The state of open source security report

2019

snyk

Table of contents

00 State of open source security report 2019

An introduction to this report 3
TL;DR - The state of open source security 2019 report, at a glance 4

01 The open source landscape

Adoption 6
Risks and impact 9
Indirect dependencies 10
Security posture of open source maintainers 11
Security audits 12

02 Known open source vulnerabilities

Known vulnerabilities in application libraries 14
Trends in severity 16
Spotlight: Zip Slip 17
Known vulnerabilities in system libraries 18
Known vulnerabilities in docker images 20
Vulnerability differentiation based on image tag 22
Spotlight: Malicious packages 24

03 Vulnerability characteristics of each ecosystem

XSS vulnerabilities 27
SQL injection vulnerabilities 28
Sensitive information exposure 29
Regular expression denial of service 30
Path traversal 31
Cleartext transmission of sensitive information 32

04 The open source security lifecycle

Discovering vulnerabilities 34
Open source security ownership 35
Finding out about vulnerabilities 36
Spotlight: Vulnerabilities without CVEs 37
Time to adopt security fixes 38
How do maintainters find out about vulnerabilities? 39
Inclusion to disclosure 40
Spotlight: Equifax, a year later 41
Releasing fixes 42
Rate of fixing 43
Spotlight: Responsible security disclosures 44

05 The future of open source

Take action 46
TL;DR - Report summary 47

An introduction to this report

Adoption of open source software has continued over recent years, and in 2018 we specifically witnessed how enterprise organizations strengthened their stakes on open source software. For example, in 2018 alone, IBM acquired RedHat for $34 billion, further proving that open source software is becoming the foundation for the modern enterprise. Microsoft acquired GitHub for $7.5 billion, demonstrating the commercial opportunity in building tools for the open source community.

As adoption of open source software continues to grow rapidly, the risk of exposure to security vulnerabilities is also increasing.

To better understand the open source security landscape, and what we can all do to make it better, we gathered information from a number of public and private data sources including the following:

- a survey created and distributed by Snyk that was completed by over 500 open source maintainers and users.
- internal data from the Snyk vulnerability database, as well as hundreds of thousands of projects monitored and protected by Snyk.
- research taken from external sources published by various vendors and data gathered by scanning millions of GitHub repositories and packages on public registries.

Let's start by showing you some of the key takeaways from this report as a dashboard on the following page.

TL;DR - The state of open source security 2019 report, at a glance

Open source adoption

- Growth in indexed packages, 2017 to 2018
  - Maven Central - 102%
  - PyPI - 40%
  - npm - 37%
  - NuGet - 26%
  - RubyGems - 5.6%
- npm reported 304 billion downloads for 2018
- 78% of vulnerabilities are found in indirect dependencies

Known vulnerabilities

- 88% growth in application vulnerabilities over two years
- In 2018, vulnerabilities for npm grew by 47%. Maven Central and PHP Packagist disclosures grew by 27% and 56% respectively
- In 2018, we tracked over 4 times more vulnerabilities found in RHEL, Debian and Ubuntu as compared to 2017

Known vulnerabilities in docker images

- Each of the top ten most popular default docker images contains at least 30 vulnerable system libraries
- 44% of scanned docker images can fix known vulnerabilities by updating their base image tag

Vulnerability identification

- 37% of open source developers don't implement any sort of security testing during CI and 54% of developers don't do any docker image security testings
- The median time from when a vulnerability was added to an open source package until it was fixed was over 2 years

Who's responsible for open source security?

- 81% of users feel developers are responsible for open source security
- 68% of users feel that developers should own the security responsibility of their docker container images
- Only three in ten open source maintainers consider themselves to have high security knowledge

Snyk stats

- In the second half of 2018 alone, Snyk opened more than 70,000 Pull Requests for its users to remediate vulnerabilities in their projects
- CVE/NVD and public vulnerability databases miss many vulnerabilities, only accounting for 60% of the vulnerabilities Snyk tracks
- In 2018 alone, 500 vulnerabilities were disclosed by Snyk's proprietary dedicated research team

01

The open source landscape

Nobody would question that open source software has made an incredible impact on modern software development, and continues to expand every year. GitHub reported that 2018 had seen more new users signing up than during all of its first six years combined. This is accompanied with a 40% rise in new organizations and new repositories created on the platform, making 2018 the year during which almost one third of all repositories that exist on GitHub were created.

Open source software is everywhere too - contributions are made across all languages and platforms, impacting growth in different industries and, as per Forrester's report*, is an essential part of a business technology strategy.

* Miller, Paul. Nelson, Lauren E. "Open Source Powers Enterprise Digital Transformation." Forrester, 25 April 2016 (source)

Adoption

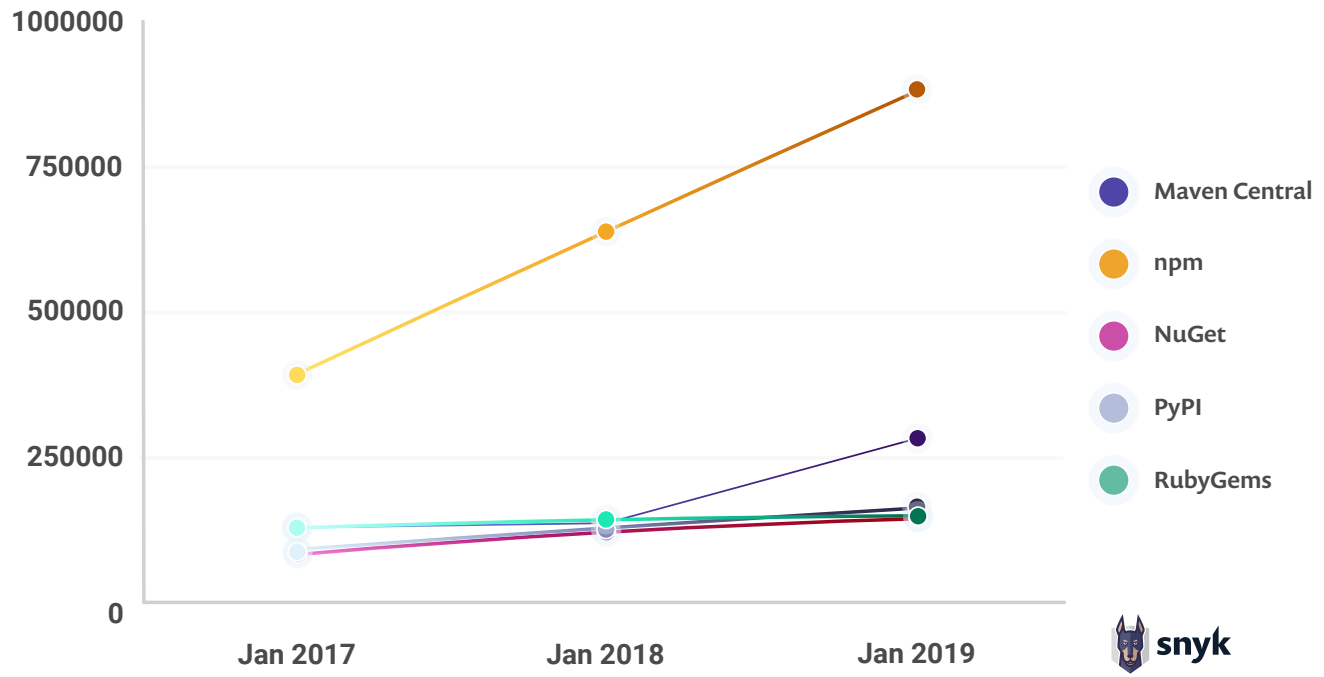
We've seen big technology players doubling-down on open source in 2018 as mentioned already earlier in this report. Let's look at the numbers. In every registry we reviewed, we saw an increasing rate of open source libraries being indexed in every language ecosystem. This is to be expected, but the rate of growth may come as a surprise to many.

All but one ecosystem presents two-digit numbers for increased growth in new libraries added to open source registries: Maven Central, with a strong growth of 102%, followed by PyPI with 40%, npm with 37%, NuGet with 26% and lastly RubyGems with 5.6% growth of newly indexed packages in the registries.

Use of open source is accelerating. In 2018, Java packages doubled, and npm added roughly 250,000 new packages

We may see further growth in numbers from 2018 due to undisclosed vulnerabilities that will only be publicized later this year, further amplifying the direction of this trend.

Total packages indexed per ecosystem



In 2018, The Linux Foundation reported that open source contributors have committed over 31 billion lines of code to date. However, with great adoption comes great responsibility and risk that need to be mitigated by anyone who owns, maintains or uses this code. In 2017 the CVE list reported more than 14,000 vulnerabilities, breaking the record for the most CVEs reported in a single year. 2018 continued the record-breaking streak with over 16,000 vulnerabilities reported.

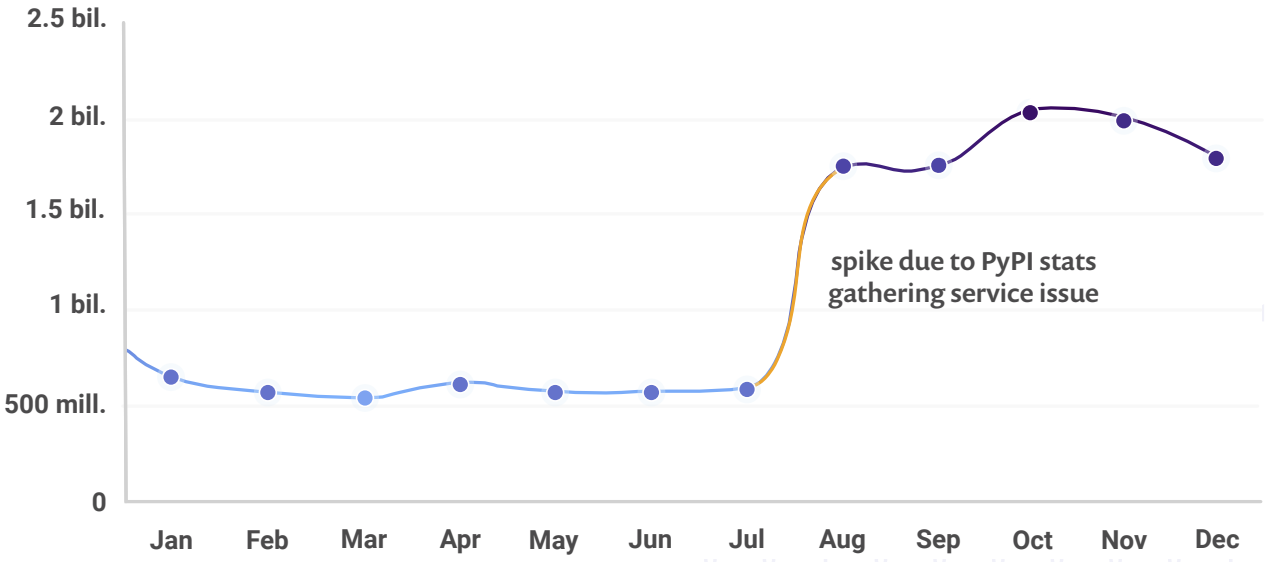
We can see how open source package growth translates into user adoption when looking at the download numbers for various packages in different ecosystems.

Examining the python registry, PyPI boasts more than 14 billion downloads during 2018, and doubles the download count in our 2017 report of approximately 6.3 billion downloads.

The spike in download count mid-year is due to a fault in linehaul, the statistics gathering service for PyPI, which missed recording about half of the downloads up until around August. The missing downloads presumably add up to more than the recorded 14 billion downloads of 2018.

Open source software consumption is also taking huge leaps forward. Twice as many Python packages were downloaded from PyPI, and a staggering 317 billion JavaScript packages from npm

Number of PyPI packages downloaded in 2018

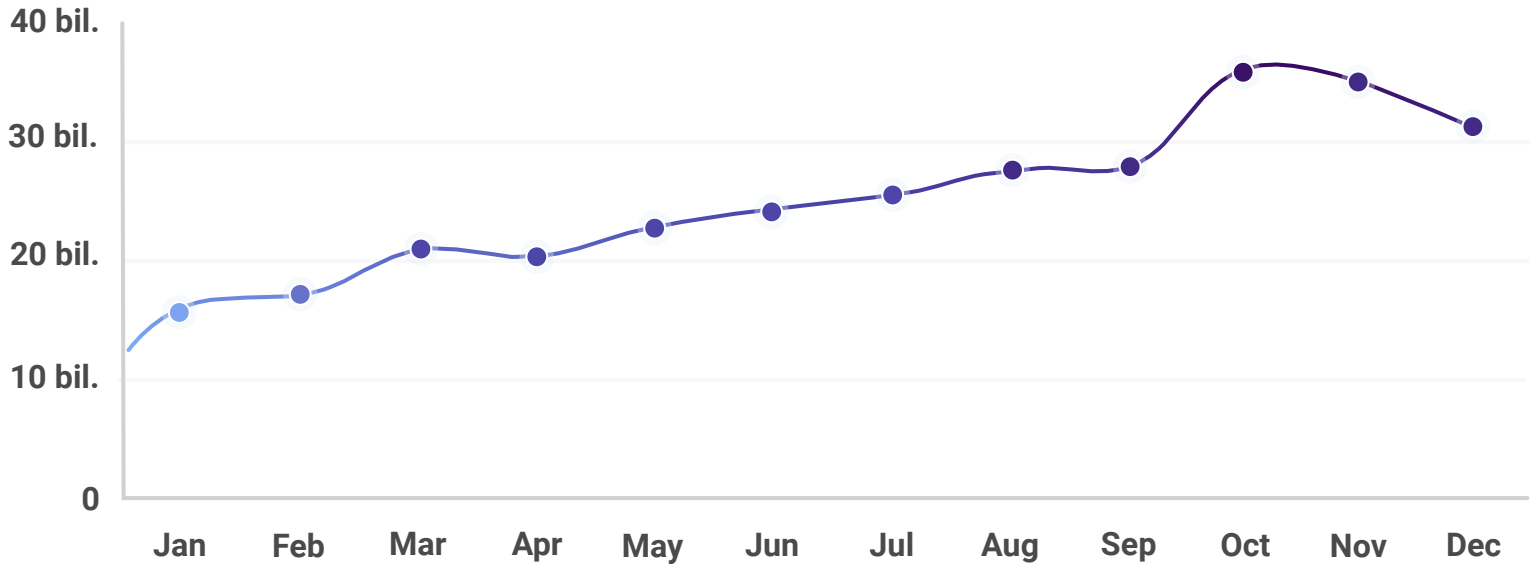


The npm registry is core to the entire JavaScript ecosystem. It has seen steady growth for both the number of packages being added and downloaded consistently over the years. It featured more than 30 billion downloads just for the single month of December 2018, and an incredible 317 billion downloads for the entire year of 2018.

The increased adoption of Docker containers further amplifies the strong growth of open source software. Docker Inc, the de-facto library and community for container images, reports more than 1 billion container downloads every 2 weeks over the last year, and about 50 billion to date, with more than 1 million new applications added into Docker Hub over the last year alone.

As package counts grow, so do their vulnerabilities. A record setting 16,000 new vulnerabilities were disclosed in 2018

Number of npm packages downloaded in 2018



Risks and impact

It shouldn't come as a surprise to most that in this year's State of the Octoverse report from GitHub, security is the most popular project integration app category with more than one integration for developers. Here's a quote from industry analyst Gartner in a recent application security report that covers the necessity for organisations to test for security as early as possible in the application lifecycle.

The more we use open source software, the more risk we accumulate as we're including someone else's code that could potentially contain vulnerabilities now or in the future. Moreover, risk doesn't solely reflect how secure the code is but also the licensing compliance of code you adopt and whether that code is in violation of the license itself.

Almost half (43%) of respondents have at least 20 direct dependencies, amplifying the need to monitor for open source vulnerabilities introduced through these libraries

Only one in three developers can address a high or critical-severity vulnerability in a day or less

> Enterprises should use SCA tools on a regular basis to audit repositories that contain software assets (such as version control and configuration management systems) to ensure that the software developed and/or used by the enterprise meets security and legal standards, rules and regulations. Application developers should have access to SCA tools to inspect the components they plan to use.
>
> — *Mark Horvath, Hype Cycle For Application Security 2018, Gartner*

Gartner®

Indirect dependencies

It is hard to imagine the days of writing software without any open source dependencies. Managing dependencies for a project is an important task, and requires due diligence to correctly keep track of the libraries you depend upon. After all, the application you are deploying bundles your code as well as your dependencies.

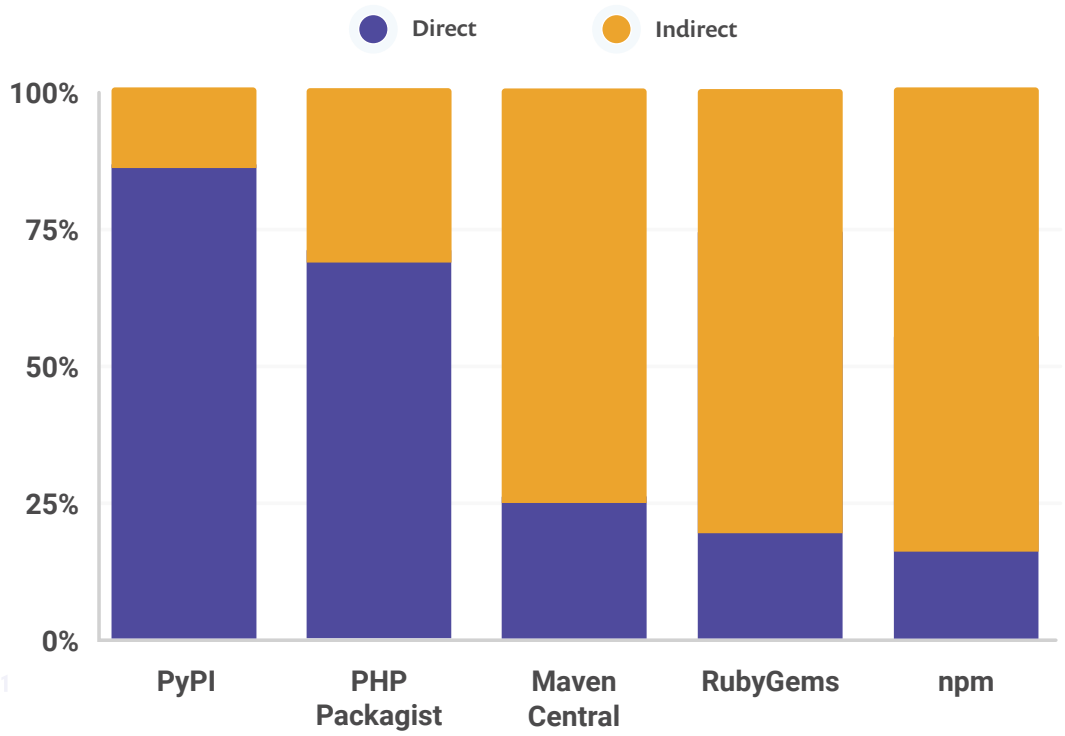
Most dependencies in npm, Maven and Ruby are indirect dependencies, requested by the few libraries explicitly defined. Vulnerabilities in indirect dependencies account for 78% of overall vulnerabilities

Snyk has scanned over a million snapshot projects and has discovered that vulnerabilities in **indirect** dependencies account for 78% of overall vulnerabilities. This further amplifies a critical need for clear insight into the dependency tree and the need to be able to correctly highlight nuances of a vulnerable path in order to address these vulnerabilities.

Of course, finding the vulnerabilities in a dependency is just the first step. Being able to precisely determine all the paths through the dependency tree in which the vulnerable dependency can be reached is a more complex issue.

Additionally, being able to suggest the steps to take that will eliminate the vulnerability while preserving the compatibility between dependencies is an even greater and much more interesting challenge.

The direct and indirect dependency split across ecosystems



Security posture of open source maintainers

Most developers and maintainers will likely agree that security should play an important role when building products and writing code. However there are no text-book rules for maintainers to follow for building open source projects, and as such their security standards can vary significantly.

Maintainers find themselves using their time and efforts on different aspects of the project, often functional, which in turn, could make security less of a priority for them in their process.

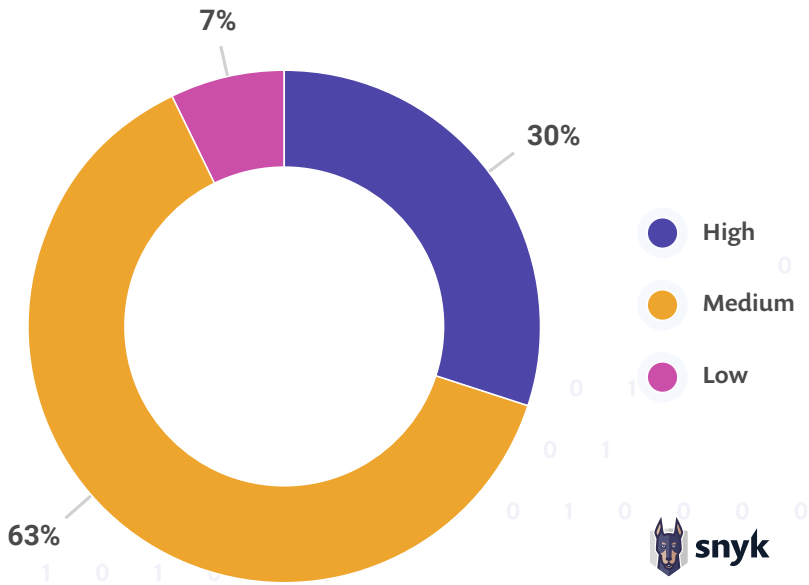
There's a positive trend towards security engagement and awareness since the time of our previous report, released in 2017.

Open source maintainers stated their security knowledge is improving but not high enough, averaging 6.6/10

This year, the majority of users ranked their security know-how as medium, with an average of 6.6 out of ten. A small portion of them (7%) ranked themselves as low, whereas the medium know-how ranking, representing the majority of users, has actually declined to 63% vs 56% last year.

The most movement is seen with the low and high rankings. Last year, security know-how was ranked as high by only 17%, while this year it has increased to almost 30%. In addition, we can see similar growth in low-ranked security know-how, which reached 26% last year but only 7% this year.

OS maintainers are confident in their own security knowledge



Security audits

A security audit could exist as part of a code review where peers ensure that secure code best practices are followed, or by running different variations of security audits such as static or dynamic application security testing. Whether manual or automatic audits, they are all a vital part of detecting and reducing vulnerabilities in your application, and should be executed as regularly and early in the development phase as possible in order to reduce risks of exposure and data breaches at a later stage.

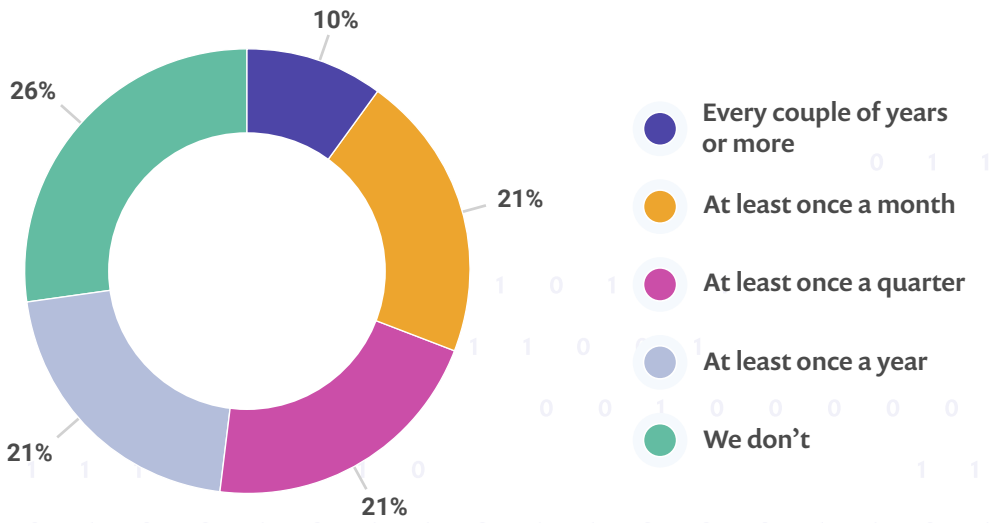
One in four open source maintainers do not audit their code bases

Last year 44% of respondents stated they had never run a security audit, while this year, the number is considerably lower with 26% of users stating they do not audit their source code. We're seeing positive trends toward repeated auditing actions this year across all audit cycles as compared to last year's report with an increase of an average 10% of users auditing their source code more often over the quarterly and yearly cycles.

Security professionals often cite the shift-left mantra in support of handling security concerns and potential problems earlier in the application lifecycle. This approach can uncover many valuable insights for developers through automation and help security keep up with the fast pace of modern, continuous development.

Shifting left, especially in security, is key and at times even critical, to reducing the cost of security incidents that are only found in production. One way to address security earlier in the process and to increase the chances of developers adopting those practices is to select tools that are developer friendly and built to integrate with their existing workflows.

OS Maintainers differ in their code auditing cadence



02

Known open source vulnerabilities

A vulnerability is a vulnerability, whether known or not. The key difference between the two is the likelihood of an attacker to be aware of this vulnerability, and try to exploit it. Therefore, the better known the vulnerability is, the more urgent it is to deal with it.

A known vulnerability might have a CVE ID associated with it as part of a responsible disclosure, or it might just be disclosed on the internet or stored in open databases. These are all types of known vulnerabilities that you should prioritize eliminating as they have a higher chance of being attacked in production. After these, vulnerabilities that are captured in closed vulnerability databases or even shared in the dark web should be considered.

Known vulnerabilities in application libraries

Today, we're witnessing an increase in the number of vulnerabilities reported across many of the ecosystems that we track, including PHP Packagist, Maven Central Repository, Golang, npm, NuGet, RubyGems, and PyPI.

In 2017, we saw a 43% increase of vulnerabilities reported across all registries, and in 2018 the vulnerability count grew by a further 33%.

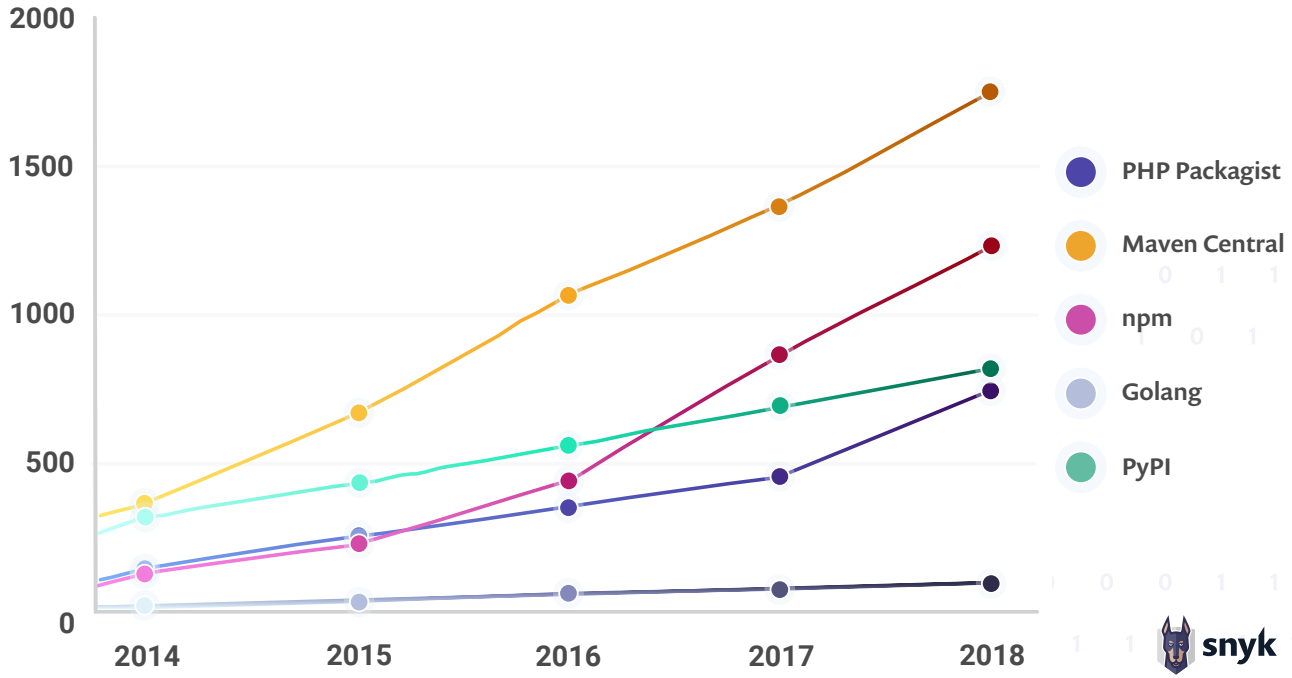
When examining the five different ecosystems: PHP, Java, JavaScript, Python and Go, we see an increasing trend in the number of vulnerabilities disclosed across all of them since 2014.

Vulnerabilities are found at an increasing pace, nearly doubling in the last 2 years

We may see further growth in numbers from 2018 due to undisclosed vulnerabilities that will only be publicized later this year, further amplifying the direction of this trend.

In 2018, new disclosures for npm grew by 47%, and Maven Central grew by 27%

New vulnerabilities each year by ecosystem



In 2018 vulnerabilities disclosed for PHP Packagist grew by a staggering 56%, and for Maven Central, disclosures increased by 27%. Although Golang is a smaller ecosystem, it has growing security research and reported 52% new vulnerabilities in 2018 over 2017.

Looking back at the data from 2014 in Snyk's vulnerability database, we see a strong overall increase in the number of vulnerabilities across the board.

Today, we track 1766 vulnerabilities in the Maven Central Repository, 1268 in npm, 746 in PHP Packagist, 807 in PyPI, and 94 in Golang.

Since 2014, the number of vulnerabilities in the Snyk database has increased by an astonishing 371%, with npm vulnerabilities increasing by an incredible 954% and Maven Central vulnerabilities increasing by 346%.

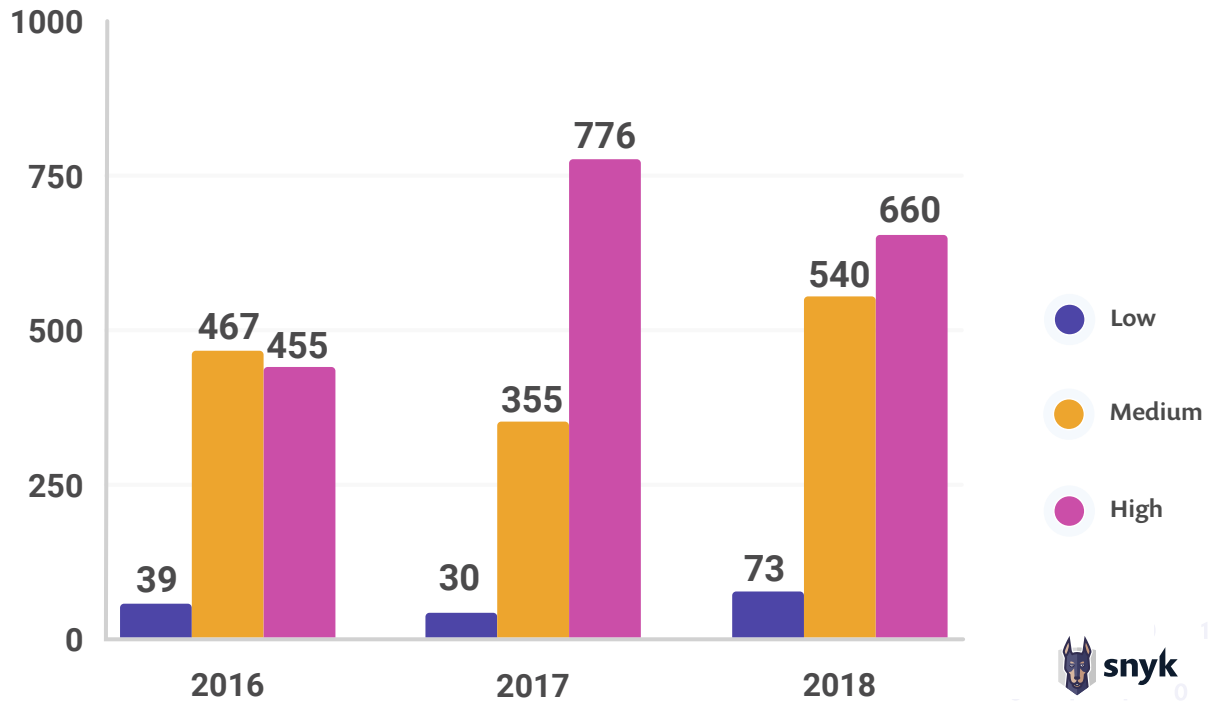
Since 2014, the number of vulnerabilities in Snyk's database for npm grew by 954% and for Maven Central by 346%

Trends in severity

When we look at vulnerability severity for application libraries disclosed over the last three years across all language ecosystems, 2018 shows a smaller number of high vulnerabilities as compared to the previous year.

However an interesting insight for both 2017 and 2018 is that there were more high severity vulnerabilities than medium or low vulnerabilities as compared to 2016.

Vulnerability severities by year



Spotlight: Zip Slip

In 2018, the Snyk Security research team responsibly disclosed many instances of a vulnerability dubbed Zip Slip, a widespread arbitrary file overwrite critical vulnerability. It can be exploited using a specially crafted archive that holds directory traversal filenames and typically results in remote command execution.

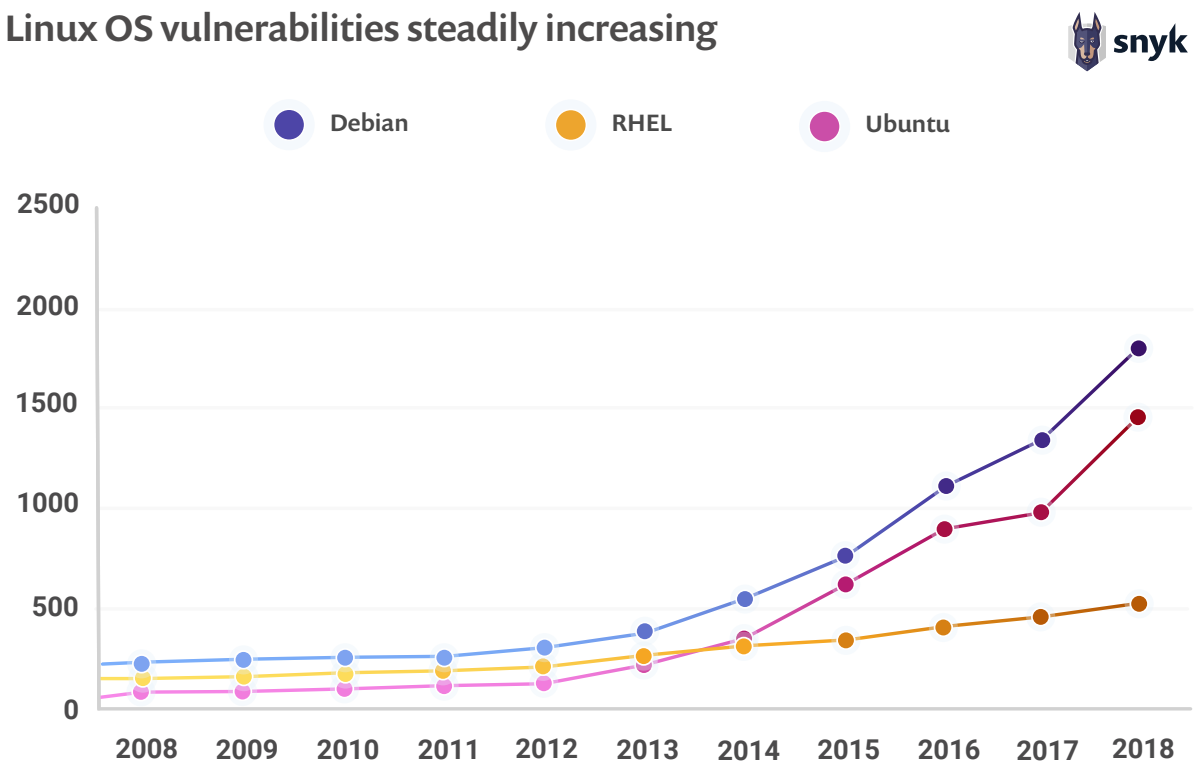
It was discovered and responsibly disclosed by the Snyk Security team ahead of a public disclosure on 5th June 2018, and affects thousands of projects, including ones from HP, Amazon, Apache, Pivotal, and many others.

The research that spanned various ecosystems uncovered tens of vulnerabilities in libraries such as Apache Ant, adm-zip, SharpCompress and others used by thousands of projects for Java, npm, NuGet, Go, .NET, Ruby, Python and C++. Almost half of them were found to be of high severity.

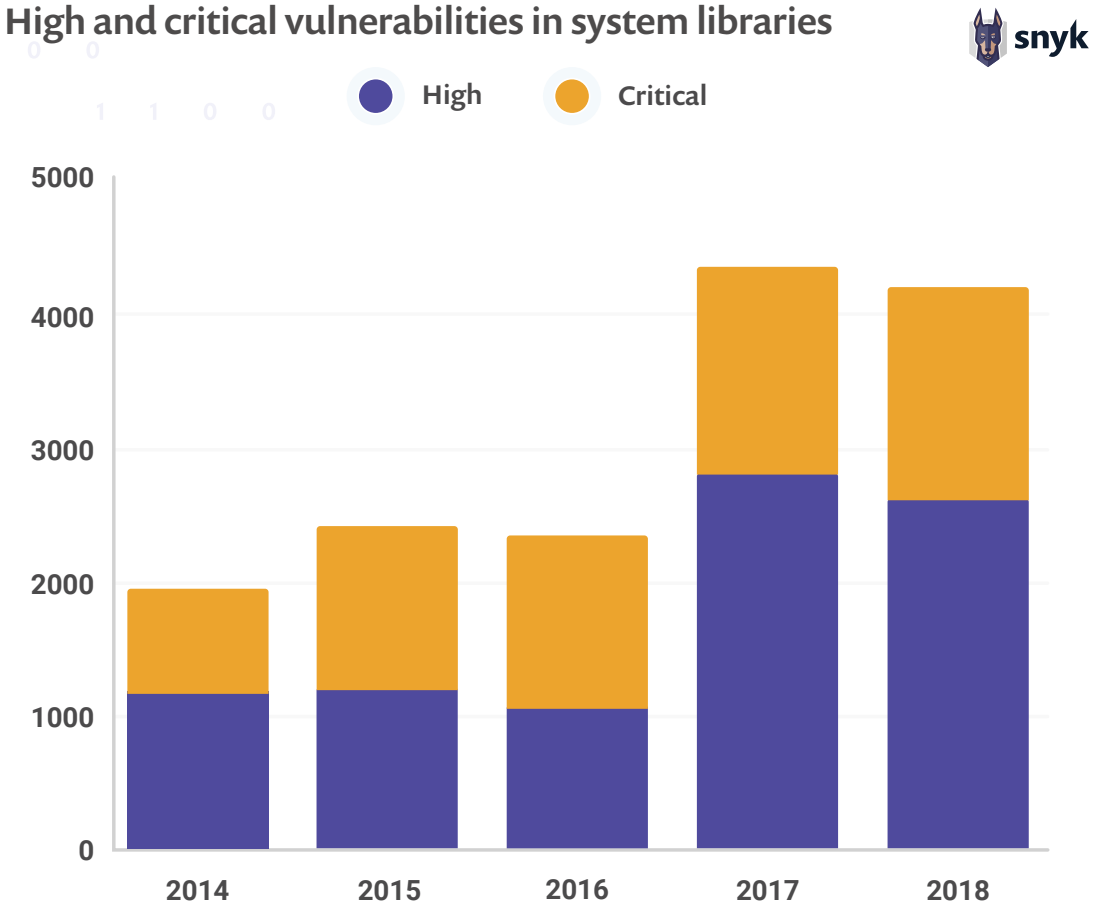
> When we discovered the first instance of the Zip Slip vulnerability in a big project, it was very exciting. It was our eureka moment, but when we discovered that every other application had a vulnerable implementation, we were extremely surprised. We realised that this vulnerability wasn't just affecting a few apps, but loads of projects across ecosystems.
>
> — *Danny Grander, Snyk CSO*

Known vulnerabilities in system libraries

There is an increase in the number of vulnerabilities reported for system libraries, affecting some of the popular Linux distributions such as Debian, RedHat Enterprise Linux and Ubuntu. IIn 2018 alone we tracked 1,597 vulnerabilities in system libraries with known CVEs assigned for these distros, which is more than four times the number of vulnerabilities compared to 2017.



As we look at the breakdown of vulnerabilities (high and critical) it is clear that this severity level is continuing to increase through 2017 and 2018.



Known vulnerabilities in docker images

The adoption of application container technology is increasing at a remarkable rate and is expected to grow by a further 40% in 2020, according to 451 Research. It is common for system libraries to be available in many docker images, as these rely on a parent image that is commonly using a Linux distribution as a base.

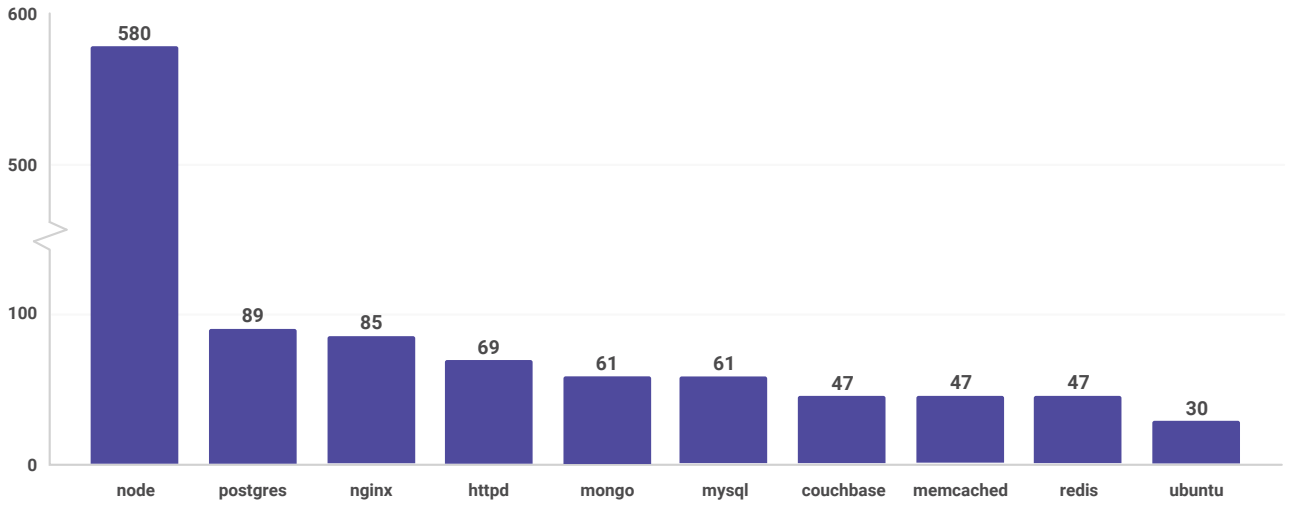
Docker Hub provides insights into the most popular docker images.

Accordingly, we've scanned through ten of the most popular images with Snyk's recently released docker scanning capabilities.

The findings show that in every docker image we scanned, we found vulnerable versions of system libraries. The official Node.js image ships 580 vulnerable system libraries, followed by the others each of which ship at least 30 publicly known vulnerabilities.

Docker images almost always bring known vulnerabilities alongside their great value

Number of OS vulnerabilities by docker image



Snyk recently released its container vulnerability management solution to empower developers to fully own the security of their dockerized applications. Using this new capability, developers can find known vulnerabilities in their docker base images and fix them using Snyk's remediation advice. Snyk suggests either a minimal upgrade, or alternative base images that contain fewer or even no vulnerabilities.

Based on scans performed by Snyk users, we found that 44% of docker image scans had known vulnerabilities, and for which there were newer and more secure base image available. This remediation advise is unique to Snyk. Developers can take action to upgrade their docker images.

Snyk also reported that 20% of docker image scans had known vulnerabilities that simply required a rebuild of the image to reduce the number of vulnerabilities.

Fix can be easy if you're aware. 20% of images can fix vulnerabilities simply by rebuilding a docker image, 44% by swapping base image

Vulnerability differentiation based on image tag

The current Long Term Support (LTS) version of the Node.js runtime is version 10. The image tagged with 10 (i.e: node:10) is essentially an alias to node:10.14.2-jessie (at the time that we tested it) where jessie specifies an obsolete version of Debian that is no longer actively maintained.

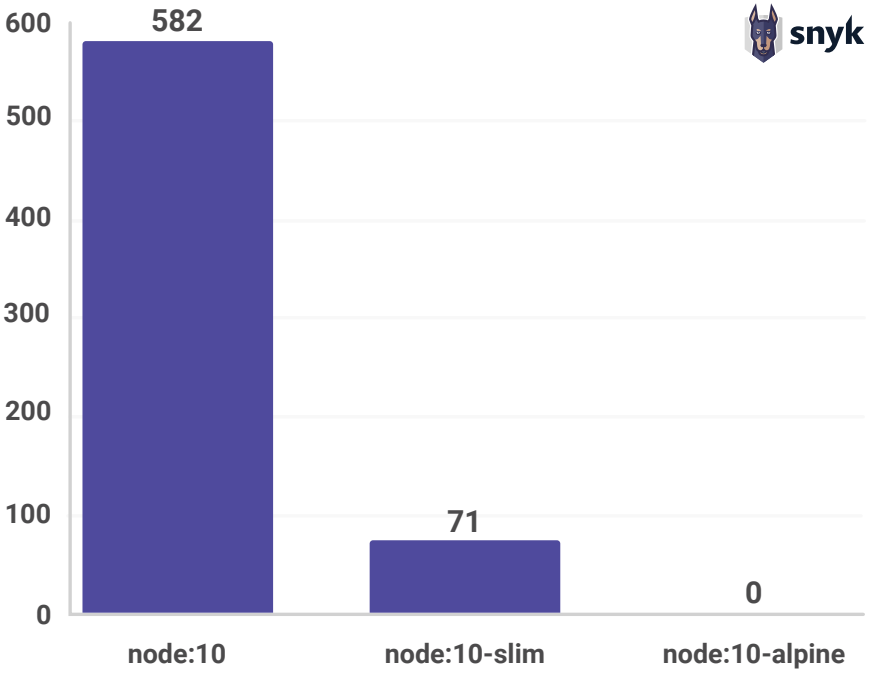
If you had chosen that image as a base image in your Dockerfile, you'd be exposing yourself to 582 vulnerable system libraries bundled with the image. Another option is to use the node:10-slim image tag which provides slimmer images without unnecessary dependencies (for example: it omits the main pages and other assets). Choosing node:10-slim however would still pull in 71 vulnerable system libraries.

The node:10-alpine image is a better option to choose if you want a very small base image with a minimal set of system libraries. However, while no vulnerabilities were detected in the version of the Alpine image we tested, that's not to say that it is necessarily free of security issues.

Alpine Linux handles vulnerabilities differently than the other major distros, who prefer to backport sets of patches. At Alpine, they prefer rapid release cycles for their images, with each image release providing a system library upgrade.

Most vulnerabilities originate in the base image you selected. For that reason, remediation should focus on base image fixes

Number of vulnerabilities by node image tag



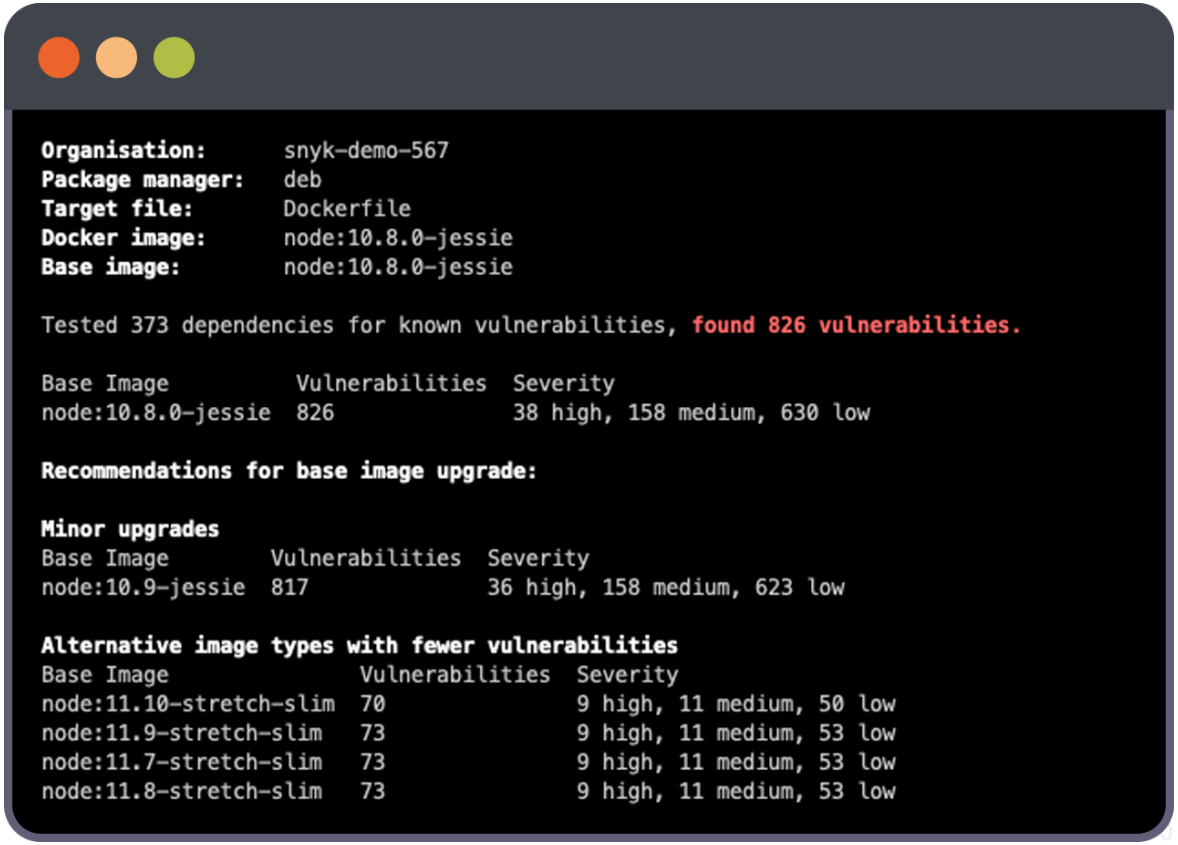
Moreover, Alpine Linux doesn't maintain a security advisory program, which means that if a system library has vulnerabilities, Alpine Linux will not issue an official advisory about it; Alpine Linux will mitigate the vulnerability by creating a new base image version including a new version of that library that fixes the issue, if one is available (as opposed to backporting as mentioned).

There is no guarantee that the newer fixed version, of a vulnerable library will be immediately available on Alpine Linux, although that is the case many times. Despite this, if you can safely move to the Alpine Linux version without breaking your application, you can reduce the attack surface of your environment because you will be using fewer libraries.

The use of an image tag, like node:10, is in reality an alias to another image, which constantly rotates with new minor and patched versions of 10 as they are released.

A practice that some teams follow is to use a specific version tag instead of an alias so that their base image would be node:10.8.0-jessie for example. However, as newer releases of Node 10 are released, there is a good chance those newer images will include fewer system library vulnerabilities.

Using the Snyk Docker scanning features we found that when a project uses a specific version tag such as node:10.8.0-jessie, we could then recommend newer images that contain fewer vulnerabilities.



Spotlight: Malicious packages

You may have heard about malicious packages in a variety of contexts, such as a malicious docker container or perhaps a malicious package in a public registry of one ecosystem or another. We have also discussed developers as a malware distribution vehicle in several other contexts such as the Induc malware that infected Delphi compilers and XCodeGhost that targeted iOS and OSx developers.

However, not all malicious packages are the same in nature. With regards to ecosystem registries we can broadly classify them into the following:

- a typosquatting attack where a malicious package uses a very similar name of a more popular package
- a compromised maintainer's CI or registry account resulting in the publishing of a malicious version, or a malicious package residing in a project's list of dependencies
- a socially engineered inclusion of a malicious package (or a package that will be malicious after inclusion) into a project list of dependencies

In 2018 we saw occurrences of all of these malicious package types in the npm ecosystem, known for being one of the registries that suffers from malicious packages more than others. The package that recently made the news in December 2018 was event-stream. It relied on a malicious dependency that was delivered through a seemingly innocent attempt to make an open source contribution.

This hack affected a staggering 8 million downloads of the malicious package in only two months. Another example in 2018 is the ESLint-scope package in which the maintainer's account was compromised. We also saw a total of 11 typosquatting attacks for malicious packages published in 2018 on the npm registry.

In 2018, a malicious package was downloaded a record 8 million times. It was one of 25 typosquatting attacks in npm and PyPI

In addition to the typical typosquatting attempts that we've seen in the past, we also saw more mature malicious attempts to attack the npm ecosystem than in previous years, such as the ESLint-scope attack. With much higher sophistication, the event-stream incident exposed a high level of expertise and targeted attacks than we've seen in previous malicious attempts in the ecosystem to date.

In contrast to the npm registry, the only other registries in which we identified malicious packages were RubyGems with just one malicious package in 2018, and Python with ten malicious packages in 2017 and thirteen in 2018.

03

Vulnerability characteristics of each ecosystem

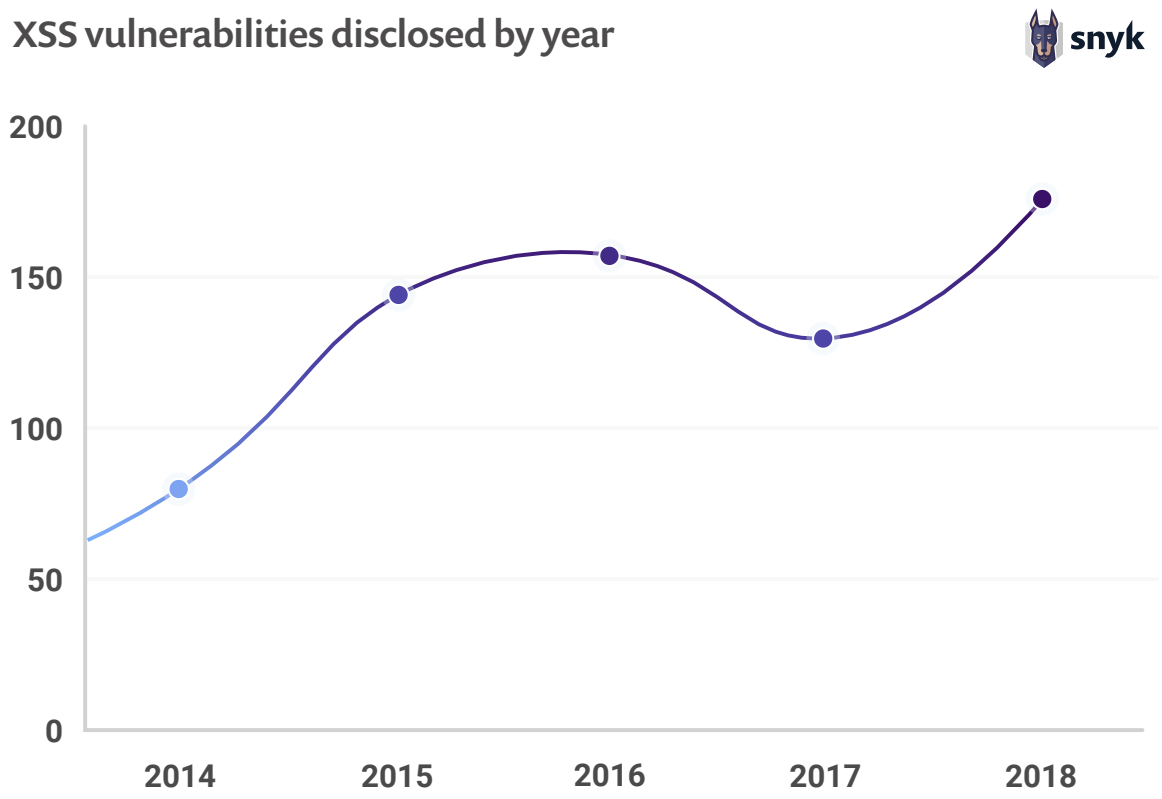
We were curious to learn more about the distinct vulnerability families found within each ecosystem in order to better understand what attackers target for exploitation.

XSS vulnerabilities

Cross-site Scripting (XSS) attacks have been an ever-increasing pain point for web applications and we see the trend in XSS vulnerabilities spiking in 2018 across all ecosystems that Snyk has been monitoring.

XSS vulnerabilities in open source libraries are still on the rise, despite being a top concern by OWASP for more than 15 years

Within these ecosystems, we've detected that the npm ecosystem has seen the most XSS vulnerabilities, disclosing 225 in total; followed by Maven Central Repository with 167; and PyPI with 163 total cross-site scripting vulnerabilities. In 2018, the PHP Packagist ecosystem disclosed the most with 56 XSS vulnerabilities, followed by npm with 54, and Maven Central with 29.

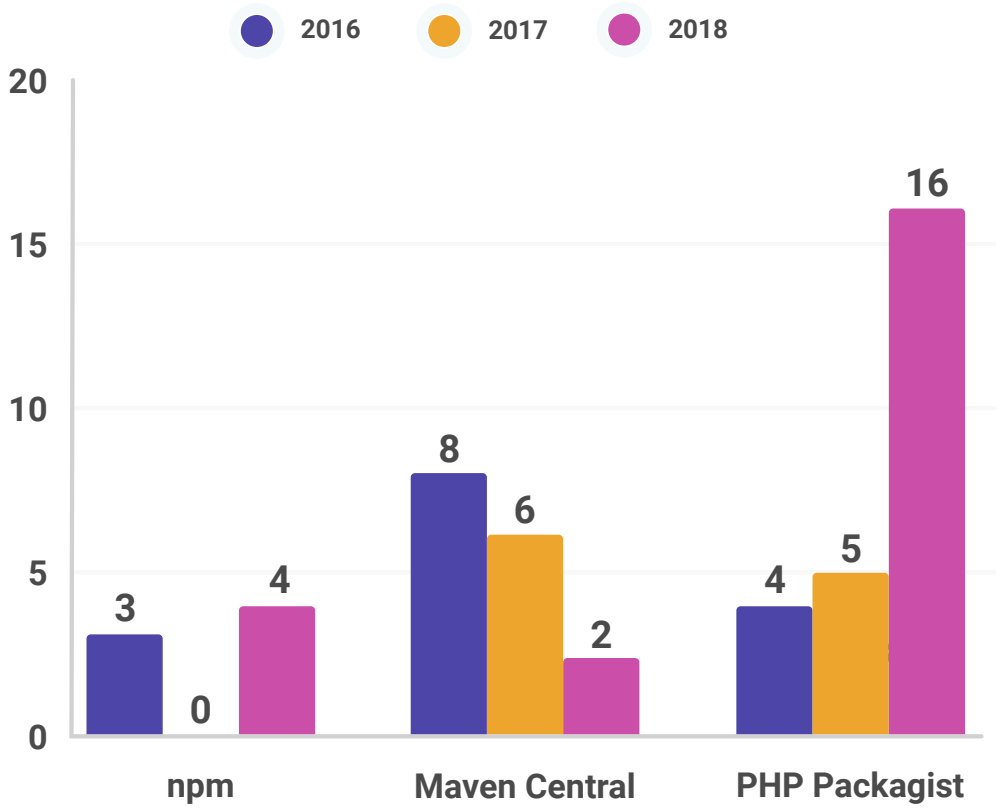


SQL injection vulnerabilities

Another common attack vector that is consistently featured in the OWASP's top 10 over the past decade is CWE-89, more commonly known as SQL Injection.

Looking across the last three years, we can see that each of the three main ecosystems we reviewed have peaks during different years. Maven libraries lead the number of SQL injection vulnerabilities disclosed in both 2016 and 2017, followed by PHP Packagist libraries, which hit a peak in 2018.

SQL injection disclosures show spikes by year and ecosystem



Sensitive information exposure

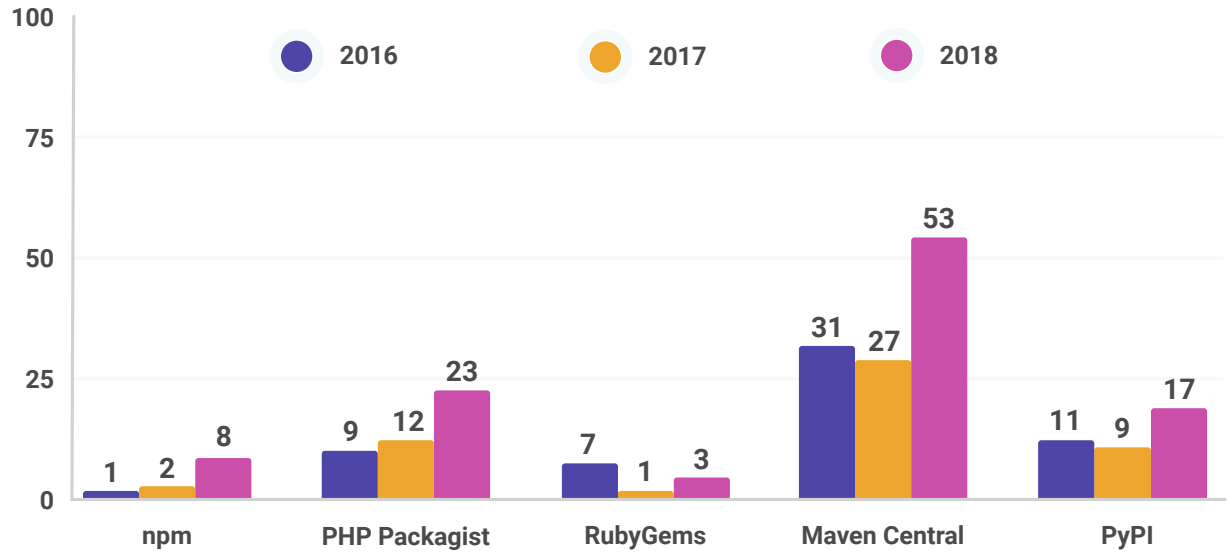
Looking at the Maven Central and PHP Packagist registries we found they had the most vulnerabilities related to information exposure, peaking in 2018 for both ecosystems.

Information exposures often happen unintentionally. They occur when a program or system discloses potentially sensitive information, such as environment variable names and values. Cases of information exposure may also occur "by design" such as when sensitive data is provided within URL parameters.

Several examples of information exposure vulnerabilities in the Maven Central registry are apache spark, jenkins core, keyclock-saml-core packages. Jenkins ssh-agent CI plugin, for example, leaked the SSH private key in the build logs for anyone with Read permissions to see.

The PyPI registry also has a good amount of vulnerabilities found in libraries, with examples of information exposure vulnerabilities. Packages such as django displayed a user password hash to admin users who only had View permissions. The package djangorestframework-api-key saved API keys in plain text.

Sensitive information exposure vulnerabilities affecting the Java ecosystem

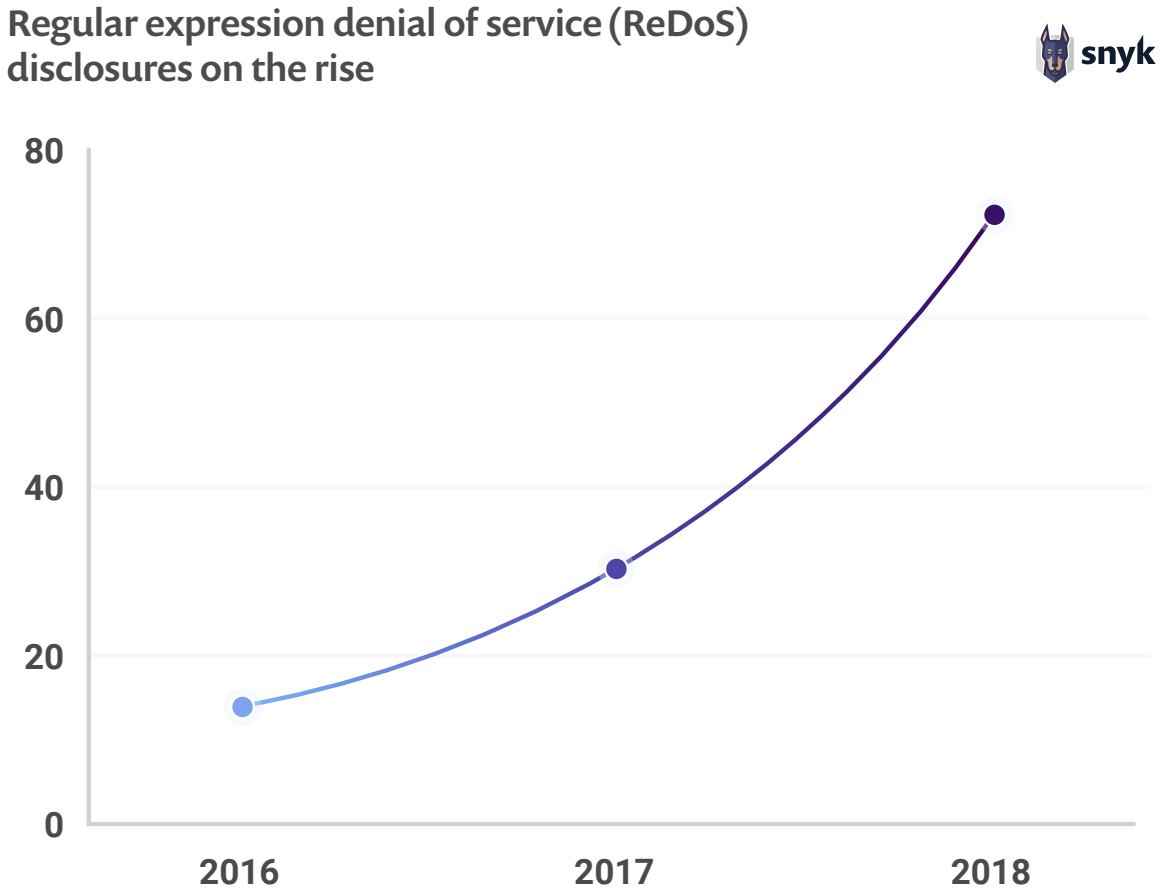


Regular expression denial of service

The Node.js runtime is known to have many strengths, but one of them, the single threaded Event Loop, can also be its weakest link if not used correctly. This happens more regularly than one might think.

Regular expression denial of service (ReDoS) attacks exploit the non-linear worst-case complexity vulnerabilities that some regex patterns can lead to. For a single-threaded runtime this could be devastating, and this is why Node.js is significantly affected by this type of vulnerability.

We found that there were a growing number of ReDoS vulnerabilities disclosed over the last three years, with a spike of 143% in 2018 alone.

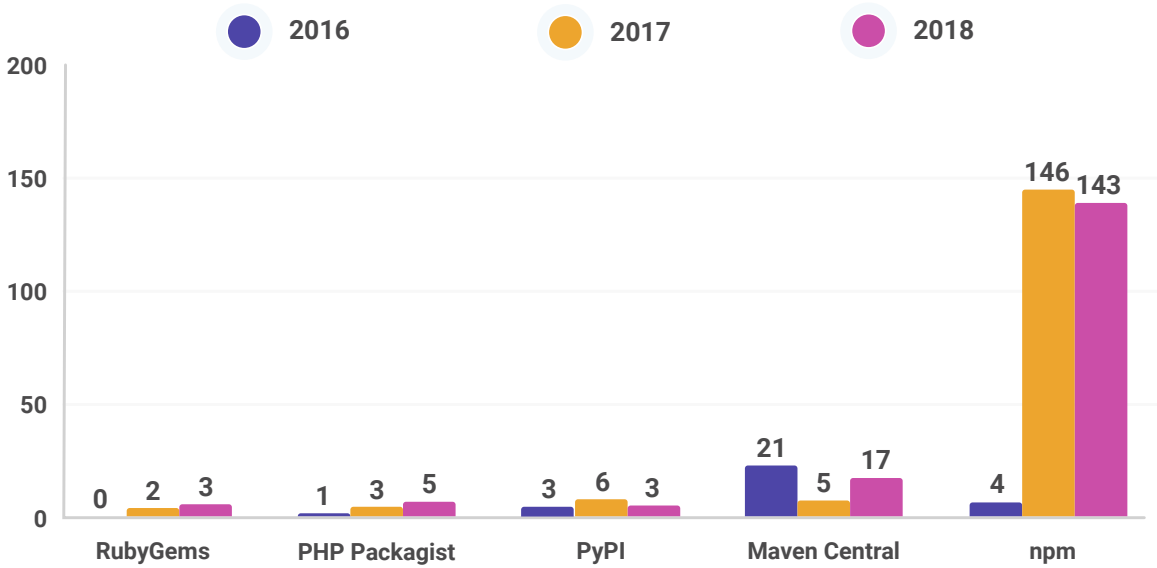


Path traversal

Path and directory traversal vulnerabilities fiercely stand out in the npm ecosystem with record numbers of 146 and 143 disclosures in 2017 and 2018, respectively. The other ecosystems are much further behind, which is a good thing!

One might presume that this may be attributed to the plethora of static and dynamic web servers built with Node.js for both production and development use, and therefore there are many more packages in which such vulnerabilities might also be found.

Path traversal vulnerabilities most commonly seen in npm



Cleartext transmission of sensitive information

Last but not least is another unique vulnerability worthy of mention in the npm ecosystem, CWE-319, also known as Cleartext Transmission of Sensitive Information, in which resources are accessed over insecure protocols. We were able to find 44 new reported vulnerabilities in packages from 2016, and this number further rises to a hefty total of 110 packages in 2017, a 250% increase.

> The state of an ecosystem's security and its public perception are often extremely different. The lack of typing in JavaScript has spread the idea that it is an unsafe language due to type manipulation, but in any case, the number of vulnerabilities discovered in npm modules over the last couple of years is still lower than those discovered on Maven central. At the same time, some vulnerabilities may be exacerbated because Node.js is still mono-threaded. ReDoS (or other CPU-exhaustion DoS), which is much more common in the Node.js world, is an example of this. Hopefully, Worker Threads will soon enable Node.js, in order to reduce these risks. The security community in Node.js has been more and more active in the past years and we can continue to work hard so that the ecosystem becomes safer in the future.
>
> *— Vladimir de Turckheim, Node.js Foundation Security WG*

04

The open source security lifecycle

A healthy approach to embracing security as part of the SDLC is to integrate it within the entire development lifecycle, from design to production. This significantly differs from the more traditional one-off phase of security testing that occurs periodically and doesn't fit the modern, fast-paced software delivery model. However, processes and guidelines may not be enough. Education, friendly tooling, and engagement with R&D teams and stakeholders are just as important to the healthy adoption of security practices within an organization.

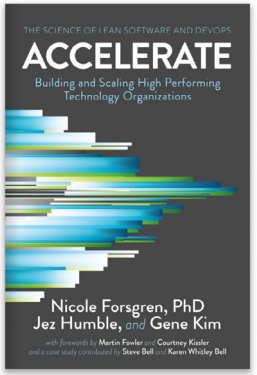
Discovering vulnerabilities

It takes a great deal of knowledge, experience, and a sharp eye to properly code review for potential security vulnerabilities within one's own code. As this isn't a straightforward task, if carried out at all, it suggests that vulnerable code may stay dormant for a long time until it is picked up by anyone.

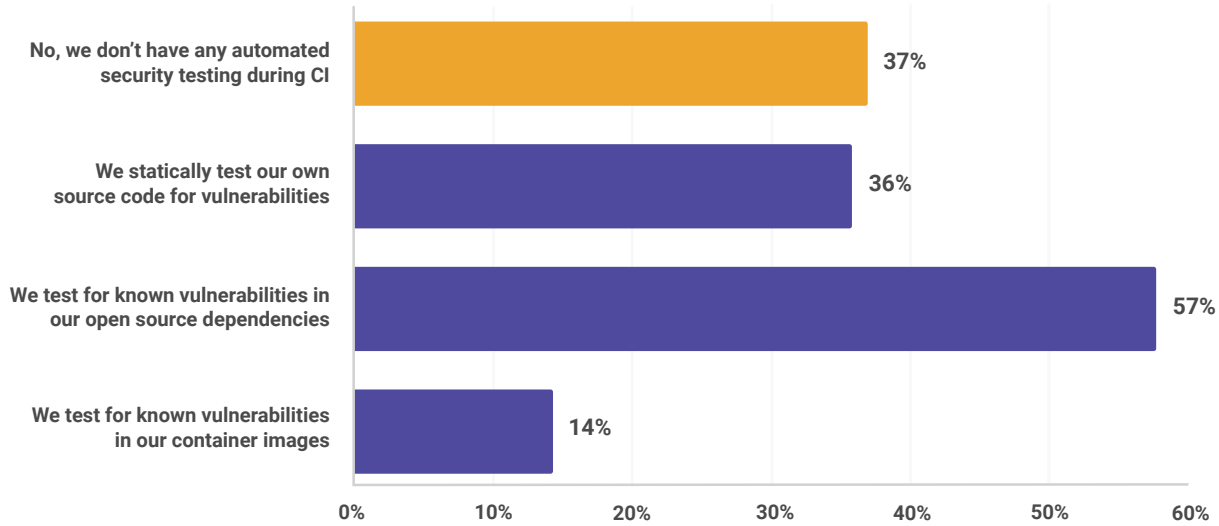
Teams that practice DevOps or have a mature CI/CD pipeline may find it easier to introduce security testing as part of their build automation, yet we find that almost 40% of users don't implement any sort of security testing during their CI runs. A reassuring note however is that more than half of them are at the very least testing for vulnerabilities in their open source dependencies.

37% of users of users don't implement any sort of security testing during CI

> Another finding in our research is that teams that build security into their work also do better at continuous delivery. A key element of this is ensuring that information security teams make pre-approved, easy-to-consume libraries, packages, toolchains, and processes available for developers and IT operations to use in their work.
>
> — *Nicole Forsgren, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*



Security testing during CI

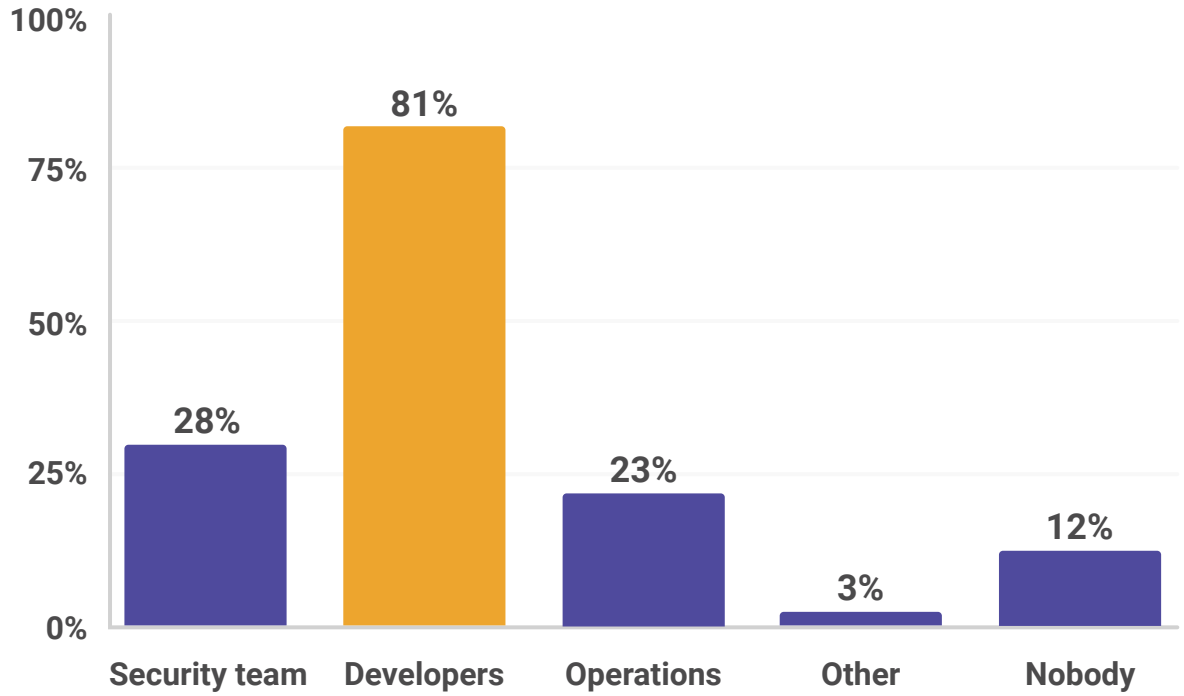


Open source security ownership

When facing such alarming statistics, we set out to find who in practice owns the security responsibility of an application or library today, as well as who users think **should** take ownership of security.

According to 81% of respondents, developers should own the security of their application code, sending a strong statement about the involvement and engagement level that is expected from developers, and supports the strong DevSecOps movement which many are adopting today.

Who is responsible for security?

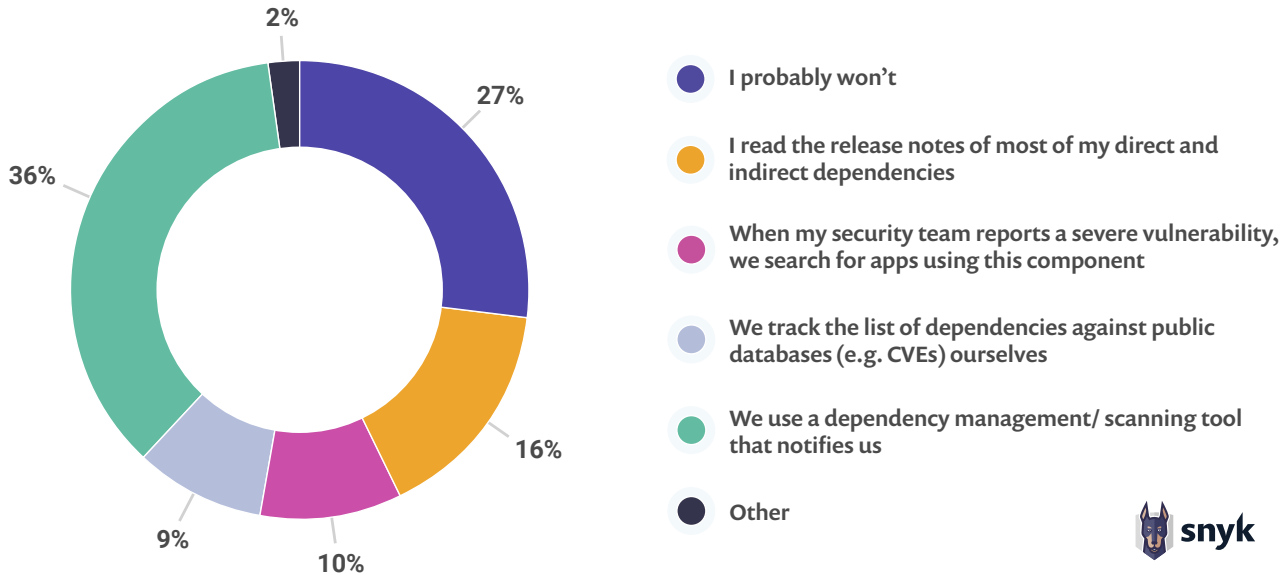


Finding out about vulnerabilities

From the user's perspective, it is interesting to gain insights into how they learn about vulnerabilities in their application dependencies in order to respond to potential threats as they are discovered.

A worrying 27% of respondents stated they do not have any proactive or automatic way to find out about newly discovered vulnerabilities in their applications. Only 36% of users confirmed that they use a dependency management or scanning tool to help surface vulnerabilities.

How do you find about vulnerabilities?



Snyk stats

- In the second half of 2018 alone, Snyk opened more than 70,000 Pull Requests for its users across Maven, RubyGems and npm ecosystems to remediate vulnerabilities in their projects.
- Out of all the dependencies in a scanned Java project, Snyk provided a remediation path to fix vulnerabilities that were found in 60% of them. It's not always possible to fix remediation paths when there is no compatibility between a direct dependency and a fixed version of an indirect dependency. The Snyk Security team can provide custom patches to fix some of these situations.

Spotlight: Vulnerabilities without CVEs

It is common for security teams to keep track of, and to react to, new vulnerabilities as they are disclosed through the National Vulnerabilities Database (NVD), or other public CVE repositories.

However, a good number of security vulnerabilities are discovered and fixed in non-official channels such as through informal communication between maintainers and their users in an issue tracker.

The Snyk database is carefully curated by an internal security analysts team, and tracks vulnerabilities not included in these official sources but mentioned in public locations such as forums or release notes. Using Snyk's DB as a barometer, we see it uncovers 67% more vulnerabilities than public databases.

In addition to comprehensiveness, CVEs and public databases are often slow to add vulnerabilities. If we look at npm as an example, vulnerabilities only show up in npm audit an average of 92 days after they are captured in Snyk's DB, and lag behind 72% of the time.

These gaps indicate the CVE system and public open source databases are not currently coping with the pace and volume of open source software vulnerabilities. These mechanisms should be reevaluated, and security conscious organisations should seek out commercial databases for timely and broad coverage.

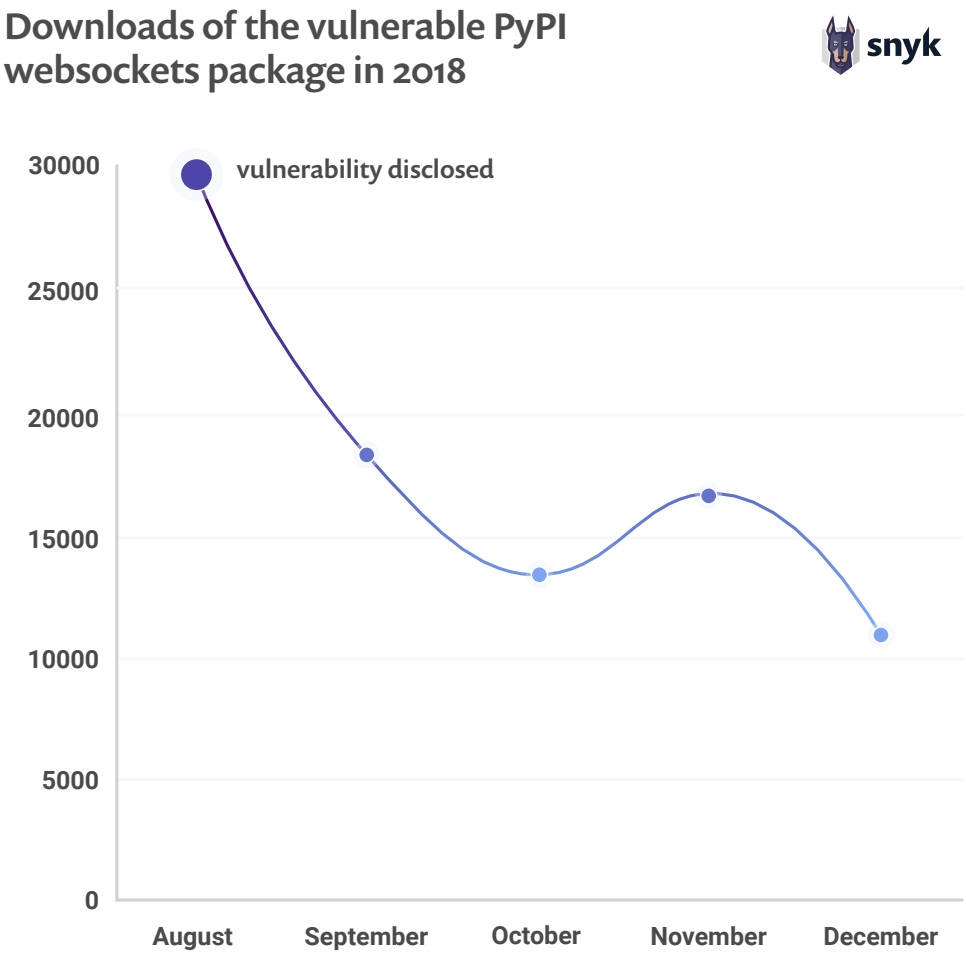
Time to adopt security fixes

How long does it take users to adopt new releases that provide security fixes to known vulnerabilities? We turned to Python's PyPI registry and its websockets package for an example to see how popular vulnerable releases continued to be used even after a vulnerability fix was released.

The websockets project is a fairly popular and well-maintained package, dating back to 2013 and showcasing regular releases to the present day.

In August 2018 a denial of service vulnerability was disclosed to the community, affecting versions 4.0 and 4.0.1 of the package. At the time of disclosure, newer versions already existed on the registry that provided the security fix, however looking at the download counts for the vulnerable versions, a long trail of users still fetch vulnerable versions of websockets can be seen.

By December 2018 we're still tracking 11k downloads of the websockets package that contain the vulnerability, even though there is a fixed version available as a major upgrade with websockets version 5.0.



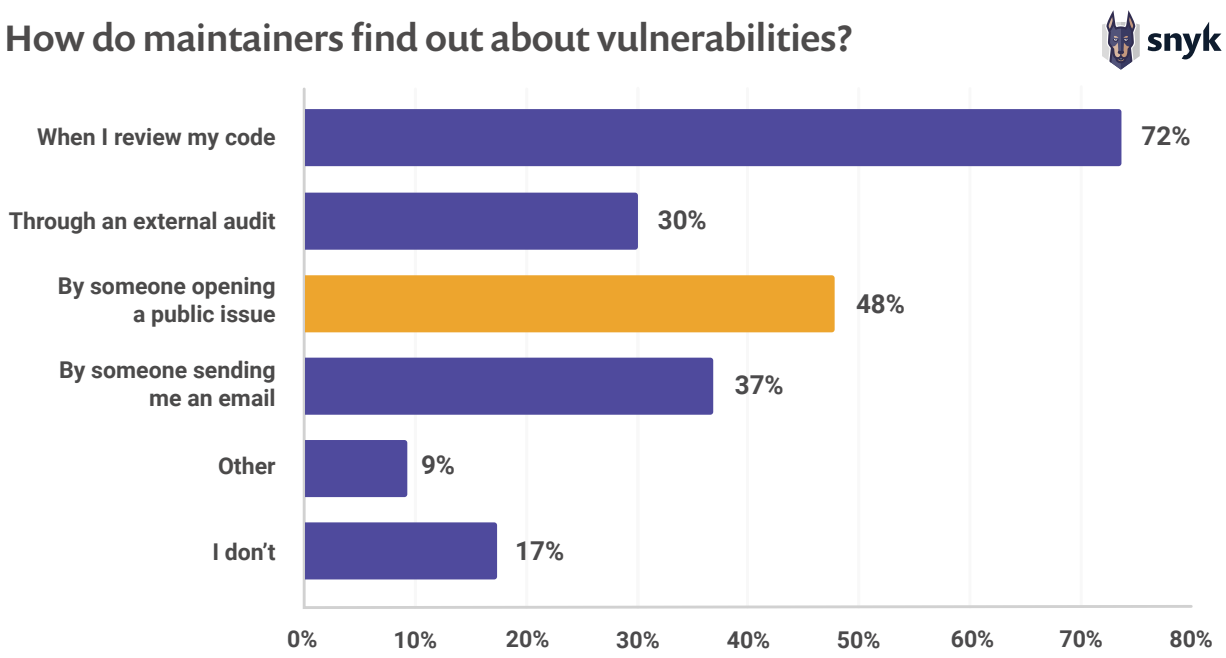
How do maintainers find out about vulnerabilities?

It is more likely that maintainers be alerted to a security concern than it is that they find out themselves. An industry-accepted best practice is a responsible disclosure policy, which details how security researchers and individuals should safely report security vulnerabilities to project maintainers.

From the survey data, we can conclude that almost half (48%) of respondents find out about a security vulnerability that is in their code from a public channel, such as when someone else is opening a public issue or contacting them over email.

72% of users said they find out about vulnerabilities in their code when they review their own code personally; however 62% of users have stated they have only medium-level security know-how whereas only 30% of them state their security expertise is high.

Furthermore, while the majority of users (72%) say they review their own code to find vulnerabilities, 48% of users still learn about vulnerabilities in their code only when someone else opens a public issue, demonstrating how hard it is to rely on just one maintainer reviewing code even if that maintainer is perceived to have good security knowledge.

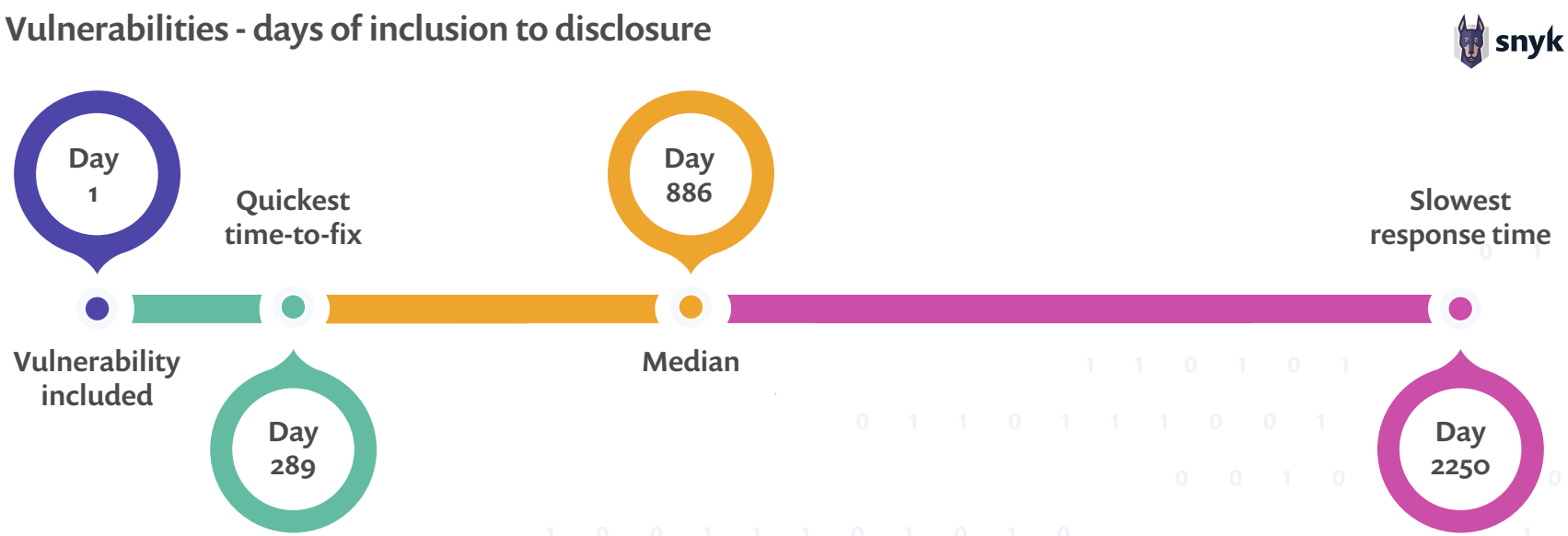


Inclusion to disclosure

One of the research questions we wanted to answer was how long it takes from the time a vulnerability enters the code base and until it is discovered and disclosed? To answer this, we set out to analyze several top libraries in the npm ecosystem and the vulnerabilities that were discovered in them during 2018.

As this is more time-consuming and tricky to accurately automate, we looked at the top six npm libraries and analysed their code bases to see the differences between the dates of the commits that introduced the vulnerability and fixed the vulnerability. Of course, these calculations are slightly biased because we're using such a small sample size, but the range and order of numbers are interesting all the same!

Of these six libraries, we saw that the quickest time-to-fix from inclusion was almost one year, or 289 days to be precise. The median time is almost 2.5 years, and the worst case we saw was 5.9 years.



Spotlight: Equifax, a year later

A recent report released by the US government deemed the infamous Equifax breach as completely preventable, and demonstrated how important it is to shift security to the left by integrating it into the development workflow.

With a DevSecOps mindset and good practices employed, a development team could have prevented the Struts vulnerability making such an impact if:

- developers would have found the issue by adopting open source dependency scanning tools that integrate with their workflow using IDE plugins or code linters.
- any new build run by a CI server would automatically test application dependencies via a CI server plugin or a CLI invocation as a task. This would immediately flag the new vulnerability, breaking the CI job and forcing a remediation action before continuing.
- a monitoring solution was in place that notified developers of the new vulnerability in their dependencies.

Further monitoring and runtime insights into how the application behaves and what vulnerable functions it invokes could have alerted of vulnerabilities in the Struts library.

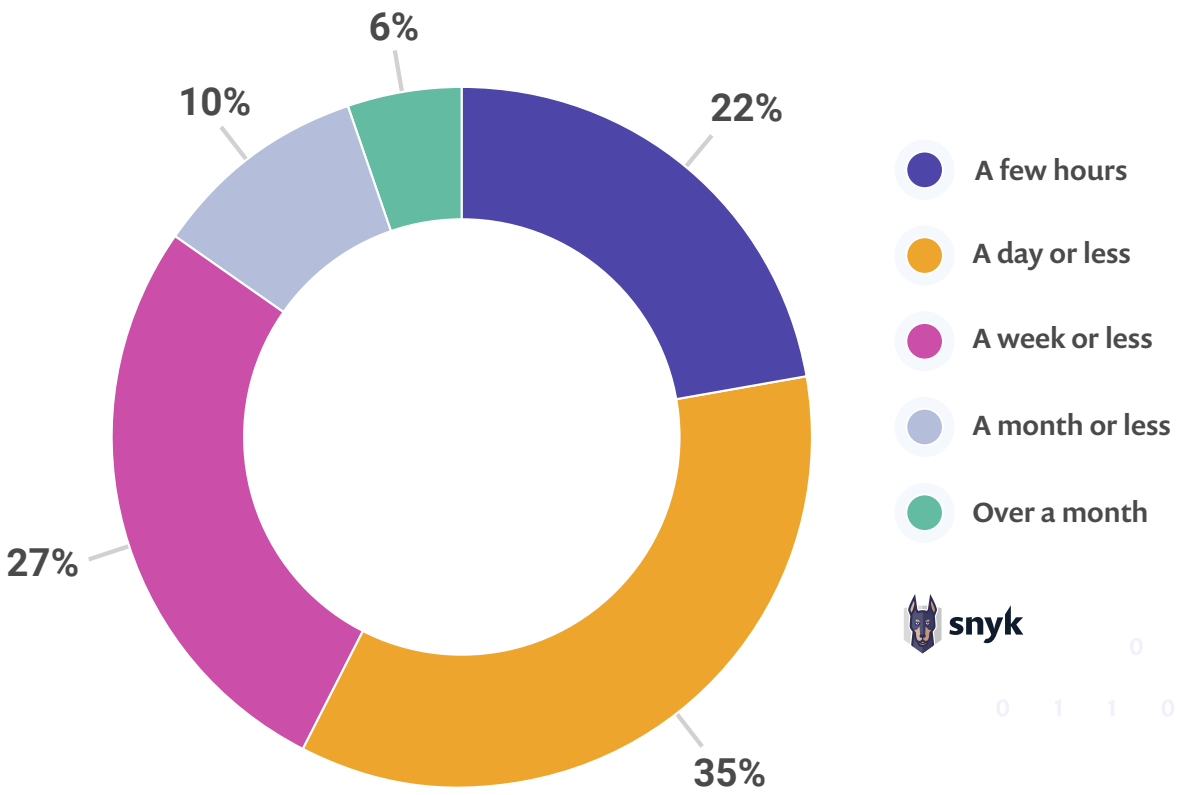
Releasing fixes

A crucial part of a responsible security disclosure is the speed of fix and roll out. It's important to be able to address the vulnerable issue as quickly as possible, thereby reducing the time it exists in the code, and also to provide sufficient time for users to upgrade to a fixed version, preferably before the issue is common knowledge.

As the nature of open source communities revolves around mostly volunteer work of developers (a BIG thank you to all the wonderful people who contribute to open source software—your kind work is very much appreciated and rarely acknowledged or appreciated publicly!), it is interesting to gauge how fast maintainers of open source software can react to a security vulnerability and provide a fix.

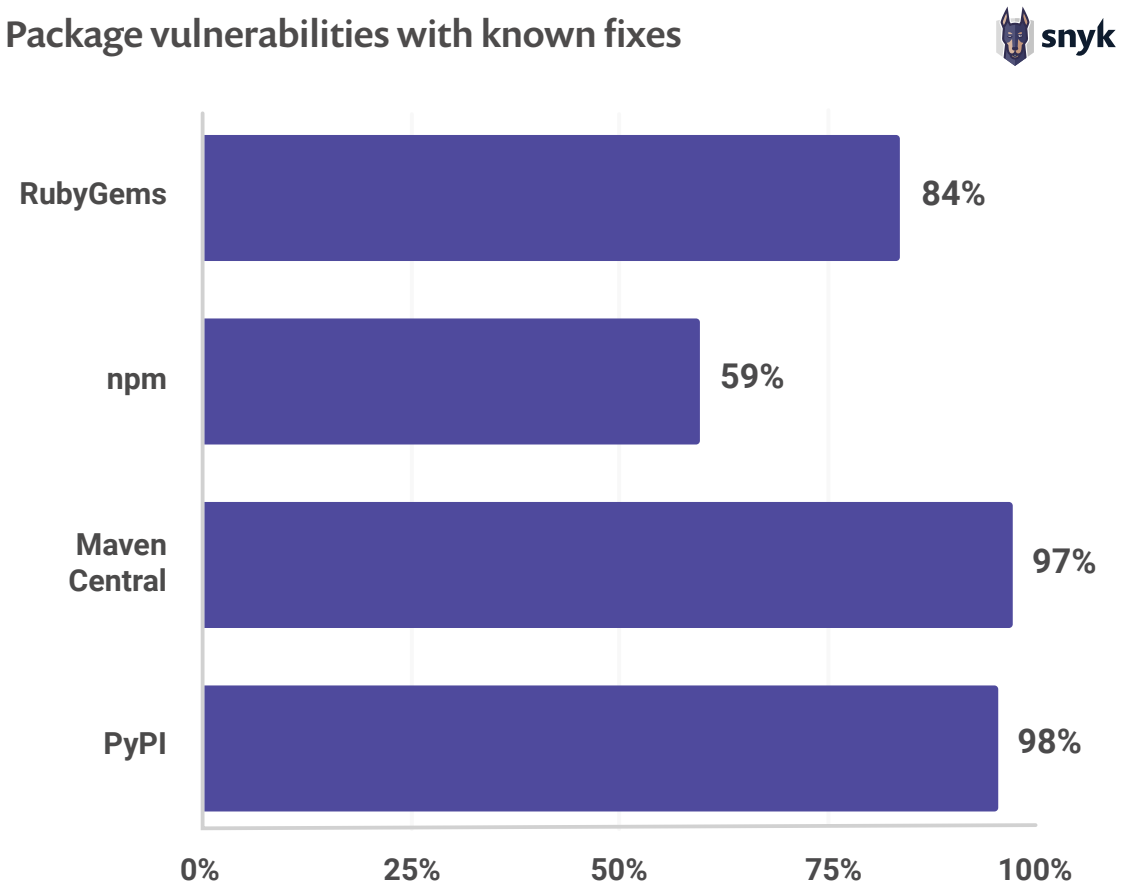
An overwhelming majority of users, totaling 84%, state they are likely to respond with a fix in less than a week. 56% are likely to address it within a day, while 22% state they can address a security issue within a few hours after the vulnerability has been reported—not all heroes wear capes!

Vulnerability report response times



Rate of fixing

Examining the Snyk vulnerability database we can determine which packages have released versions that contain vulnerability fixes. This paints a less than ideal picture for some ecosystems– looking at you JavaScript! Java and Python exhibit ecosystems with strong attention to security vulnerabilities, whereas JavaScript and Node.js as a whole show that only 59% of packages have known fixes for disclosed vulnerabilities.



Spotlight: Responsible security disclosures

A significant benefit of having a responsible disclosure policy is to keep users out of harm's way. When a vulnerability is reported and triaged in a confidential manner with the project maintainer it allows the maintainer to prepare a fix before the information is disclosed to the general public. If maintainers can act quickly and release a fix, then they provide a window of time during which their users can upgrade to the fixed version. This time window significantly decreases the number of users that consume the vulnerable versions.

We believe that having a responsible disclosure policy in place will also communicate the maintainer's high commitment to security. We recommend to use a badge on the project's homepage, and including a SECURITY.MD policy file in the project's repository as a good practice.

In the last report we found that maintainers who have a public-facing disclosure policy in place are far more likely to receive disclosures from users in confidence, than those who do not.

About 21% of maintainers with no public disclosure policy have been notified privately about a vulnerability, as compared to 73% of maintainers with a disclosure policy in place.

Websites are susceptible to web security vulnerabilities and would benefit from clear guidelines about web security policies.
An emerging proposal to aid with this is the SECURITY.TXT (RFC 5785) which has seen early adoption already. The purpose of such as policy file is to effectively communicate to security researchers the relevant contacts, preferred languages, exact policy and ways of communication, including public keys to securely and efficiently disclose security vulnerability.

The future of open source

Open source is a core part of virtually all software applications today. Even the Java and Node.js platforms are open source! There's no getting away from the obvious fact that open source is here to stay and a welcome part of modern software development. It's easier than it has ever been to create a new open source project, as well as use other projects from other members of the community. This speed of development and sharing has led to coding standards and practices varying greatly between open source projects, as it's not always easy for developers to think about the consequences of unintentionally sharing insecure code. In the great, wise, and slightly adapted words of Dr. Malcolm:

> Your developers were so preoccupied with whether or not they could, they didn't stop to think if they should.
>
> — *Dr. Ian Malcolm, Jurassic Park*

We'd like to conclude this report with some security advice for both open source project maintainers as well as those who consume open source dependencies. Oh, I guess that's pretty much everyone then!

Take action

As OS maintainers and developers there are actions you can take to improve the security in projects you own and contribute to.

Open source maintainers

As an open source maintainer, you should offer secure releases of your code and provide a communication strategy to those consumers in order to positively impact other projects and applications, ultimately benefiting your own projects as well.

- Practice secure code review with your peers if possible and follow secure-code best practices. Make security considerations part of your code review checklist and educate those who are reviewing so that they know what they should be looking for.
- Regularly audit your code base for vulnerabilities, through static and dynamic code analysis, for example, that can be automated into your development workflow and make it easier to catch vulnerabilities before they become public.
- Clearly define a simple process for communication of responsible disclosures, using your own policy or by referring to an existing program. To communicate your security awareness consider adopting a SECURITY.MD policy and a project badge that reflects the security health of the project.
- Implement a shift-left security strategy that provides your team the insight into security issues during development, CI, and even when pull requests are created to eliminate all chances of vulnerable code entering your projects.

Open source developers

As a consumer of open source components, it's your responsibility to fully understand the direct and indirect dependencies your projects use, including any security flaws that might exist in that dependency tree. Consider adopting the following security guidelines:

- Regularly audit your code base with a tool that automatically detects vulnerabilities in your third-party dependencies, providing remediation advice to your team and monitoring a project's dependencies even after it has been deployed.
- Follow responsible disclosure policies if you are reporting a security vulnerability to make sure you don't put users in harm's way. If you are unsure about how to do this, consider disclosing to a security company that work through the disclosure with you, such as Snyk's responsible disclosure program.
- Subscribe to the security communication channels of your open source dependencies, if they have them, so you're aware of any potential vulnerabilities as they are reported.

TL;DR - Report summary

Data in this report was collected from the following sources:

- Over 500 open source maintainers and users
- Internal data from the Snyk vulnerability database
- Hundreds of thousands of projects monitored and protected by Snyk
- Research taken from external sources published by various vendors
- Scanning millions of GitHub repositories and packages on public registries

Known vulnerabilities in application libraries

- Security vulnerabilities almost double in two years. In 2017 we saw a 43% increase in the number of vulnerabilities reported. In 2018, that total increased by a further 33% across all registries
- The number of Golang vulnerabilities grew in 2018 by 52%
- Since 2014, the number of vulnerabilities in the Snyk database has increased by an astonishing 371%, with npm vulnerabilities increasing by an incredible 954% and Maven vulnerabilities increasing by 346%

Security ownership

- 81% of users feel developers are responsible for security
- Only 28% of users feel security teams are responsible for security
- 68% of users feel that developers should own the security responsibility of their docker container images

Open source adoption

- GitHub saw a 40% rise in new organizations and new repositories created in 2018
- Almost one third of all repositories that exist on GitHub were created in 2018
- Growth in indexed packages from 2017 to 2018.
  - Maven Central - 102% growth
  - PyPI - 40% growth
  - npm - 37% growth
  - NuGet - 26% growth
  - RubyGems - 5.6% growth
- The CVE list reported a record-breaking number of vulnerabilities reported in 2018, which now totals more than 16,000 vulnerabilities in the database overall
- npm reported 304 billion package downloads for the entire year of 2018.
- Docker reported over 1 billion container downloads every 2 weeks over the last year, and about 50 billion to date
- Docker also reported 1 million new applications added into Docker Hub over the last year
- 78% of vulnerabilities are found in indirect dependencies
- On average, open source maintainers rate their own security knowledge as 6.6/10
- Only 3 in ten open source maintainers consider themselves to have high security knowledge
- One in four open source maintainers do not audit their code bases

Vulnerabilities in docker images

- Each of the top ten most popular default docker images contains at least 30 vulnerable system libraries
- 44% of scanned docker images can fix known vulnerabilities by updating their base image tag
- 20% of docker image scans had known vulnerabilities that simply required a rebuild of the image to reduce the number of vulnerabilities
- 37% of open source developers don't implement any sort of security testing in their CI and 54% of developers don't do any docker image security testing

Known vulnerabilities in system libraries

- 1597 vulnerabilities in system libraries with known CVEs were raised in 2018 for the Debian, RHEL and Ubuntu distributions
- In 2018, we tracked over four times more vulnerabilities found in RHEL, Debian and Ubuntu compared to 2017
- According to 451 Research, the adoption of application container technology is expected to grow by a further 40% in 2020
- Ruby's default docker image ships with 583 system library vulnerabilities

Vulnerability characteristics of each ecosystem

- In 2018 there were 11 typosquatting attacks for malicious packages published on the npm registry
- The number of XSS vulnerabilities is again on the increase. In 2018 the PHP Packagist ecosystem disclosed the most with 56 XSS vulnerabilities, followed by npm with 54, and Maven Central with 29
- Path and directory traversal vulnerabilities fiercely stand out in the npm ecosystem with record numbers of 146 and 143 disclosures in 2017 and 2018, respectively
- The number of Cleartext Transmission of Sensitive Information vulnerabilities has increased by 250% since 2016

Discovering vulnerabilities

- Almost 40% of open source users don't implement any sort of security testing during CI
- Over half of open source users test for vulnerabilities in their open source dependencies

The Snyk Vulnerability database

- CVE/NVD and public vulnerability databases miss many vulnerabilities, only accounting for 60% of the vulnerabilities Snyk tracks
- In 2018 alone, 500 vulnerabilities were disclosed by our proprietary research
- 72% of the vulnerabilities in npm audit were added to the Snyk vulnerability database first
- On average, Snyk discloses vulnerabilities 92 days sooner than they are published on npm-audit

How do maintainers find out about vulnerabilities?

- Almost half (48%) of respondents find out about a security vulnerability that is in their code from a public channel, such as a public issue
- 72% of users said they find out about vulnerabilities in their code when they review it

Inclusion to disclosure

- Of the seven libraries we analysed, the quickest time-to-fix from inclusion was 289 days. The median time is almost 2.5 years, and the worst case we saw was 5.9 years

Adopting fixes

- 84% of users state they are likely to respond to a fix in less than a week
- 22% state they can address a security issue within a few hours of a report
- 27% of users stated they do not have any proactive or automatic way to find out about newly discovered vulnerabilities in their applications.
- Only 36% of users confirmed that they use a dependency management or scanning tool to help surface vulnerabilities
- In the second half of 2018 alone, Snyk opened more than 70,000 Pull Requests for its users across Maven, RubyGems and npm ecosystems to remediate vulnerabilities in their projects

snyk

Snyk helps you use open source and stay secure.

Get started at snyk.io

Twitter: @snyksec

Web: https://snyk.io

Office info

London

1 Mark Square

London EC2A 4EG

Tel Aviv

40 Yavne st., first floor

Boston

WeWork 9th Floor

501 Boylston St

Boston, MA 02116

Report author

Liran Tal (@liran_tal)

Report contributors

Simon Maple (@sjmaple)

Guy Podjarny (@guypod)

Rachel Cheyfitz (@spinningrachel)

Report design

Growth Labs (@GrowthLabsMKTG)