Created by snyk

2017

The State of Open Source Security

The open source landscape is vast and only getting more diverse. The overall security of open source is an important measuring stick. We need to know where we stand today to know what we can do better.

Any attempt to try and provide a global view of the ecosystem's security health requires data. To help better understand how secure open source is and what we can all do to make it better, Snyk distributed and analyzed a survey that was filled out by more than 500 open source maintainers and users. We also looked at Snyk internal data based on more than 40,000 projects, as well as information published by Red Hat Linux and data we gathered by scanning millions of GitHub repositories and packages on registries.

This report summarizes those findings.

***For more information:** https://snyk.io | contact@snyk.io*

THE STATE OF OPEN SOURCE SECURITY

At a Glance

The number of open source packages indexed has exploded in the past year.

PERCENTAGE INCREASE:

Rubygems	10%
Python	32%
Maven	28%
npm	57%

There's no guarantee of the security knowledge of an open source maintainer.

Only 16.8% of maintainers consider their security know-how to be high, and 43.7% have never conducted a security audit on their code.

A test of 430,000 sites showed that 77% of them run at least one front-end library with a known security vulnerability.

The number of open source application library vulnerabilities increased by 53.8% in 2016. Already in 2017, they've increased by an additional 39.1%.

Once a fix has been released, maintainers need to notify users so that they can secure their applications.

88% of maintainers inform users through release notes, while only 9% file for a CVE ID and only 3% inform a vulnerability monitoring service.

Vulnerabilities generally remain undiscovered for a long time.

The median time from inclusion to discovery for in application libraries: **2.5 years**

Once discovered, maintainers generally react very quickly.

The median time from when a vulnerability is disclosed to when it is fixed: **16 days**

The National Vulnerability Database (NVD) has seen a large increase in vulnerabilities, but coverage can be an issue.

PERCENTAGE OF VULNERABILITIES COVERED BY THE NVD:

npm	11%
Rubygem	67%

The Open Source Landscape

Open source as a whole is thriving. Both Forrester and Gartner have stated that somewhere between 80-90% of all commercial software developers use open source components within their applications. Organizations all over the world, from every vertical imaginable, use open source code to power their business.

Adoption

The use of open source components is booming, no matter which ecosystem you look at.

Through September of this year, 6.3 billion Python packages have been downloaded using PyPi and over 87.3 billion Node packages using npm.

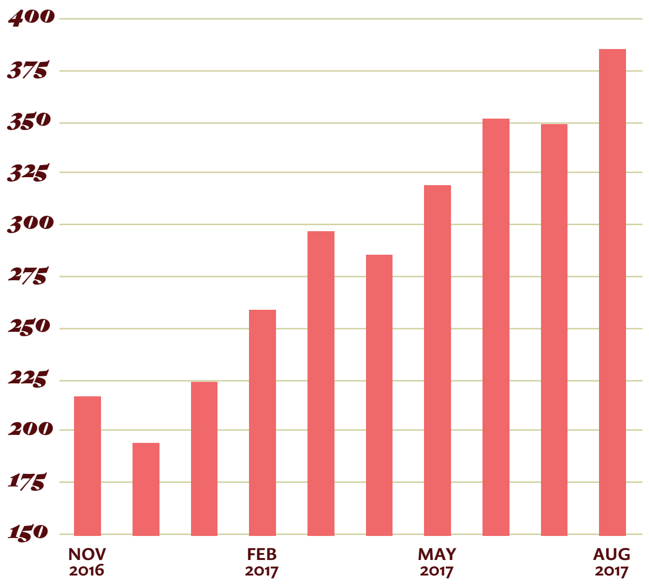
In the last year (October 1, 2016, to October 1, 2017), the number of open source packages indexed has exploded. The number of Rubygems has increased by 10.3%, the number of Python libraries by 32%, the number of Maven artifacts by 28% and the number of npm packages by 57%.

Meanwhile, the number of public applications available on Docker Hub is now over 900,000, up from around 460,000 a year ago.

Simply put, there has never been a wider variety of open source components available, and it's never been easier for us to use them.

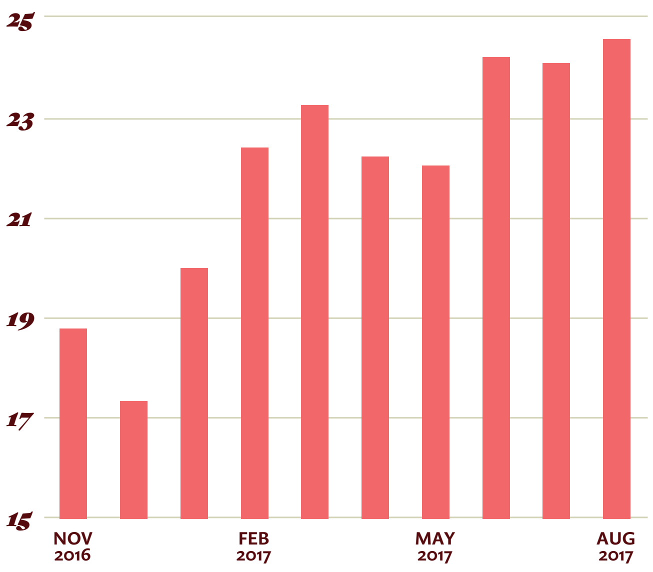
npm Packages Downloaded, Per Day Average

PER DAY AVERAGE, IN MILLIONS



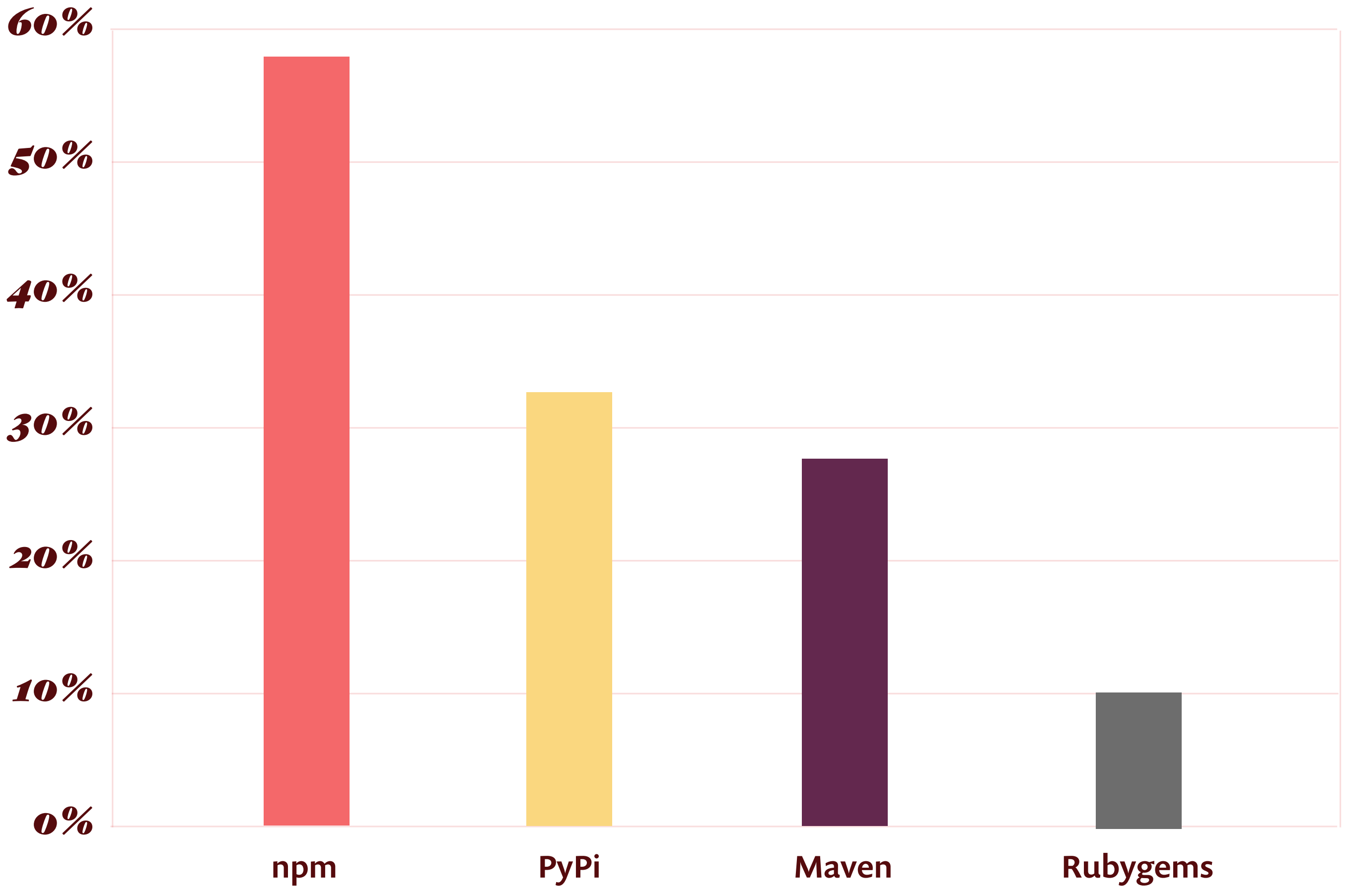
PyPi Packages Downloaded, Per Day Average

PER DAY AVERAGE, IN MILLIONS



Percentage Increase in Total Packages Indexed

OCTOBER 2016 – OCTOBER 2017



Risks and Impact

Chances are if you're using open source code, security is a top concern. In Github's recent Open Source Survey, 86% of users said security was extremely or very important. But there are natural risks with open source — you don't know who you're downloading from, and the security standards and know-how of maintainers differs dramatically between the different open source projects.

Known vulnerabilities in open source components also makes for a very attractive attack vector. Once they have been publically disclosed, exploits are quick to follow.

Consider the Struts2 vulnerability (CVE-2017-5638) from this year. It was disclosed March 7th, 2017. Over the next six days, Imperva reported seeing thousands of attempted exploits, coming from 1,323 different IP addresses from over 40 countries.

Spotlight on Equifax

One of the most notable breaches of 2017 was the one suffered by Equifax, resulting in millions of records being exposed. The breach was accomplished by exploiting a known vulnerability in the Apache Struts2 package which made it possible for a remote attacker to send a malicious request that would allow them to execute arbitrary commands.

The timeline of the vulnerability is what stands out. The vulnerability was publicly disclosed, and fixed, on March 6, 2017. One day later, exploit scripts started to appear in the wild. Two months later, on May 13, the Equifax breach began. It took another two months (until July 29) to detect it. This vulnerability went from public to active exploits in one day, to breach in two months. It's never been more critical to prioritize open source security.

Security Posture Of Open Source Maintainers

The same freedom that allows open source to thrive also presents challenges from a security standpoint. There are no "rules" that open source development must follow to ensure a secure delivery, so the security posture differs dramatically from one project to another. The challenge for open source maintainers is that they are often maintaining these open source resources in addition to other day-to-day work, and so don't have an abundance of time and resources to put towards securing their projects.

While the vast majority of people contributing code to open source are good actors, only 16.8% of the maintainers we surveyed said they consider their security know-how to be high.

One method of reducing the number of vulnerabilities included in an application is to conduct regular security audits—whether manually or by hiring a pentester—to see what security risks may be as yet undiscovered. Yet 43.7% of open source maintainers say they have never had a security audit done on their code, and another 31.8% say they've only audited their code once or twice in the lifetime of the project.

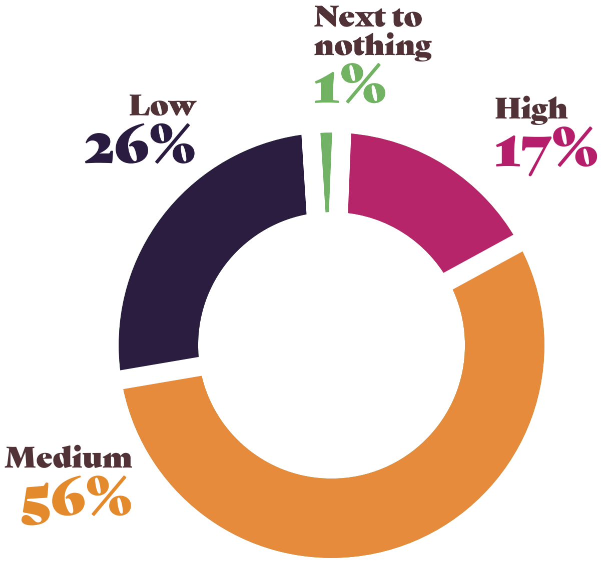
43.7%
of open source maintainers have never had a security audit

16.8%
of open source maintainers consider security know-how to be high

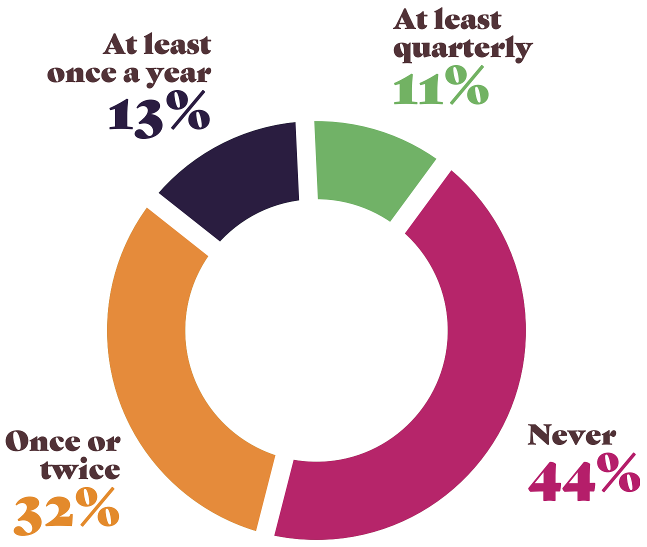
Currently, no standard exists for documenting basic security information about open source projects, and ad-hoc use is minimal. Of the top 400,000 public repositories on Github, less than 10,000 currently have such a file in place—around 2.4% in total.

Lacking any public information can make it very difficult to assess the overall commitment to security from any individual open source package or project, as well as to understand how to disclose newly discovered vulnerabilities to the open-source maintainers.

How Maintainers Rank Their Security Expertise



How Often Maintainers Audit Their Code



How Many Vulnerabilities Are There?

Mistakes will happen. According to a study by Carnegie Mellon University, for every thousand lines of code in commercial software, there are somewhere between 20-30 bugs. And as bugs are created, vulnerabilities will inevitably follow.

Known Vulnerabilities In Application Libraries

The number of published vulnerabilities in application libraries has been steadily increasing since 2009 and shows no signs of slowing down. Last year (2016) saw a 53.8% increase in the number of open source security vulnerabilities published (based on the Snyk database which tracks npm, Maven, Pip, Rubygems and Go), and this year has already increased an additional 39.1% over last year.

The problem extends to the front-end as well. In a test we conducted of over 430,000 sites, 77% of them were running at least one library with a known security vulnerability in place.

Known Vulnerabilities In System Libraries

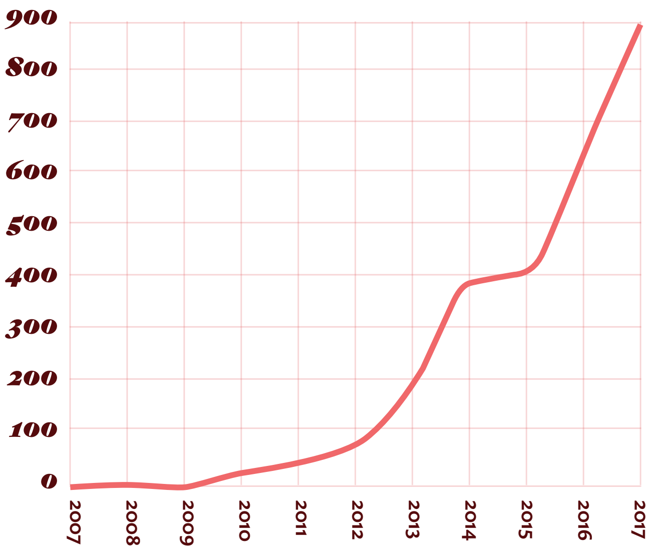
Looking at Red Hat Linux itself, we see the opposite trend. The number of vulnerabilities impacting Red Hat Linux has been steadily decreasing since 2012, dropping by 62% since then.

In a test we conducted of over 430,000 sites, 77% of them were running at least one library with a known security vulnerability in place.

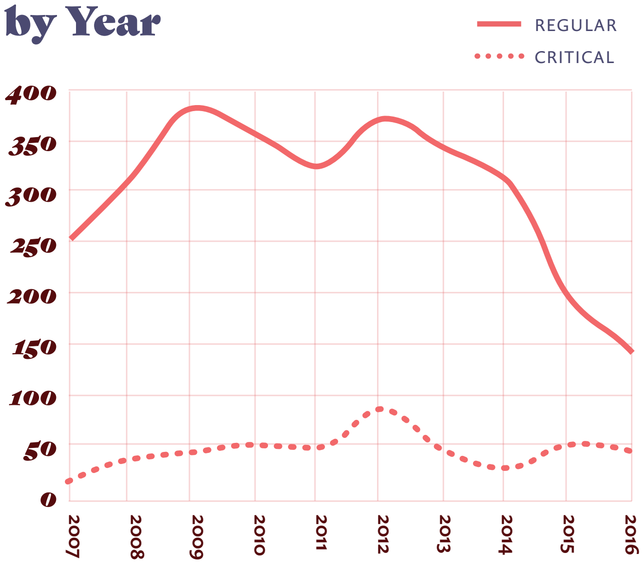
53.8%

increase in the number of open source security vulnerabilities published in 2016

Open Source Vulnerabilities Published by Year



Red Hat Linux Vulns by Year

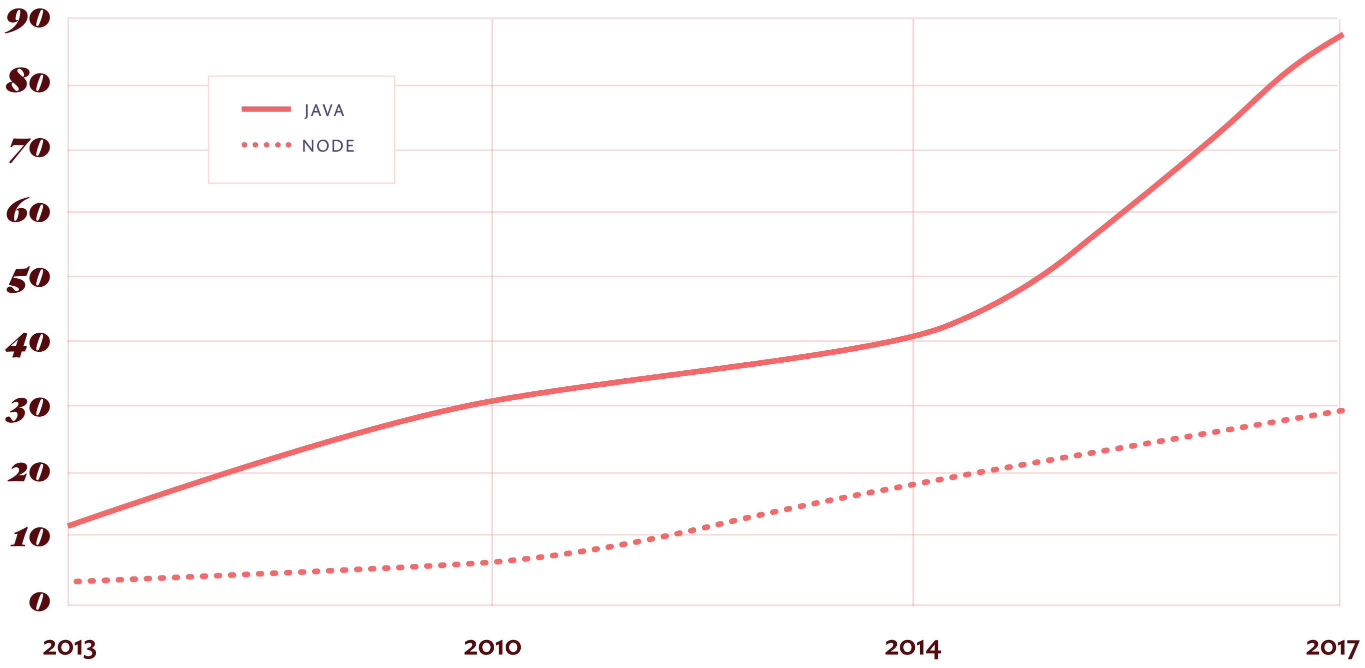


Trends in Severity

Raw vulnerability counts are one thing, but it's fair to question whether or not they're actually impactful. A high severity vulnerability found in a library with only a few downloads would inflate these numbers somewhat artificially.

We took it a step further and looked at high severity vulnerabilities found in npm packages with over 50,000 downloads, as well as high severity vulnerabilities found in Java packages based on how prominent those packages are with our customers. In both cases, we found that the number of high severity vulnerabilities with high impact has been steadily increasing. So far in 2017, we've seen a very similar trend in growth.

Critical Java & Node Package Vulnerabilities



Vulnerabilities in Docker Containers

Containers face a similar challenge. Containers provide great convenience by pulling all the libraries, packages, settings and more into a pre-configured environment.

But as with anything else, if they're not kept up to date, known vulnerabilities tend to creep in. Of the top 1,000 most popular Docker containers, 76.6% have known vulnerabilities, and 62.3% of them contain at least one high-severity vulnerability.

76.6%
of the top 1,000 Docker containers have known vulnerabilities

High Severity Vulnerabilities in System Libraries

Once again, Red Hat Linux vulnerabilities paint a different picture. The number of critical vulnerabilities impacting Red Hat peaked in 2012 but otherwise has stayed mostly consistent from year to year.

Red Hat Linux seems to be finding some level of stability. While that is not the case of application libraries—where vulnerability counts and severity is increasing—it does make us optimistic that better security is achievable with a little bit of work.

The Open Source Security Lifecycle

There are a lot of steps involved in the lifecycle of an open source security vulnerability—from discovery through final adoption of fixes. Each part of the process is important in its own way, and ultimately plays a role in the overall state of security.

Discovering Vulnerabilities

How vulnerabilities are discovered and how long they go unnoticed.

Releasing Fixes

The time it takes from disclosure to when a vulnerability is addressed.

Notifying Users

How users are made aware of fixes to vulnerabilities.

Adopting Published Fixes

How quickly users adopt fixes once they're published.

Discovering Vulnerabilities

The first period to look at is how long it takes for a vulnerability to be discovered after it has entered the library.

The median time from when a vulnerability was introduced (included in a formal release) in an application library and when it was publically disclosed is 2.53 years (924 days). Studies have repeatedly shown that for Linux vulnerabilities, on the other hand, the time from introduction to public disclosure is around five years. This amplifies the importance of frequently conducting security audits—it's very likely that there are numerous other vulnerabilities that have just not yet been disclosed. Formally auditing code is an important step for maintainers to take to ensure the security of their projects.

How Do Maintainers Find Out About Vulnerabilities?

How these issues are disclosed and addressed are critical characteristics of the project's overall security. In an ideal world, vulnerabilities should be reported to maintainers privately, letting them fix it before the world learns about the flaw.

In practice, the process of vulnerability disclosure doesn't always go smoothly. In part, this is because public facing disclosure policies are few and far between. 79.5% of maintainers said that they had no public-facing disclosure policy in place. This lack of clear communication about disclosure makes it challenging for researchers to report vulnerabilities in a private and responsible manner.

As a result, maintainers say they learn about vulnerabilities in a variety of different ways. The vast majority of vulnerabilities are discovered by someone outside of the project—only 25% of maintainers say they've discovered a vulnerability themselves. Open source maintainers are just as likely to receive a private disclosure (31.4%) as they are to have a public issue filed on their repository (30.3%).

Maintainers who have a public facing disclosure policy in place are far more likely to receive a private disclosure than those who do not. About 21% of maintainers with no public disclosure policy have been notified privately about a vulnerability, compared to 73% of maintainers with a disclosure policy in place.

There may also be some confirmation bias in play. While only 13.5% of maintainers with a disclosure policy in place say they've never had a security issue to their knowledge, 32% of maintainers with no policy say they've never had a security issue to their knowledge. While this could be the case of maintainers putting a policy in place only after they've first had a security issue reported to them, given the complications involved in disclosing without the guidance provided by a disclosure policy, it's just as likely that many vulnerabilities have been found and not disclosed.

Vulnerabilities: *Inclusion to Disclosure*

5.9 years
Maximum time from inclusion to disclosure

0 days
Minimum time from inclusion to disclosure

2.5 years
Median time from inclusion to disclosure

Vulnerabilities: *Disclosure to Fix*

94 days
Maximum time from disclosure to fix

0 days
Minimum time from disclosure to fix

16 days
Median time from disclosure to fix

Releasing Fixes

The next period to look at is the "days of risk": the time it takes from disclosure to when a vulnerability is addressed. The moment a vulnerability is publicly disclosed, the risk rises dramatically and continues to increase until the vulnerability has been fixed and you can take action. For this, the picture is much rosier.

Handling Vulnerabilities With Urgency

Open source maintainers are very eager to respond to security vulnerabilities as quickly as possible. 34% of maintainers said they could respond to a security issue within a day of learning about it, and another 60% said they could respond within a week.

We see this reinforced by the response to the most impactful npm vulnerabilities we saw this year. The median time from disclosure to when a fix has been included in a formal release was 16 days.

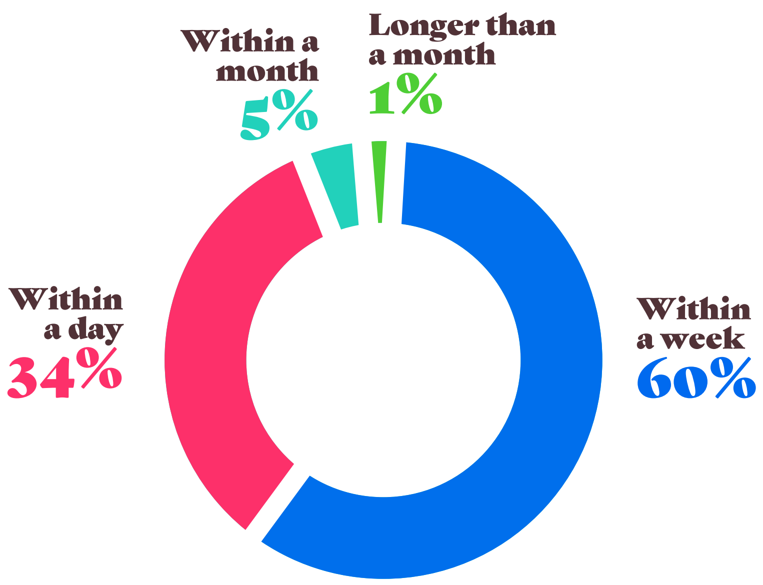
Similarly, the majority of vulnerabilities in Red Hat Linux are also fixed soon after disclosure.

In 2016, 69% of Red Hat Linux vulnerabilities were fixed within a day of their public disclosure, and 90% were fixed within 14 days of their public disclosure. In fact, the pace of fixing has been steadily improving since 2014.

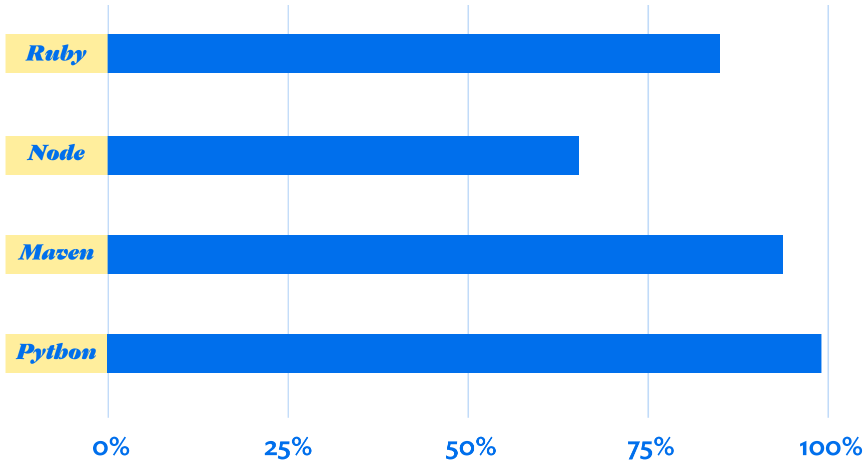
Rate of Fixing

While the cadence for fixes was pretty good, the reality is that not all vulnerabilities will be fixed. Looking at the Snyk database, the number of vulnerabilities with available fixes (once you exclude issues like malicious packages) varies from ecosystem to ecosystem.

How Quickly Could Maintainers Respond to a Vulnerability?



Percentage of Vulnerabilities with Known Fixes Available



Spotlight on Nokogiri and LibXML

Sometimes the chain of dependencies can put open source projects in a tough spot. This is the case for the massively popular Nokogiri library. Nokogiri uses the libxml2 library to allow users to easily parse and extract data from XML, SAX, Reader or HTML documents.

Unfortunately, libxml2 has a vulnerability that can make it easy for an attacker to conduct an XML External Entities (XXE) attack (an attack that leads to issues like denial of service and private information disclosure).

While the vulnerability was published back in November of 2016, libxml2 has yet to incorporate a fix. This has left Nokogiri in a tight spot where the vulnerability of the library it depends on has ended up leaving it in a vulnerable state. Nokogiri has tried to dissuade users from falling victim to the attack through making the default configuration as secure as possible, but until libxml2 addresses the issue, Nokogiri will remain vulnerable.

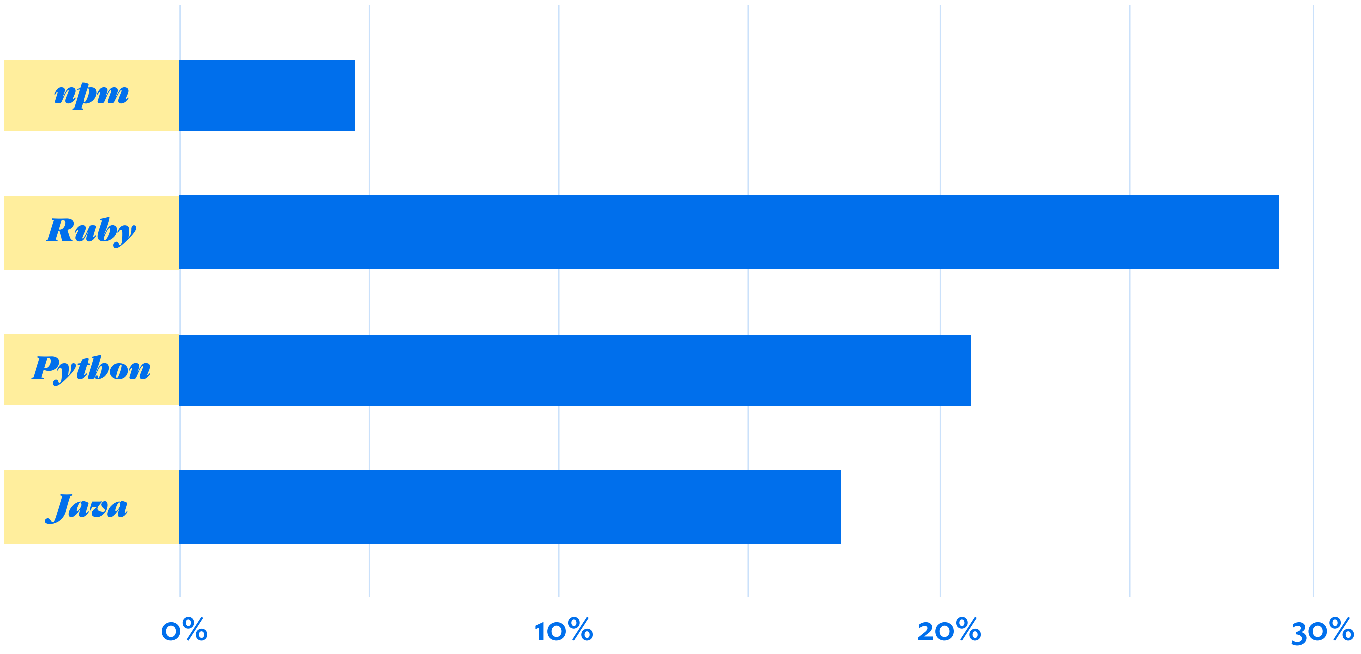
Most of the time, fixes aren't backported to other version streams. Only 16.1% of the vulnerabilities in the Snyk DB have fixes backported to other versions streams. The situation is pretty much the same across ecosystems, except npm where backporting is particularly rare: only 4.1% of npm fixes are applied to other version streams.

The lack of backporting makes it hard for many to upgrade to the new fix, as moving to a new major version of a library requires time and effort to make any updates necessary to conform to the changes the library made.

16.1%

Vulnerabilities in the Snyk database that have fixes backported to other versions streams

Percentage of Fixes Backported to Other Version Streams

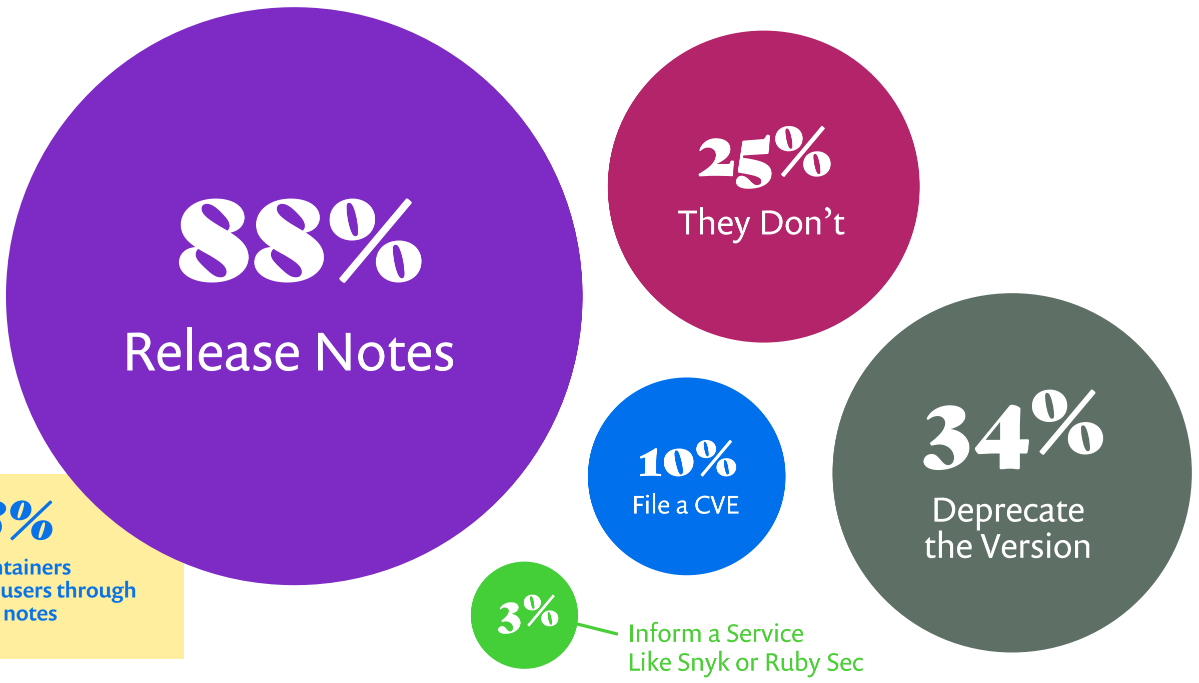


Notifying Users

After a vulnerability has been addressed, users must be made aware quickly so that they know what steps to take to secure their applications.

How Do Maintainers Tell Users About Security Issues?

(RESPONDENTS WERE ABLE TO SELECT MULTIPLE RESPONSES)



88%
of maintainers inform users through release notes

The majority of maintainers, 88%, say they inform users through the release notes if there has been a security issue and about 34% of maintainers will deprecate the latest vulnerable version to encourage adoption of the fix.

Of significant concern is the fact that an alarming 25% of maintainers say they do not do anything to inform their users if there's a security issue. Without any communication to the user about the vulnerability, it's highly unlikely that users will quickly adopt the latest fix. Instead, the vulnerable version is likely to stay in production long after users could have been protected from it.

One step not taken very often is the process of informing a database or vulnerability monitoring service. Ideally, each of these vulnerabilities winds up in a database of known vulnerabilities for consumers to peruse. Only 3% of maintainers say they inform a service, such as Snyk, about the vulnerability.

Finally, 8.9% of maintainers say they file for a Common Vulnerabilities and Exposures (CVE) ID for the vulnerability–a small percentage that hints at some of the trouble being encountered by CVE.

Spotlight on Next

Next, a framework for server-rendered React applications, was discovered to have a directory traversal vulnerability this year. The vulnerability has been in Next since its initial release in October of 2016. The vulnerability was disclosed May 31, 2017, and on June 1, Next released a new version of the library with a fix in place.

Not only does Next deserve credit for their rapid remediation, but their response was equally applause-worthy. While many times a security fix receives a one-liner in library release notes, Next took the opposite approach. In the release notes for the fixed version, they provided detailed information about the vulnerability, how to address it, who was impacted and where to stay up to date on further security issues that may arise.

CVE, CPE, and NVD

The U.S. Government created and sponsors the Common Vulnerability Enumeration (CVE) list—a free dictionary of vulnerabilities. The CVE is an attempt to provide a single destination for known vulnerabilities—not just for open source and application dependencies, but also for consumers to peruse.

CVE itself is a list, not a database, and contains minimal information about the various vulnerabilities it references. Providing more information is the National Vulnerability Database (NVD). The NVD pulls from the CVE list (though not always right away) and provides more information about the vulnerabilities—from remediation details to providing a Common Product Enumeration (CPE) that helps to pinpoint which product or application contains the vulnerability and a Common Vulnerability Scoring System (CVSS) score to provide some indication of the severity and characteristics of the vulnerability.

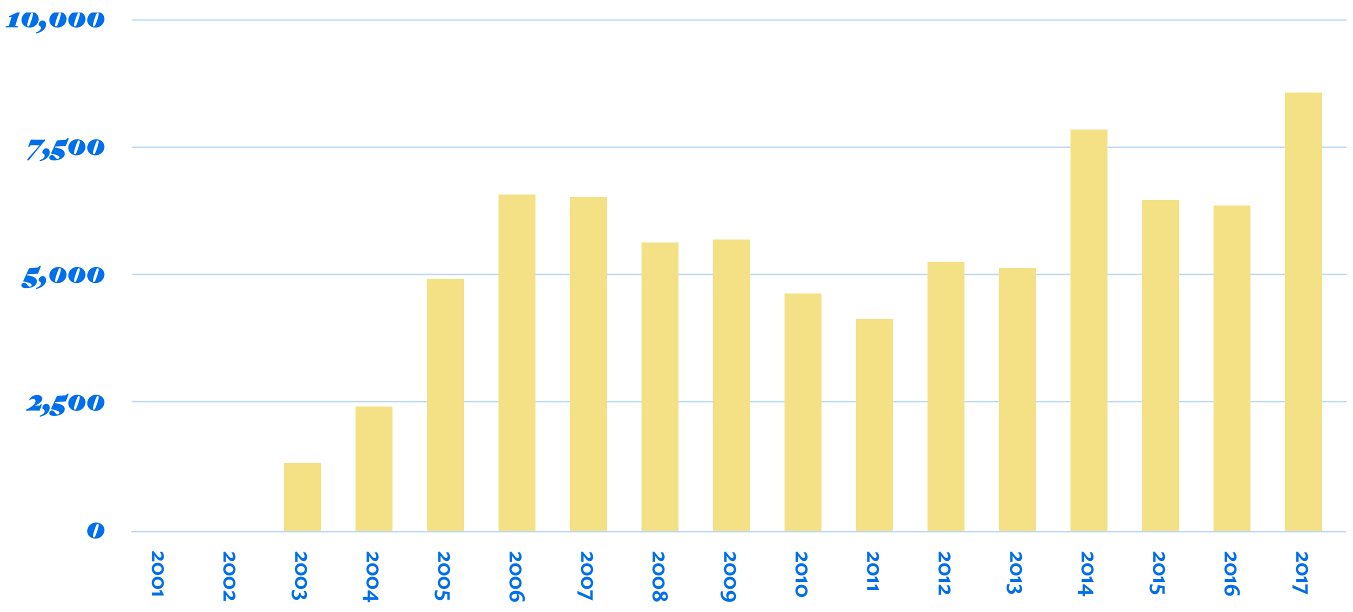
Overall, adoption of the NVD has been positive. Since its launch in 1999, the NVD has grown to over 81,000 vulnerabilities and shows no signs of slowing down. Already in 2017, over 8,500 vulnerabilities have been added to the database—more than any other year in the history of its existence.

This federated approach to maintaining the vulnerability database is effective, but many vulnerabilities are still missed.

11%
npm vulnerabilities in the Snyk database that are covered by NVD

67%
Rubygem vulnerabilities in the Snyk database that are covered by NVD

CVE By Year



This federated approach to maintaining the vulnerability database is effective, but many vulnerabilities are still missed.

Comparing to our open source vulnerability database, CVE/NVD coverage varies dramatically depending on the language and ecosystem. For example, NVD covers only 67% of Rubygem vulnerabilities and only 11% of npm vulnerabilities.

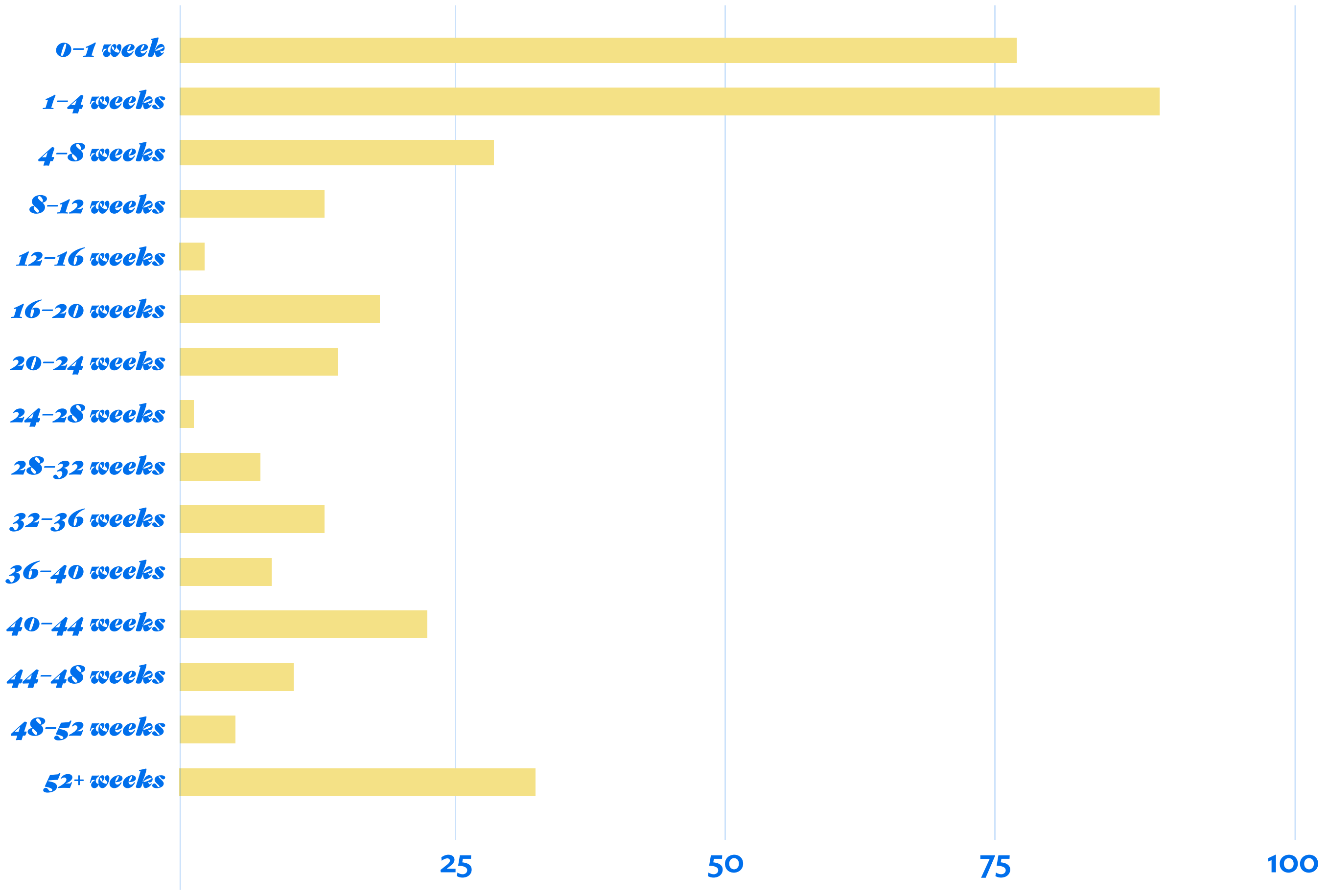
The other challenge CVE/NVD faces is the pace. While a CVE may get applied relatively early, it frequently takes a long time for that same vulnerability to make its way into the NVD and get a CVSS score and a CPE.

53% of the application library vulnerabilities were public (whether in a vulnerability database, library release notes or another mechanism) for four weeks or longer before being added to the NVD. The long tail, in this case, is very long. 28.9% of vulnerabilities were public for longer than half a year, and 9.4% for longer than a year before being found in the NVD.

Part of the reason why CVE lags behind is that most open source maintainers don't participate in filing CVEs when new vulnerabilities are discovered. As discussed before, when we surveyed maintainers only 8.9% say they have filed a CVE when a vulnerability was found. This will continue to be a challenge, unfortunately, for any attempt at a federated approach to vulnerability management.

...most open source maintainers don't participate in filing CVEs when new vulnerabilities are discovered.

How Long Were Vulnerabilities Public Before Being Added to NVD/CVE



Adopting Published Fixes

The final timeline to look at is how quickly users adopt fixes once they've been published. This can be challenging, as users must both learn the fix exists, and then make sure the new version doesn't break their application. In other words, this can take some time.

According to our survey, developers use a variety of methods to keep their dependencies up to date—including occasional sweeps to bump versions (46.9%), using tools to alert them to vulnerabilities (41.6%), using tools to alert them to new versions of dependencies (37.2%) and word of mouth (8.6%).

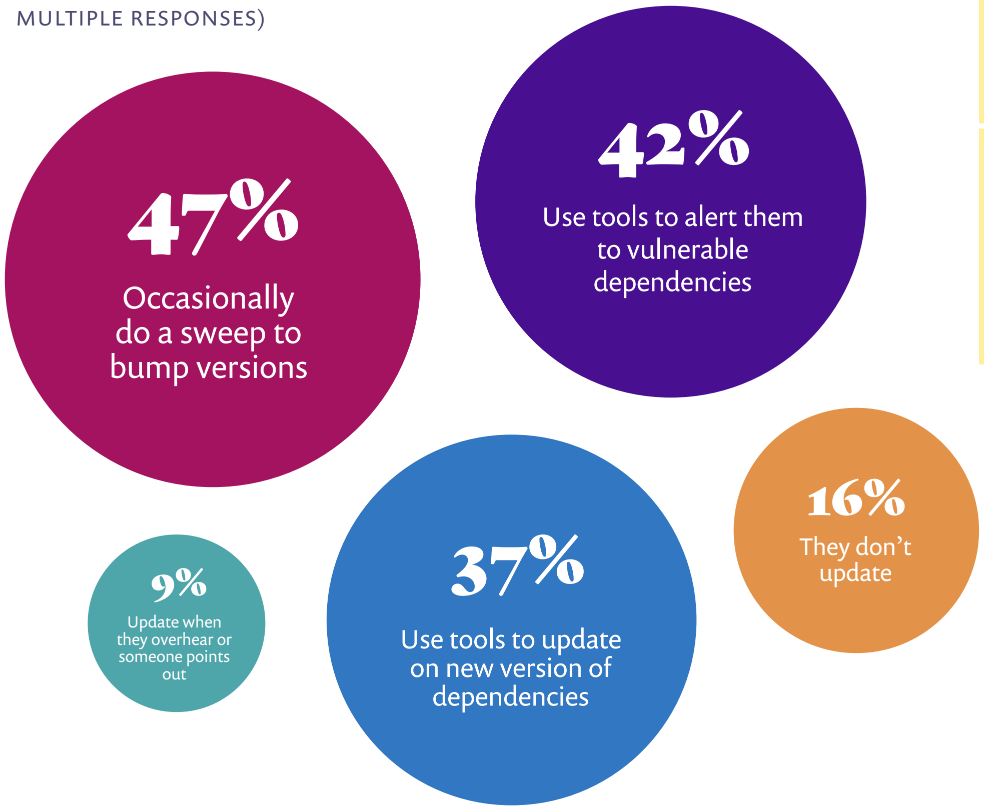
Surprisingly, 16.3% of users say they don't update their dependencies. And while at first glance, the number of users taking advantage of tools to help them manage their dependencies seems alright, there's a lot of overlap. Quite a few users use both a tool to alert them to new versions and a tool to alert them to security issues.

In fact, 38.7% of users don't use tooling to help keep their dependencies up to date. This inevitably leads either to situations where dependencies are not updated as frequently as they should be, or to situations where a lot of strain is put on the development process.

In the absence of tooling, an organization that takes security seriously must manually verify and validate new versions of dependencies to ensure its up-to-date and free of known vulnerabilities. In the face of the rapid rate of change within the open source community, this manual approach becomes a troublesome bottleneck and is unmaintainable.

How Do Users Keep Dependencies Up to Date?

(RESPONDENTS WERE ABLE TO SELECT MULTIPLE RESPONSES)



38.7%

of users don't use tooling to help keep their dependencies up to date

16.3%

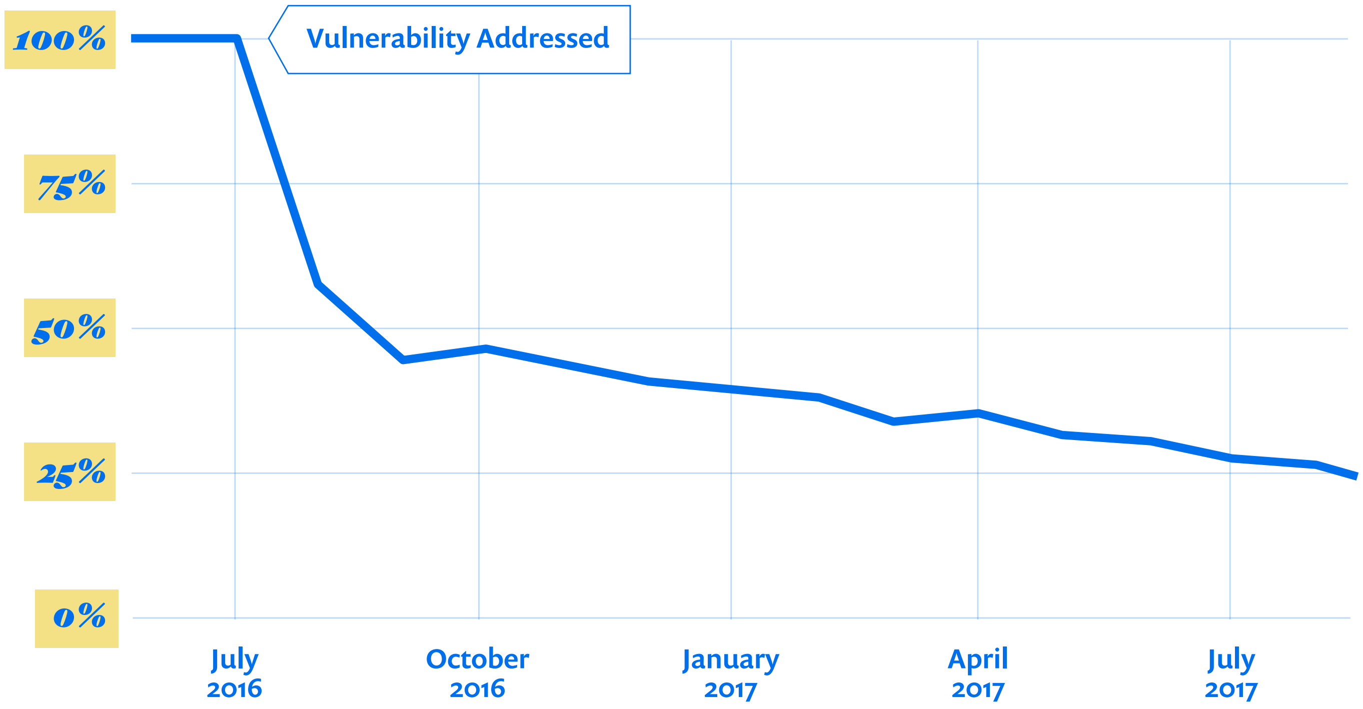
of users say they don't update their dependencies

Download Numbers of Vulnerable Library Versions

The pace of adoption tends to have a very long tail. Consider the popular Python “requests” library. The “requests” library is is one of the most popular Python packages with just under 12,000,000 downloads per month over the past year, and it was hit by an HTTP Request Redirection vulnerability in 2016. Given its tremendous popularity, we can zoom in on the versions being used to give us an idea of how quickly users adopt fixes.

From May of 2016 through July of 2016 (the three months before the vulnerability was addressed), vulnerable versions of the “requests” package made up ~99% of all downloads. In August 2017, the month where 2.11.0 was released, vulnerable versions plummeted to 56.6% of all downloads. After that initial burst, however, the long tail kicked in. In September 2017, 22.4% of all downloads of the “requests” package were still vulnerable versions, despite the fix coming just over a year prior.

Vulnerable versions of Python’s “requests” downloaded as % of total



The Future of Open Source

Open source is booming, and there is quite literally no sign of it slowing down. But while the benefits of open source are by now well known, knowledge of the risks are still a work in progress. But that's primed to change.

It's clear we have some room for improvement, but it's also clear we have a lot of opportunities to do so. It's easy to see that maintainers are eager to make their projects more secure and that users want to make security a priority in their open source consumption. It's just a matter of ironing out the wrinkles a bit.

If you're a maintainer, take three simple steps to get started:

1. **Make sure your project has a public facing disclosure policy** so that anyone who finds vulnerabilities can quickly report them to you.

2. **Run regular audits and security checks** against your codebase to make it easier for you to catch vulnerabilities before they become public.

3. **Make it clear to users of your project that you care about security.** Have detailed information that is clearly presented to them when a security issue is addressed, so they know exactly how to proceed.

If you're pulling open source components into your application, there are a few things you can do as well:

1. **Use a tool to automatically detect known vulnerabilities in your third-party components** so you can fix them as quickly as possible.

2. **Contribute back.** Most of the time, maintainers are working hard on open source projects in addition to their day-to-day work. Rolling up your sleeves and helping address issues is one of the best ways to ensure your favorite project stays healthy and secure.

3. **Report vulnerabilities as responsibly as possible.** If you find something amiss, look for a private way to tell the maintainers about it. If there's no public disclosure policy, see if you can discretely inform them through email or some other form of communication.

Securing open source is not something that will happen overnight. As we've seen in this report, there are many different considerations and factors that come into play. But together, with all of us making a concerted effort to take baby steps to improve our security posture, we can improve the state of open source security, and in the process, ensure that it remains a thriving and vibrant ecosystem.