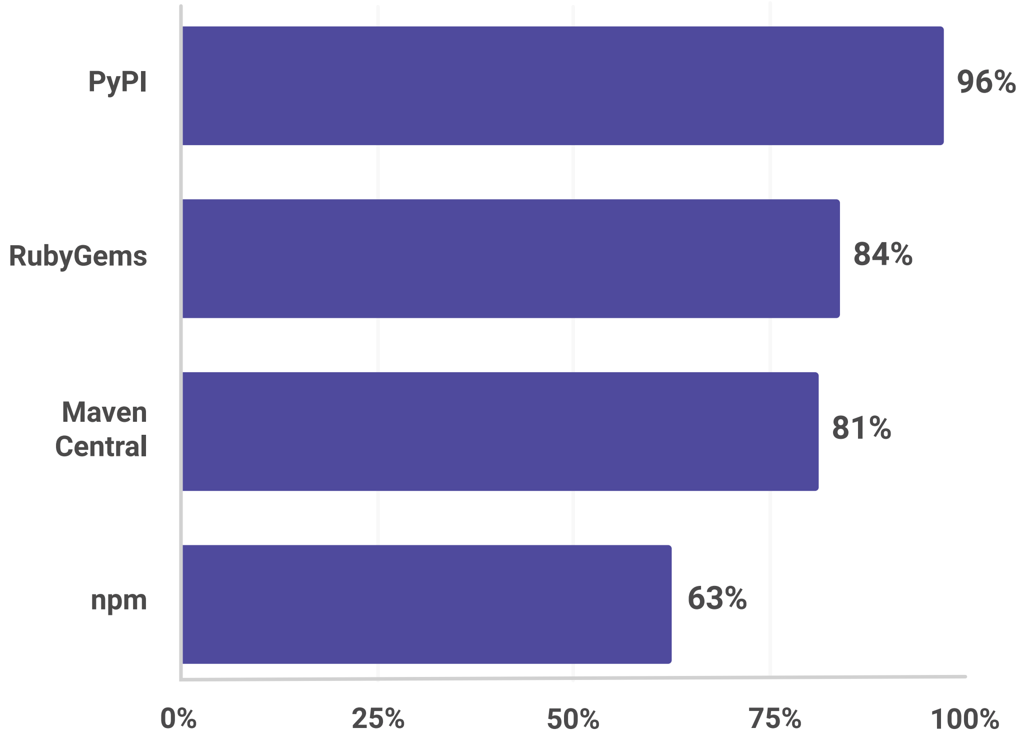
It is no secret that one of the greatest contributing factors to the state of open source security is the ability of maintainers to produce secure code and to remediate security issues quickly once they are reported.

Vulnerability remediation

Just as we did last year, once again we looked at the percentage of known vulnerabilities in key ecosystems that had been remediated in subsequent versions of the package. The results are a mixed bag of good and bad news. In the Node.js space, 63% of disclosed vulnerabilities have fixes available—up from 59% in the previous year. That is good news.

But what happened with Java? Last year Java boasted an impressive 97% of vulnerabilities addressed with known fixes. However, in this year's analysis, that number has dropped off considerably to slightly less than 81%. Python and Ruby for their part have remained relatively consistent year-over-year.

Packages with known fixes



Maintainer vulnerability awareness

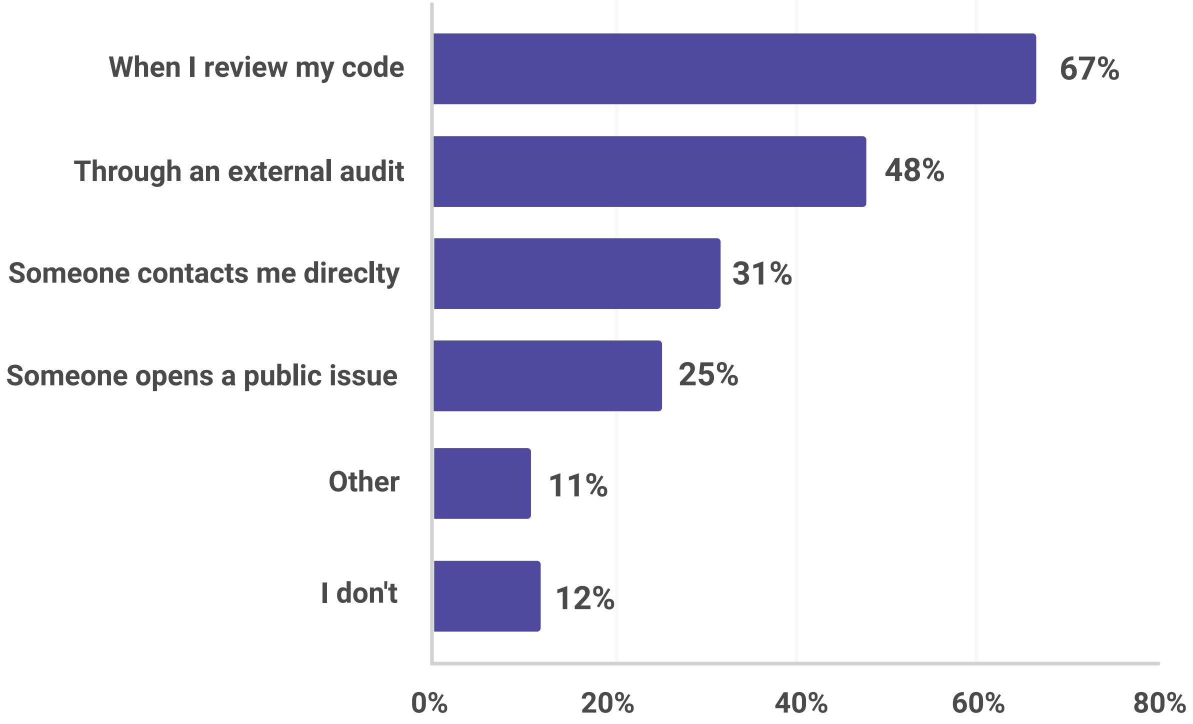
Perhaps some explanation for the numbers of vulnerabilities with available fixes can be found in an analysis of how maintainers become aware of vulnerabilities in their software in the first place. In the survey, we asked how project maintainers become aware of new vulnerabilities in their code.

There are many ways in which new vulnerabilities would be identified. Certainly, the preference would be that maintainers of the code identify the vulnerabilities themselves through some form of code review or analysis. Indeed, the majority of participants indicated that this is one way they find vulnerabilities. External audits were also a popular answer. Disclosure by external parties ranked lower on the survey perhaps suggesting that most maintainers are more proactive about discovering vulnerabilities.

Compared to last year's results for the same question, there are some positives and some negatives in this data. The percentage that indicated they would find the vulnerabilities on their own dropped from 72 to 67 percent. However, that may be offset by the number that indicated external audits as the source of information which climbed from 30% last year to almost 50% this year.

Also encouraging is the percentage of responses that stated they don't find out about vulnerabilities in their code. That number dropped from 17 to 12 percent this year.

How do you find out about vulnerabilities in your code?



Multiple responses allowed.

Report conclusions

While there were many interesting findings in this year's report, the following are the five areas that were the most notable in terms of their impact on the open source community.

- More than half of survey respondents view security as a shared responsibility across developers, security, and operations which is an improvement over 2019.
- New vulnerabilities are **down 20%** across the most popular ecosystems; npm saw the greatest reduction in vulnerabilities disclosed, yet retains the worst fix rate of the popular ecosystems.
- Vulnerabilities that have received attention for many years continue to be reported in high numbers; however, more complex and less understood vulnerability types had higher impact. Prototype pollution, for instance, affected over **25% of projects** scanned by Snyk in 2019.
- The top ten most popular official container images have significant numbers of known vulnerabilities. Pulling an official image is not a replacement for regular security hygiene.
- There are still significant improvements to strive for as many still don't treat security with proper urgency: a third of vulnerabilities in projects were fixed in under 20 days, but another third took **70 days or more.**

Recommendations

Based on the trends and themes identified in this year's report, there are some key next steps that organizations and package maintainers can take to improve the security of their software.

Open source maintainers

- Greater emphasis is still needed when it comes to **inventorying** open source. Creating greater visibility into the full dependency tree will drive proactive vulnerability identification as well as enabling more effective response to emerging threats.
- **Education** on new vulnerabilities and exploits for developers must be a priority to reduce the breadth of impact across multiple projects in new vulnerability types. Additionally, continued security hygiene and monitoring is clearly needed regardless of a package's perceived health and popularity.
- Establish and track **metrics** regarding vulnerability remediation to ensure that expectations and actual achievement can be reconciled.

Container security

- Pulling an "official" image does not guarantee that it is free from vulnerabilities. Regular **security hygiene** should be performed for any new container images used.
- **Minimize container image size.** "Latest" tags often pull the most comprehensive version of the image. However, in our research we found that using a "slim" image instead can reduce the number of vulnerabilities in the image by as much as 95%.
- Kubernetes environments offer standard **configuration options** that should be default for any new cluster launched. Limiting root level access, ensuring audit logging is enabled, and preventing the installation of known bad modules are key steps that can be taken.

Organizational security culture

- Security at **all phases** of the delivery pipeline should be seen as a shared responsibility across the organization. Establish clear and common goals that apply to developers, operations, and security personnel.
- Launch **formal programs** such as security champions, job shadowing, and daily task integration that drive empathy and understanding across Dev, Sec, and Ops disciplines.
- More comprehensive practices are needed in **DevSecOps pipelines**. Look to enable practices as early as user stories in the backlog and select automated security tooling that integrates with existing development and pipeline management.

snyk

Develop fast. Stay secure.

Report author

Alyssa Miller (@AlyssaM_Infosec)

Report editor

Eirini Eleni Papadopoulou (@Esk_Dhg)

Report contributors

Simon Maple (@sjmaple)
Ron Powell (@whyDoMy3y3sHurt)
Vincent Danen (@vdanen)

Report design

Growth Labs (@GrowthLabsMKTG)

Office info

London

97 Hackney Road
London E2 8ET

Tel Aviv

40 Yavne st., first floor

Boston

200 Berkeley, 24th floor
Boston, MA 02116