The 6th Annual Report on Global Open Source Software Development

PRESENTED BY

IN PARTNERSHIP WITH

Contents

Introduction 4

CHAPTER 1
Open Season on Open Source 5

Software Supply Chain Attacks: Past and Future 6

Rise of Next-Gen Software Supply Chain Attacks (2015-2020) 7

Speed Remains Critical When Responding to Legacy Software Supply Chain Attacks 10

CHAPTER 2
Open Source: Supply and Demand 12

JavaScript 13

Java 14

.NET 14

DockerHub 14

CHAPTER 3
Identifying Exemplary Open Source Suppliers 15

Researching the Best Performing OSS Projects 16

Finding Different Behavioral Groups 16

- Exemplars 16
- Laggards 17
- Cautious Teams 17

Projects with Updated Dependencies Are More Secure 18

Guidance for Open Source Project Owners and Contributors 19

Guidance for Enterprise Development Teams 19

CHAPTER 4
How High Performance Teams Manage Open Source Software Supply Chains 20

Survey of Open Source Management Practices 21

- Comparing High Performers vs. Low Performers 23
- Comparing High Performers vs. Security First 23

Variables Most Impacting Performance and Risk Management 24

- Influencing Risk Management Outcomes 24
- Influencing Productivity Outcomes 26
- Influencing Job Satisfaction 27

Guidance for Enterprise Development Teams 27

Patterns Across OSS Component Updates: Easy, Difficult, and Planned 28

CHAPTER 5
The Trust and Integrity of Software Supply Chains 31

1 in 10 OSS Downloads Are Vulnerable 32

Enterprises Rely on Code From 3,500 Suppliers, But Quality Varies 33

OSS Components Make Up 90% of a Modern Application 33

21% of Enterprises Experienced Open Source Breaches 34

CHAPTER 6
The Changing OSS Landscape: Social Activism and Government Standards 35

Social Activism and Open Source Software 36

Governments Apply New Standards to Secure Software Supply Chains 36

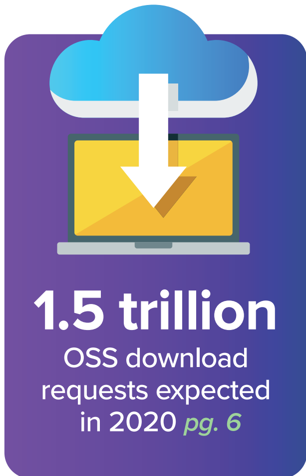
- United States 36
- United Kingdom 38
- Australia 39

Summary 40

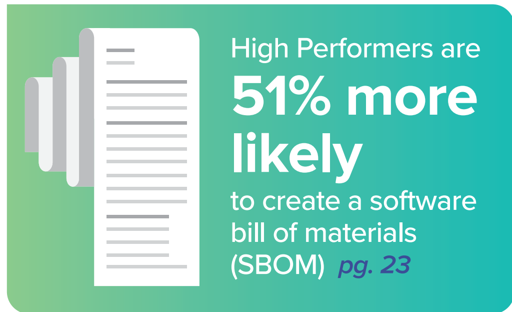
Sources 41

Appendix A 42

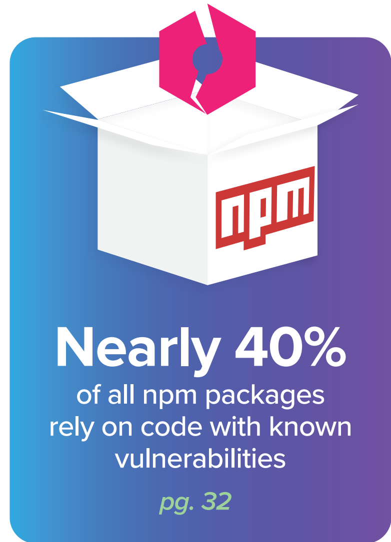
Appendix B 43

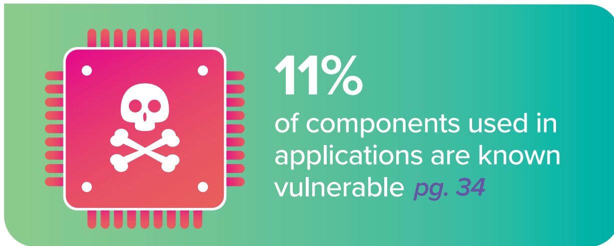

1.5 trillion
OSS download requests expected in 2020 pg. 6

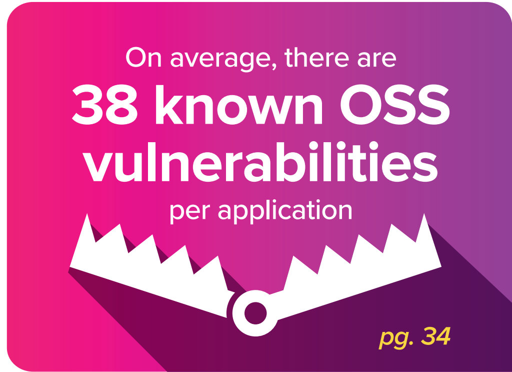
High Performers detect and remediate OSS vulnerabilities
26x faster
pg. 23

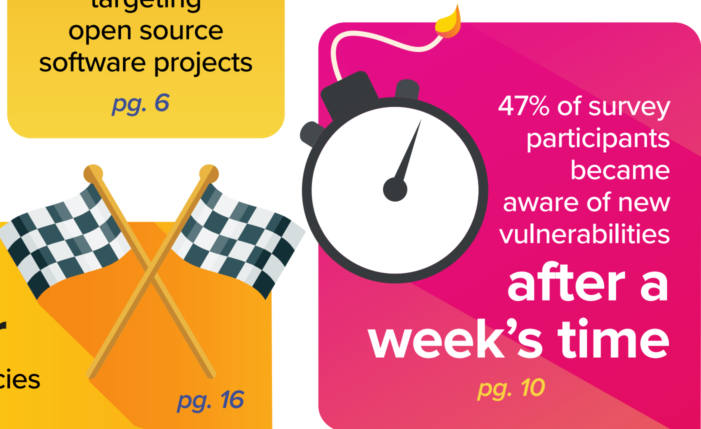

High Performers are
51% more likely
to create a software bill of materials (SBOM) pg. 23

430% YOY growth
in cyber attacks targeting open source software projects
pg. 6

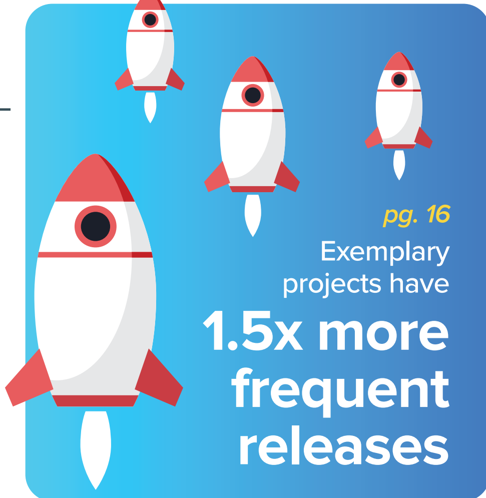

npm
Nearly 40%
of all npm packages rely on code with known vulnerabilities
pg. 32


11%
of components used in applications are known vulnerable pg. 34

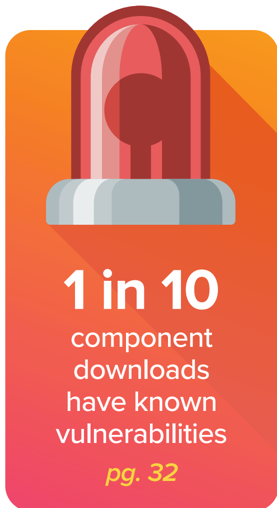

On average, there are
38 known OSS vulnerabilities
per application
pg. 34


47% of survey participants became aware of new vulnerabilities
after a week's time
pg. 10

NIST
NIST introduces
new standards
that call for SBOMs and OSS security checks pg. 38


pg. 16
Exemplary projects have
1.5x more frequent releases

Exemplary projects are
530x faster
at updating dependencies
pg. 16


1 in 10
component downloads have known vulnerabilities
pg. 32

373,000
average enterprise downloads of OSS components per year
pg. 33

High Performers are
59% more likely
to be using software composition analysis (SCA) tools pg. 23

High Performers are
28% more likely
to enforce OSS governance in Continuous Integration (CI) pg. 23

Introduction

Digital innovation is the ultimate source of competitiveness and value creation for almost every type of business. As a result, three things are increasingly common among corporate software engineering teams and the 20 million software developers that work for them:

- They seek faster innovation
- They seek improved security
- They utilize a massive volume of open source libraries

The universal desire for faster innovation demands efficient reuse of code, which in turn has led to a growing dependence on open source and third-party software libraries. These artifacts serve as reusable building blocks, which are fed into public repositories (npm, Maven Central, PyPI, NuGet Gallery, RubyGems, etc.) where they are freely borrowed by millions of developers in the pursuit of faster innovation. This is the definition of the modern software supply chain.

Now in its sixth year, Sonatype's State of the Software Supply Chain Report continues to examine compelling and measurable practices of secure open source software development and delivery. For the second year in a row, we've collaborated with research partners Gene Kim from IT Revolution and Dr. Stephen Magill, CEO at MuseDev, to examine how high performing enterprise software development teams successfully balance their performance and risk management practices while assembling applications with open source components.

The 2020 State of the Software Supply Chain Report blends a broad set of public and proprietary data, along with survey results from over 5,600 professional developers to reveal important findings, including:

- 430% growth in next generation cyber attacks actively targeting open source software projects (Chapter 1)
- 1.5 trillion open source component and container download requests in 2020 (Chapter 2)
- 530x faster mean time to update dependencies and 2.8x more commits for exemplary open source projects (Chapter 3)
- 26x faster detection and remediation of open source vulnerabilities for high performance enterprise development teams (Chapter 4)
- 11% of OSS components used in applications have known vulnerabilities (Chapter 5)

Once again, the report summarizes the latest government and industry initiatives designed to protect software supply chains and strengthen the foundations of open source.

Together with our partners, we are proud to share this research. We hope that you find it valuable.

CHAPTER 1

Open Season on Open Source

In 2020, developers around the world will request more than 1.5 trillion open source software components and containers for one reason: it accelerates the pace of innovation.

In the past 12 months, the number of next generation cyber attacks aimed at actively infiltrating open source increased 430%. The attacks are a uniquely efficient way for adversaries to gain leverage and scale by exploiting software supply chains.

Simply stated, members of the world's open source community are facing a novel and rapidly expanding threat that has nothing to do with passive adversaries exploiting known vulnerabilities in the wild — and everything to do with aggressive attackers implanting malware directly into open source projects. **To that end, it is important to distinguish between legacy supply chain exploits, and next-generation supply chain attacks.**

Software Supply Chain Attacks: Past and Future

Legacy software supply chain "exploits," such as the now famous Struts incident at Equifax, prey on publicly disclosed open source vulnerabilities that are left unpatched in the wild. Conversely, next generation software supply chain "attacks" are far more sinister because bad actors are no longer waiting for public vulnerability disclosures. Instead, they are taking the initiative and actively injecting malicious code into open source projects that feed the global supply chain. **By shifting their focus "upstream," bad actors can infect a single component, which will then be distributed "downstream" using legitimate software workflows and update mechanisms.** Two high profile examples of these modern upstream attacks are event-stream,[1] which targeted the Copay cryptocurrency wallet in November 2018, and the recent Octopus Scanner Malware targeting the NetBeans open source IDE in May 2020.[2]

According to security researchers at the University of Bonn, SAP Labs France, and Fraunhofer FKIE, "From an attacker's point of view, [large scale, public internet-based] package repositories represent a reliable and scalable malware distribution channel. Thus far, Node.js (npm) and Python (PyPI) repositories have been the primary targets of malicious packages, supposedly due to the fact that malicious code can be easily triggered during package installation."[3]

Next-generation software supply chain attacks are possible for three reasons:

1. **Open source projects rely on contributions from thousands of volunteer developers,** and discriminating between community members with good or malicious intent is difficult, if not impossible.
2. **Open source projects themselves typically incorporate hundreds — if not thousands — of dependencies** from other open source projects, which may contain known vulnerabilities. While some open source projects demonstrate exemplary hygiene as measured by mean time to remediate (MTTR) and mean time to update (MTTU), many others do not (see Chapter 3). The sheer volume of open source in use and the massive number of dependencies makes it difficult to quickly evaluate the quality and security of every new version of a dependency.
3. **The ethos of open source is built on "shared trust"** between a global community of individuals, which creates a fertile environment whereby bad actors can prey upon good people with surprising ease.

FIGURE 1A

Combined Reach of 100 Influential Maintainers

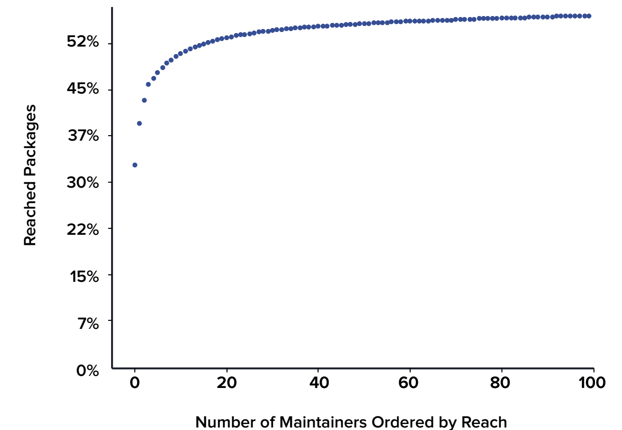
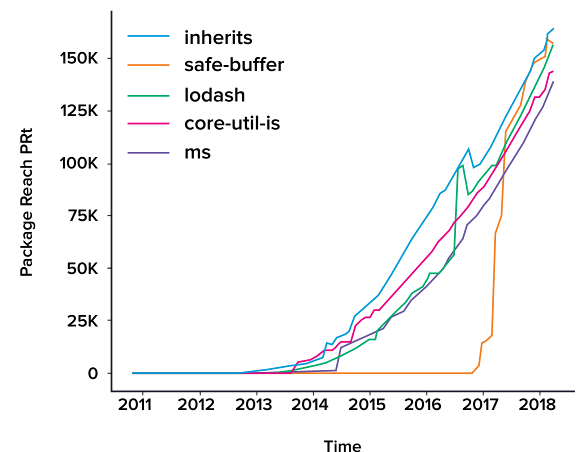


FIGURE 1B

Evolution of Package Reach for the Top 5 npm Packages



SOURCE 1A, 1B: Markus Zimmermann and Cristian-Alexandru Staicu, TU Darmstadt; Cam Tenny, r2c; Michael Pradel, TU Darmstadt

In 2019 Darmstadt University researchers found that a typical npm package contained an abnormally large number of dependencies — loading an average of 79 third-party packages from 39 different maintainers. **The research team also found that 391 highly influential project contributors affect more than 10,000 components through their complex web of dependencies.**[4]

If an adversary were to successfully identify entry points into projects supported by one of these 391 maintainers, they could dramatically widen the aperture and impact of their open source supply chain attacks. For example, the Darmstadt team said that adversaries **gaining access to 20 popular npm maintainer accounts could deploy malicious code impacting more than half of the npm ecosystem** (FIGURE 1A).

Furthermore, the researchers found that the package reach of the top 5 packages was between 134,774 and 166,086 other packages, making them an extremely attractive target for attackers (FIGURE 1B).[5]

Exacerbating the risks even further, the Linux Foundation's Core Infrastructure Initiative found that **of the top 10 most-used software packages, seven were hosted under individual developer accounts;** the researchers then questioned "what happens if one of these accounts is hacked? Would you, farther down the software supply chain, even know?"[6]

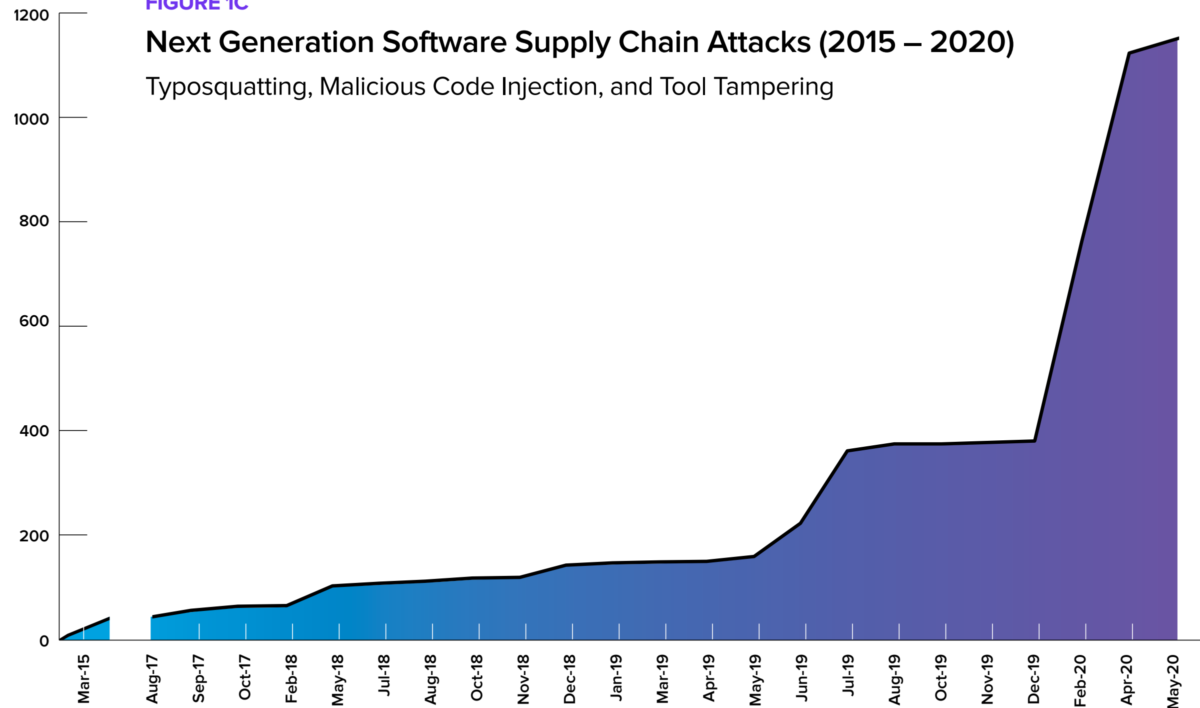
Rise of Next-Gen Software Supply Chain Attacks (2015-2020)

Next generation cyber attacks actively targeting open source software projects have increased 430% since we published this report last year. From **February 2015 to June 2019, 216 such attacks were recorded. Then from July 2019 to May 2020 an additional 929 attacks were documented** (FIGURE 1C).

FIGURE 1C

Next Generation Software Supply Chain Attacks (2015 – 2020)

Typosquatting, Malicious Code Injection, and Tool Tampering



The most common type of attack is Typosquatting, an indirect attack vector that preys on developers making otherwise innocent typos when searching for popular components. If a developer accidentally types "lodahs" when their intention is to source "lodash," they might accidentally install a malicious component of a similar name (see Lodahs, November 2019).

Another common attack is Malicious Code Injection, which is carried out through a variety of means, including stealing credentials from a project maintainer (see **rest-client,** August 2019), releasing new versions of a project to a public repository (see **bootstrap-sass**, April 2019) contributing pull requests to a project that include malicious code (see **event-stream**, November 2018), or tampering with open source developer tools that inject malicious code into downstream applications (see **Octopus Scanner**, May 2020).

When malicious code is deliberately and secretly injected upstream into open source projects, it is highly likely that no one knows the malware is there, except for the person that planted it. This approach allows adversaries to surreptitiously set traps upstream, and then carry out attacks downstream once the vulnerable code has moved through the supply chain and into the wild.

An abbreviated list of next-generation software supply chain attacks occurring from January 2019 – May 2020:

JANUARY 2019

▸ pytz3-dev

The author of this PyPI package seems to have copied the 'pytz' package code and then added malicious code that finds the Discord application's data folder on Windows machines and attempts to extract the Discord token from a SQLite database file. The package has been downloaded about 47 times per month.[7]

▸ smartsearchwp

Published in January 2019 and then yanked from the npm repository in June 2020, included malicious code that provided a backdoor to support data exfiltration.[8]

MARCH 2019

▸ simple-captcha2 0.2.3 and datgrid 1.0.6

As distributed on RubyGems.org, included a code-execution backdoor inserted by a third party.[9]

APRIL 2019

▸ bootstrap-sass

Someone removed a version of the library, bootstrap-sass v3.2.0.2 and immediately released a new version, moments later (v3.2.0.3) with malicious code injected into it.[10]

JUNE 2019

▸ 23 RubyGems packages

Including chrome_taker, color_hacker, aloha_analyser, get-text, ruby_nmap, get-texts, colourize, and btc-ruby were pulled from the public repository because they contained code for crypto mining or cookie/password stealing.[11]

▸ electron-native-notify (version 1.1.6)

An npm package contained code designed to steal cryptocurrency wallet seeds and other login instruction details specific to cryptocurrency apps. Tipped off by npm researchers, makers of the Agama cryptocurrency wallets shifted $13 million worth of currency before adversaries could steal it.

JULY 2019

▸ libpeshnx

A PyPI package discovered to include a backdoor vulnerability. While the package had been reported as containing a known vulnerability, it had not been removed from the Python package repository.

▸ 230 RubyGems

Pulled for typosquatting or impersonating popular open source packages.

AUGUST 2019

▸ 109 RubyGems

Yanked from the repository for typosquatting.[12]

▸ rest-client, coming-soon, and cron_parser

Adversaries compromised the account of a rest-client maintainer to install crypto miners in versions 1.6.10 to 1.6.13. Affected versions were downloaded about 1000 times. Similar vulnerabilities were found in Gem packages: coming-soon and cron_parser.[13]

▸ bb-builder

Removed from the npm repository after it was discovered that it stole login information from the computers it was installed on and sent sensitive information to a remote server.[14]

OCTOBER 2019

▸ basic_authable

Three versions of this Gems package released in 2017 were yanked from the Gems repository due to their malicious nature.

NOVEMBER 2019

▸ sj-tw-test-security

All versions of the component contain malicious backdoor code that downloads and runs a script that opens a reverse shell in the system, allowing a remote attacker to compromise the affected system.[15]

▸ lodahs, web3b, and web3-eht

Taking advantage of a typosquatting exploit for lodash npm packages, all versions of the "lodahs" package contained malware designed to find and exfiltrate cryptocurrency wallets. web3b and web3-eht were removed for the same exploit pattern.

DECEMBER 2019

▸ python3-dateutil and jellyfish

Two trojanized PyPI packages were caught stealing SSH and GPG keys from the projects of infected developers. The two libraries imitated the popular "dateutil" and "jellyfish" (the first L is an I).[16]

JANUARY 2020

▸ 1337qq-js

The malicious npm package exfiltrates sensitive information such as hard-coded passwords or API access tokens through install scripts and targeting UNIX systems only.

FEBRUARY 2020

▸ 381 RubyGems

Packages were yanked from the public repository as a result of typosquatting concerns.[17]

APRIL 2020

▸ 362 RubyGems

Were removed from the public repository for typosquatting and crypto mining malware. They include "atlas-client" (downloaded 2,100 times by developers).[18]

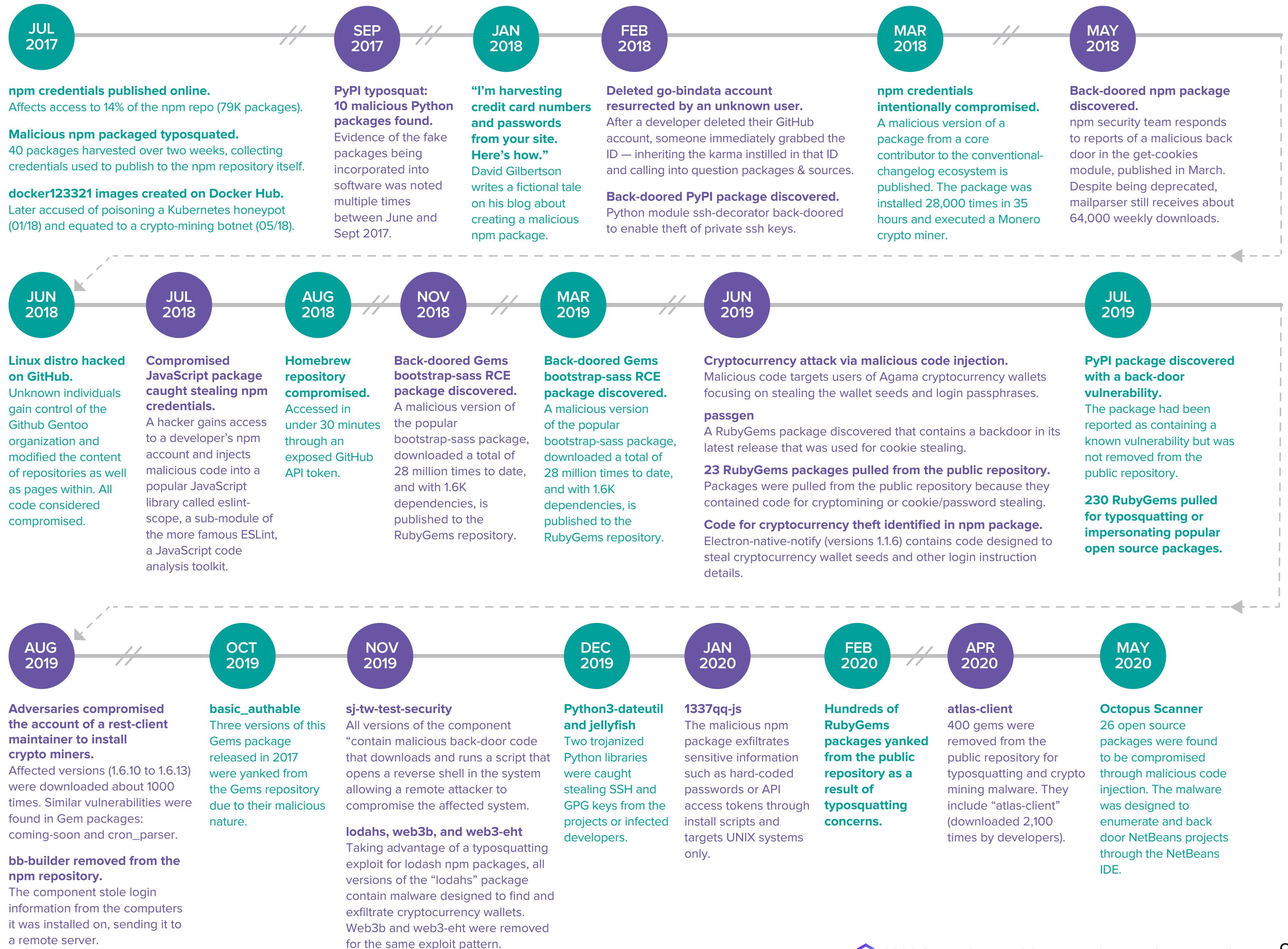
MAY 2020

▸ Octopus Scanner

26 open source packages were found to be compromised through malicious code injection. The malware was designed to enumerate and backdoor projects through the NetBeans IDE.

FIGURE 1D

Software Supply Chain Attacks, July 2017 to July 2020



Speed Remains Critical When Responding to Legacy Software Supply Chain Attacks

While bad actors are increasingly shifting their attention upstream, it is critical to understand and manage the software supply chain threats that remain prominent downstream. Specifically, organizations must establish a "rapid upgrade posture" so they can respond quickly to new zero-day disclosures by finding and fixing vulnerable open source dependencies in production applications.

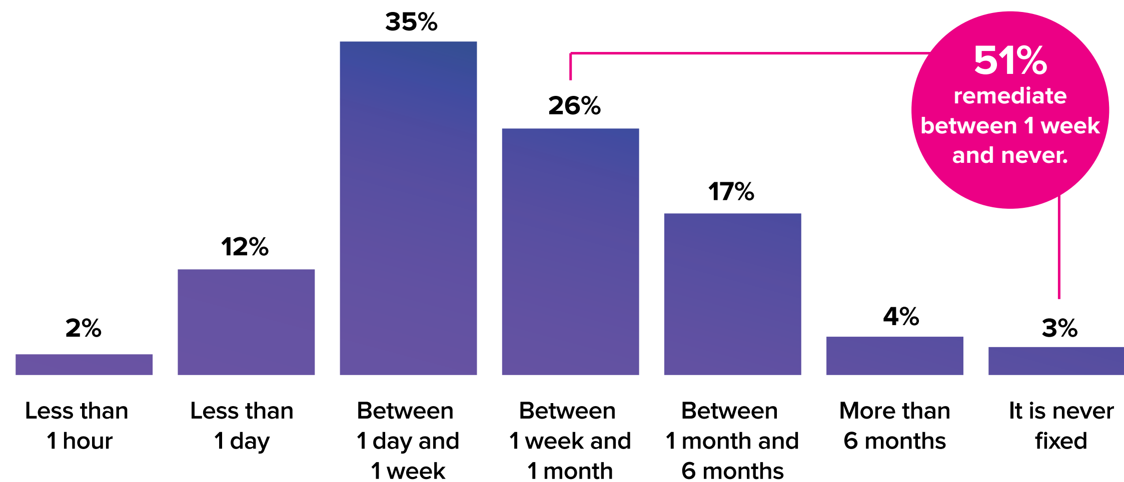
Perhaps the best example of why this hygiene is so critical is the Equifax breach that began in March 2017. Following public disclosure from the Apache Foundation pertaining to a severe vulnerability in the popular Struts2 Framework, adversaries sprang into action and began exploiting the newly-known defect within 72 hours, well before many commercial IT teams (including Equifax) could respond and update their frameworks. This remarkably small window to respond led to numerous high-profile breaches, including Canada Statistics, Canada Revenue, the GMO Payment Gateway, Okinawa Power, Japan Post, India Post, and India's AADHAAR digital identification system.

A similar exploit timeline played out with SaltStack this year. Vulnerabilities discovered in the open source application were announced on April 29th — along with safer, fixed versions. **Within three days, 26 organizations that had not updated SaltStack lost control of their application to adversaries** (FIGURE 1F).[19]

The window of exploitability — once vulnerabilities are disclosed — is critical for enterprises to understand. Our 2020 survey of 679 development professionals revealed that **only 17% of organizations become aware of new open source vulnerabilities within a day of public disclosure.** Thirty five percent (35%) find out within one to seven days, and the remaining 48% become aware of new vulnerabilities after a week's time.

FIGURE 1E

Time to Remediate Known OSS Vulnerabilities After Detection



Once an organization becomes aware of a new open source vulnerability, mitigating actions can begin. The same survey revealed that **51% of participants required more than a week to respond** (FIGURE 1E). This means that adversaries averaging three days to exploit newly disclosed vulnerabilities hold an advantage over half their enterprise targets.

With a better understanding of adversaries attack vectors on software supply chains, our next chapter will shed light on the industry's growing supply of and insatiable demand for open source components. ■

FIGURE 1F

Adversaries exploited open source vulnerabilities within 3 days of disclosure.

26 organizations breached in May 2020.

Open Source: Supply and Demand

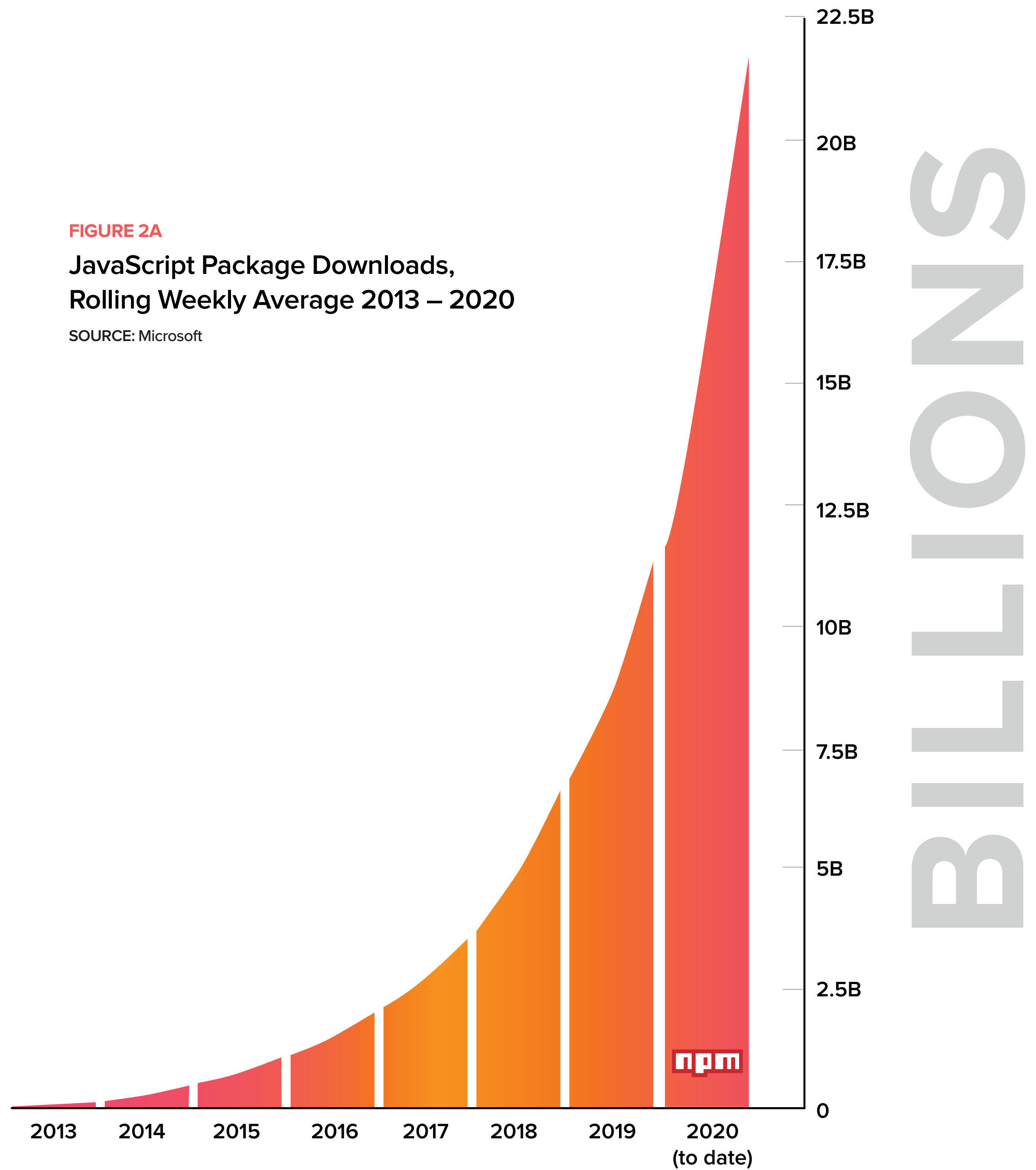
JavaScript

One trillion JavaScript packages will be downloaded in 2020 based on monthly download volumes today. With over 86 billion package downloads in May 2020, the average monthly download traffic for npm packages has grown more than 100% year over year.[20] For the 10.7 million JavaScript developers around the world, this means each will download an average of 93,457 packages in 2020.[21] To keep pace with demand for component-based development, JavaScript community members introduced over 500,000 new component releases in the past year. There are now 1.3 million npm packages available to developers — up 63% from last year.

FIGURE 2A

JavaScript Package Downloads, Rolling Weekly Average 2013 – 2020

SOURCE: Microsoft



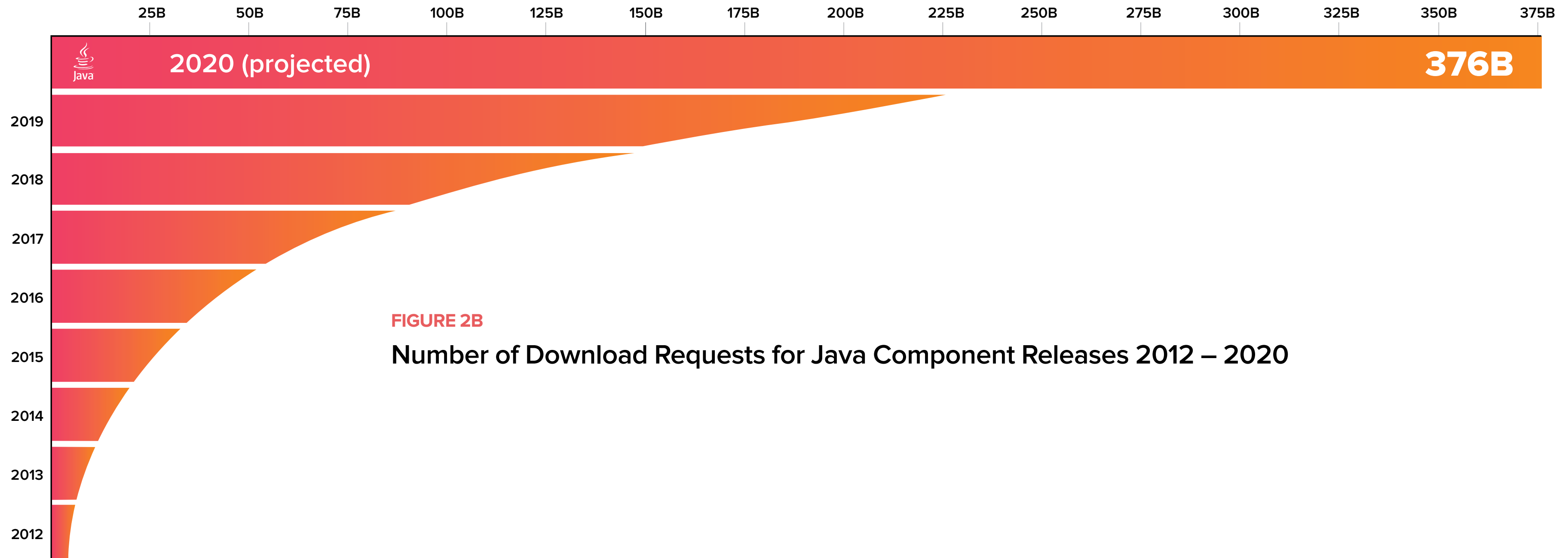


FIGURE 2B

Number of Download Requests for Java Component Releases 2012 – 2020

Java

There are an estimated 7.6 million Java developers worldwide.[22] In 2019, those developers triggered 226 billion open source software component download requests from Maven Central. Download request traffic was up 55% year over year, with the average developer requesting 29,736 component releases annually. With over 31 billion download requests in June, **annual download requests for 2020 are on pace to top 376 billion.**

For Java developers, the supply of Maven packages increased from 3.7 million (June 2019) to over 5 million (June 2020). There are 337,000 Java open source projects that make their component releases available on Maven Central.

.NET

.NET developers were also eager to consume open source software packages over the past year. Developers who downloaded an annualized 16.2 billion NuGet packages in 2019 increased their appetite 177% to reach **44.8 billion annualized downloads in 2020.**[23] The supply of components increased by 700,000 package releases in the past year — now totaling 2.3 million.[24] Over 200,000 open source projects now make their packages available on the NuGet Gallery.

DockerHub

According to stats available from the Docker Index, pulls of container images topped 8 billion for the month of January.[25] This means annualized image **pulls from the repository should top 96 billion this year.**[26] To keep pace with demand, suppliers pushed 2.2 million new images to DockerHub over the past year — up 55% since our last report.

Now that we have examined supply and demand levels, our next chapter aims to shed light on attributes to look for when selecting the best open source projects to rely upon. ■

Identifying Exemplary Open Source Suppliers

Researching the Best Performing OSS Projects

To better understand the health and habits of the open source component ecosystem, we researched thousands of Java components housed in The Central Repository ("Maven Central") to help answer the following questions:

- Do differences exist in how effectively OSS projects update their dependencies and fix vulnerabilities? Are there exemplary components that do this better than others?
- Are exemplary components more widely-used than "non-exemplary" components?
- What factors correlate with exemplary components?

Components included in the research had to meet the following criteria:

- Published to the Central Repository
- Released at least two versions
- Represented in the open source supply chain (e.g., is itself a dependency, or has a dependency)
- Followed the Maven standard for versioning (e.g., correct use of numeric version strings, components separated by dots)
- Has dependencies satisfying all of the above
- Has updated a dependency at least once

With a final data set of 24,053 components, we examined a number of attributes to identify relative hygiene across open source projects including, **responsiveness to reported security vulnerabilities, number of dependencies, number of stale dependencies, frequency of releases, popularity, number of commits per month, developer team size, presence of continuous integration, and support type** (foundation, commercial, or other).

FIGURE 3A

SMALL EXEMPLAR (329)	LARGE EXEMPLAR (560)	LAGGARDS (3,040)	FEATURES FIRST (581)	CAUTIOUS (3,691)
Small development teams (1.6 devs), exemplary MTTU, likely to be commercially supported and 4.3x more popular.	Large development teams (8.3 devs), exemplary MTTU, likely to be foundation supported, 2.5x more popular.	Poor MTTU, high stale dependency count, more likely to be commercially supported.	Frequent releases, but poor TTU. Still reasonably popular.	Good TTU, but seldom completely up to date.

Finding Different Behavioral Groups

As a result of our analysis, we identified five clusters representing 8,201 open source projects (FIGURE 3A).

Exemplars

We defined Exemplars to be those teams in the fastest 20% by Median Time to Update (MTTU) dependencies, and in the best (lowest) 20% by stale dependency count. Exemplars demonstrate statistically significant differences as compared to the rest of the data set in the following attributes:

- 530x faster MTTU
- 2.8x more commits
- 1.5x more frequent releases
- 1.4x larger development teams
- 2.9x fewer dependencies
- 2.5x more popular
- 173x less likely to have at least one dependency out of date

LARGE EXEMPLARS

Large exemplary teams (top 50% by size, with an average of 8.3 developers committing code on at least a monthly basis), commit code frequently, release frequently, and do an excellent job of managing their dependencies. For example, we can see that **large exemplary teams are 608x faster at updating their dependencies and they release 2.9x more frequently than non-exemplar clusters.** We can see that 21% of these projects are associated with an open source foundation — a higher representation than any other cluster group.

Large exemplars are 608x faster at updating their dependencies and they release 2.9x more frequently than non-exemplar clusters.

SMALL EXEMPLARS

The smallest 50% of exemplary teams by number of developers have an average of less than two developers, but still manage to run popular, widely

used, and high quality projects. **However small in team size, they still update dependencies 475x faster than the rest of the population** and are 4.3x more popular by download count compared to the Laggards and Cautious teams. Small projects were also 7x more likely to be commercially supported versus open source foundation supported.

Laggards

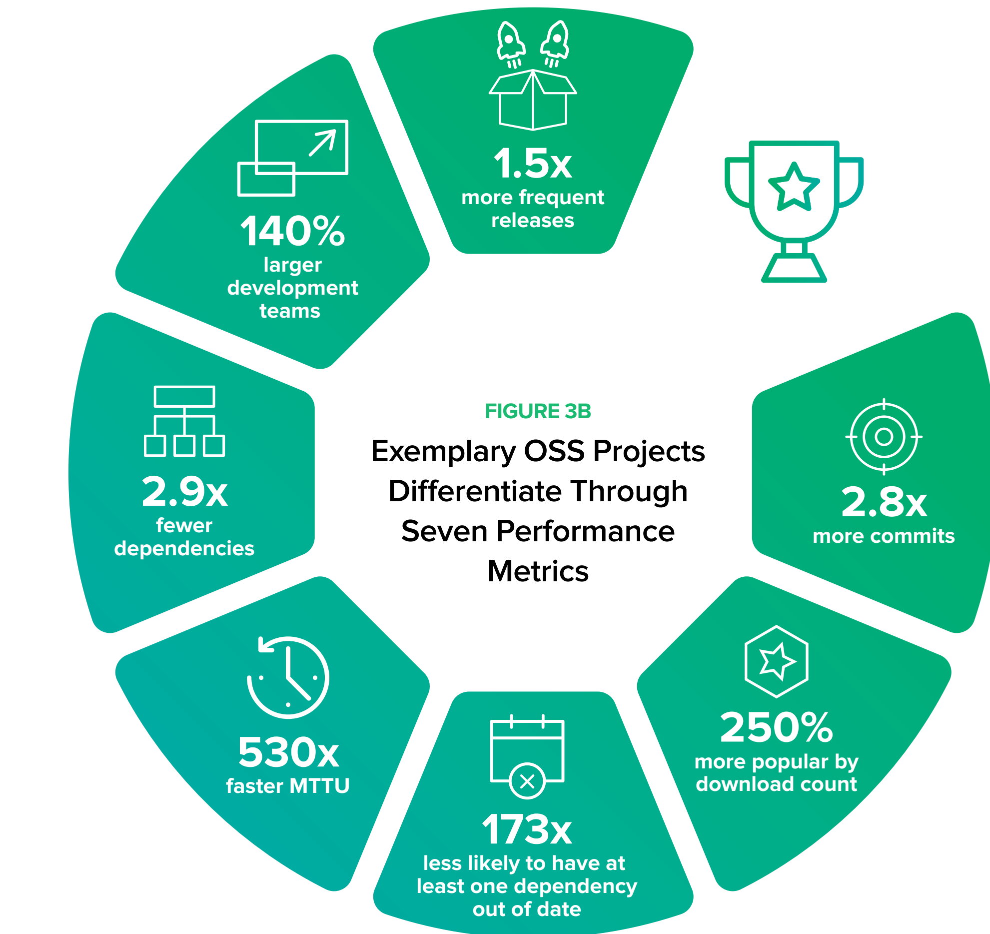
The teams in the bottom 20% in MTTU and stale dependencies are the furthest behind in terms of update hygiene. **These teams release infrequently (around twice each year) and take on average almost two years to adopt updates to dependencies**. The average period at least one of their dependencies is out of date is 203 days. They are 1.7x less popular (not downloaded as often as other projects on average). However, there are 288 projects in this group that are among the top 10% most downloaded projects from The Central Repository. This group represented 37% of our dataset.

FEATURES FIRST LAGGARDS

These teams release frequently (top 50%) but otherwise fall into the Laggard category (bottom 20% MTTU and stale dependencies). They have larger than average (2.4x larger) development teams than other Laggards, but do not prioritize upgrading dependencies. They release a new version every 29 days on average, but **take an average of 501 days to upgrade dependencies** when new versions are released. As a result, 88% of dependencies are out of date at release time. This was a small group, with 7% of the five cluster population exhibiting this behavior.

Cautious Teams

We checked to see how many teams were in the top 50% with respect to MTTU, but the bottom 20% with respect to stale dependencies.



Cautious teams release new versions about every two months, which is 1.3x more frequently than Exemplar teams, yet they were 11x slower at updating dependencies. By comparison, Cautious teams were 27x faster at updating dependencies than their Features First Laggard peers.

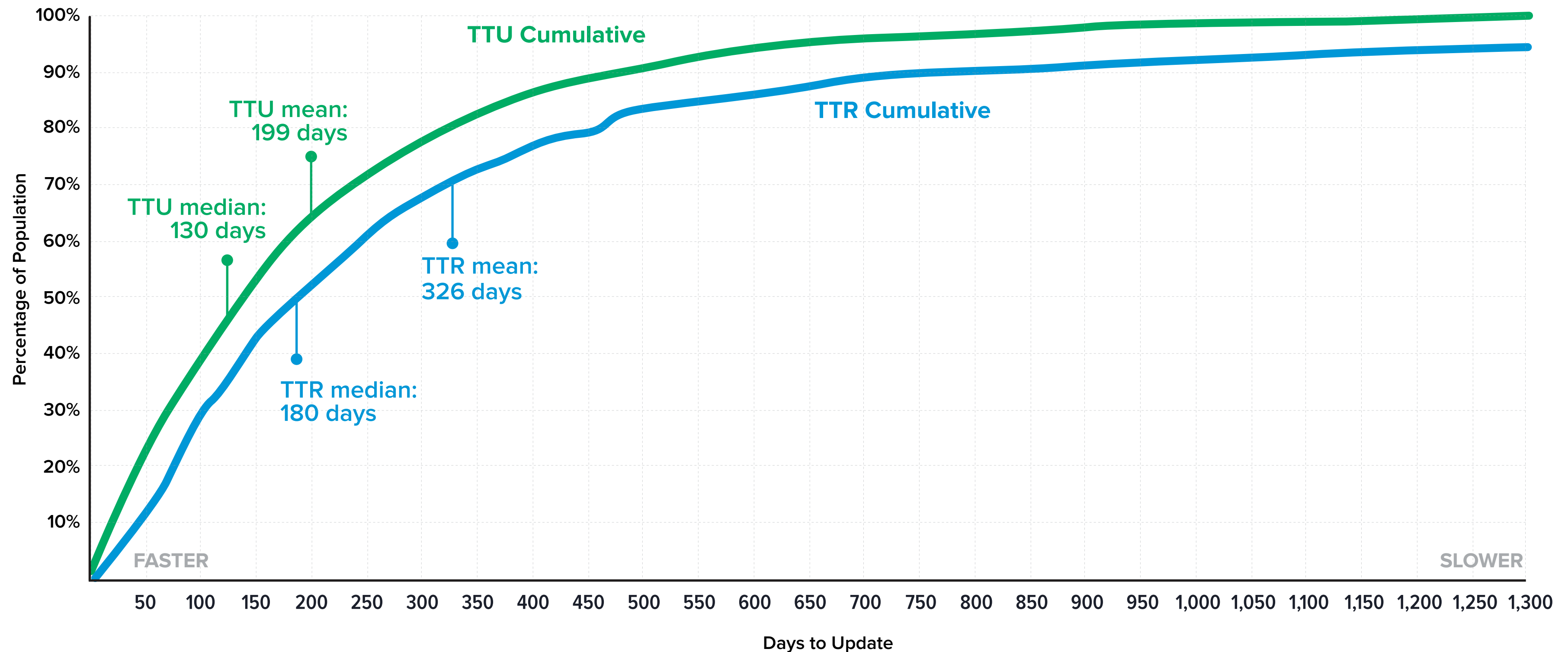
These teams maintain better-than-median update cadence, yet do not immediately adopt new versions of dependencies, choosing instead to wait a few months before moving to a new dependency release. This group represented 45% of our dataset falling into this category.

However small in team size, Small Exemplars still update dependencies 475x faster than the rest of the population.

FIGURE 3C

Time to Remediate (TTR) vs. Time to Update (TTU)

(cumulative percentage)



Projects with Updated Dependencies Are More Secure

The adoption curve for upgrading dependencies and remediating vulnerabilities are similar, as shown in **FIGURE 3C**. When comparing MTTR with MTTU for non-security-relevant updates on a per-component basis, we see a correlation between update behavior for security relevant updates (MTTR) and non-security-relevant updates.

As we discovered in our 2019 report, developers staying up to date on dependencies will generally stay up to date on security updates, because security updates are a subset of general updates. We observed that many teams follow this practice, exhibiting very similar median times to remediate (MTTR) and mean time to update (MTTU) values. Large and small exemplars will generally achieve better security outcomes because of their strong MTTU performance **(SEE FIGURE 3C)**.

To adopt this practice, security managers should encourage component and dependency updating practices by partnering with their development counterparts.

Teams should aim for a minimum of four releases annually and aim to upgrade at least 80% of their dependencies with every release.

Guidance for Open Source Project Owners and Contributors

Given its association with good security practices and outcomes, we recommend a focus on accelerating and maintaining rapid MTTU. In addition to investing development effort on new features, bug fixes, etc., projects should commit similar resources to dependency management. This means that developers maintaining OSS projects who are considering adding a new dependency, and looking for a metric to guide that choice, would be wise to select dependencies with fast MTTU because such components naturally exhibit better security hygiene.

To progress comfortably into the status of Exemplar (top 80% of Exemplars), teams should aim for a minimum of four releases annually and aim to upgrade at least 80% of their dependencies with every release. A higher frequency of dependency updates statistically results in higher quality and more secure code.

Guidance for Enterprise Development Teams

Enterprise development teams working with software supply chains often rely on an unchecked variety of supply from OSS projects where each developer or development team can make their own sourcing and procurement decisions. The effort of managing 3,552 different projects and 11,294 unique releases (see Chapter 5) can introduce significant drag on development and is contrary to an enterprise's need to develop faster as part of any agile, continuous delivery or DevOps practice.

Choosing open source projects should be considered an important strategic decision for enterprise software development organizations. Different components demonstrate healthy or poor performance that impacts the overall quality of their releases. Therefore, MTTU should be an important metric when deciding which components to utilize within your software supply chains. Rapid MTTU is associated with lower security risk and is accessible from public sources.

Just as traditional manufacturing supply chains intentionally select parts from approved suppliers and rely upon formalized procurement practices — enterprise development teams should adopt similar criteria for their selection of OSS components. This practice ensures the highest quality parts are selected from the best and fewest suppliers — a practice Deming recommended for decades. Implementing selection criteria and update practices will not only improve code quality, but can accelerate mean time to repair when suppliers discover new defects or vulnerabilities. Chapter 4 will further explore the impact of OSS component selection on overall application quality. ■

Just as traditional manufacturing supply chains intentionally select parts from approved suppliers and rely upon formalized procurement practices — enterprise development teams should adopt similar criteria for their selection of OSS components.

CHAPTER 4

How High Performance Teams Manage Open Source Software Supply Chains

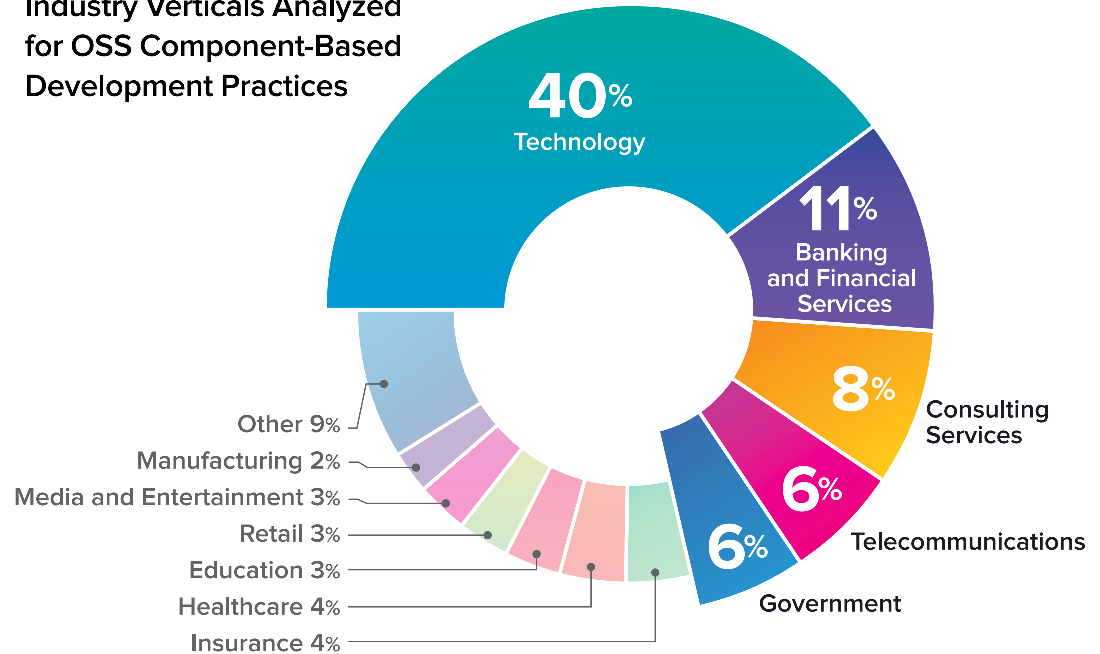
Analyzing the performance and security of open source component-based software development is made easier because, similar to manufacturing supply chains, the inventory is visible.

For this year's report, we expanded our survey of OSS component-based development practices to include 679 engineering professionals employed in commercial roles. We inquired about software delivery outcomes (e.g., deployment frequency, security, engineering productivity, job satisfaction) and practices (e.g., approaches and philosophies to utilizing open source components, organizational design, governance, approval processes, and tooling). The goal was to discover to what extent various practices contribute to success. To assess this, we performed a number of analyses including fitting regression models to the data, clustering, and examining statistically-significant between-group effects.

We believe the results we found can help organizations evaluate their approaches to using open source components and improve the performance and security of their software delivery practices.

FIGURE 4A

Industry Verticals Analyzed for OSS Component-Based Development Practices



Survey of Open Source Management Practices

We created a survey with 41 questions, exploring ten areas of software outcomes (dependent variables), and twenty-four areas of software practices, tooling, organization, policies, etc. (independent variables).

We obtained responses from 679 individuals across a wide variety of industry verticals, including Banking, Retail, Healthcare, and Government (SEE FIGURE 4A). Organizations of all sizes were represented, ranging from 10-developer organizations to companies with more than 5,000 developers. 63% of respondents were individual contributors or team leads, while 37% were managers, VPs, or executives. Participants achieved a 75% completion rate, defined as respondents that answered all of the questions that fed into our statistical data analysis.

Cluster Analysis and Findings

To identify cohorts with similar reported outcomes, and identify high and low performers, we used a cluster analysis.[27] We found four clusters with markedly different levels of performance, with different patterns of practices, and with almost all factors being statistically different. We labeled them as follows:

Analyzing the performance and security of open source component-based software development is made easier because, similar to manufacturing supply chains, the inventory is visible.

- **High Performers**: high productivity, great risk management outcomes (N=151)
- **Low Performers**: low productivity, poor risk management outcomes (N=107)
- **Security First**: low productivity, great risk management outcomes (N=167)
- **Productivity First**: high productivity, poor risk management outcomes (N=103)

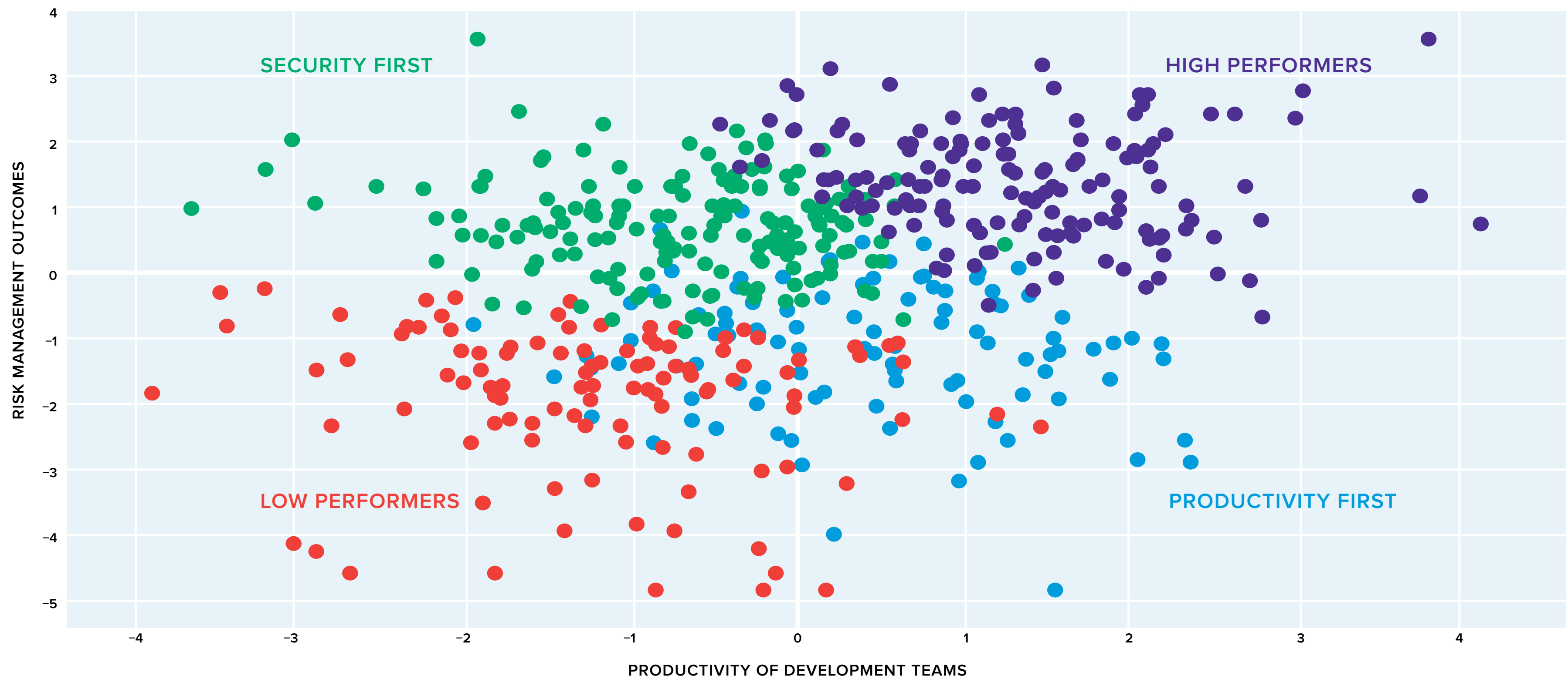
We can quickly see the different characteristics of the four clusters by projecting them onto a quadrant — on one axis are all the productivity-related outcomes combined into a single dimension, and on the other are the risk management outcomes combined into a single dimension (both using principal components analysis).

The resulting graph **(FIGURE 4B)** identifies a High Performers cluster (purple, upper right) who demonstrate superior risk management outcomes while maintaining high levels of productivity. The Low Performers cluster (red, lower left) identifies the opposite pattern: demonstrating substandard risk management outcomes and low levels of productivity. The Security First cluster has high security outcomes, but low productivity, and the Productivity First has high productivity, but poor security outcomes.

It is important to note that the High Performers achieved even higher average productivity levels than the Productivity First cluster. As seen in **FIGURE 4B**, the High Performers are tightly clustered in the upper right quadrant, while the Productivity First group is more distributed across the bottom left- and right-quadrants.

FIGURE 4B

Measuring Risk Management vs. Productivity Outcomes



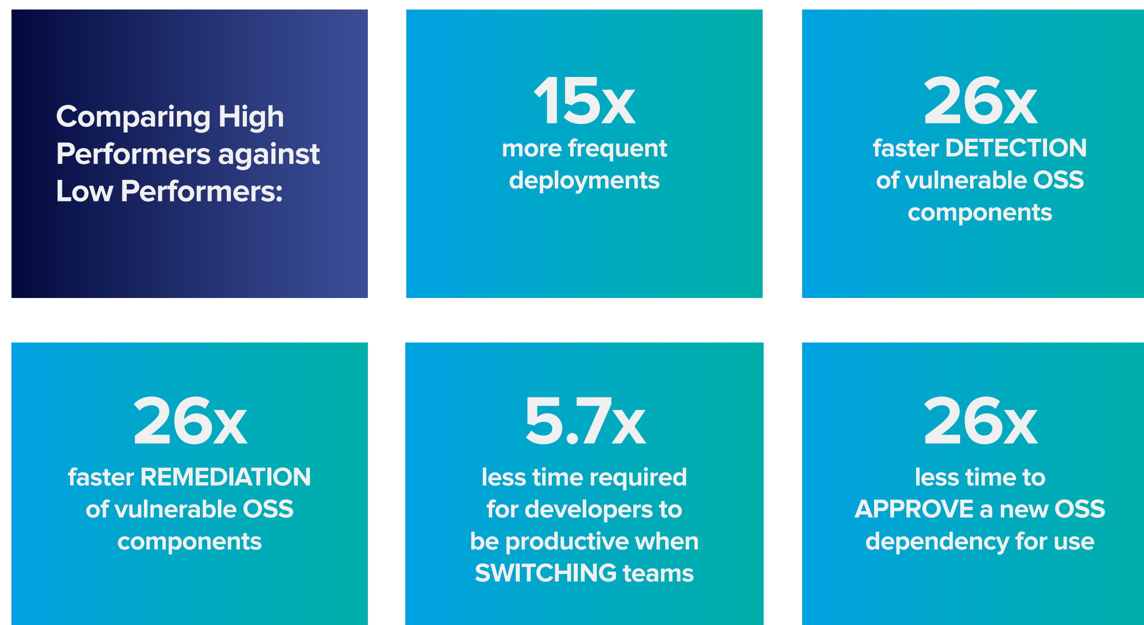
Comparing High Performers vs. Low Performers

The tables on the following pages show how decisively the High Performers outperform the low performers in software delivery and security — **they deploy more frequently, they detect and remediate vulnerable OSS components more quickly, onboard developers onto new teams more quickly, and approve new OSS components for use more quickly.**

Furthermore, High Performers are more confident about the compliance and security of their OSS components, and have fewer problems updating their OSS components. Compared to Low Performers, High Performers are.

- 4.9x less likely to have dependencies break application functionality
- 3.8x more likely to describe updating dependencies as easy (i.e., not painful)
- 33x more likely to be confident that OSS dependencies are secure (i.e., no known vulnerabilities)
- 4.6x more likely to be confident that OSS licenses of dependencies are compliant with internal requirements
- 2.1x more likely to have access to newer OSS component versions where prior defects have been fixed
- 1.5x more likely for employees to recommend their organizations as a great place to work

FIGURE 4C



Comparing High Performers vs. Security First

Many have argued that effective risk management practices are always at the expense of developer productivity, (i.e., "better security slows down development"). We can see these outcomes in the Security-First cluster (green, upper left) that seemed to be achieving good security outcomes in a way that impeded developer productivity. **By comparison, the High Performer cluster shows high productivity and superior risk management outcomes can be achieved simultaneously.**

To better understand these differences, we examined what practices separate the High Performers from this Security-First cluster. It turns out that **High Performers tend to have a governance structure that relies much more heavily on automated tooling.** Compared to the Security-First group, the High Performers were:

- 77% more likely to automate approval, management, and analysis of dependencies
- 59% more likely to be using software composition analysis (SCA) tools
- 28% more likely to enforce governance policies in Continuous Integration (CI)
- 56% more likely to have centrally-managed CI infrastructure
- 51% more likely to maintain a centralized record of all deployed artifacts, supporting the collection of a Software Bill of Materials (SBOM) for each application
- 96% more likely to be able to centrally scan all deployed artifacts for security and license compliance.

Variables Most Impacting Performance and Risk Management

In this section, we state all of the hypotheses we had when we designed the survey, and state which practices (independent variables) we believed would affect the performance outcomes (dependent variables) — we also define how we measured them.

To better understand the connection between practices and outcomes, and potentially understand how one can improve performance, we fit a linear model to the data.[28] We measured and, where appropriate, report r^2 values, which describe the proportion of variance in each outcome explained by the model and describe the top practices, based on their contribution to increases in the outcome being analyzed.[29] [30]

(All independent and dependent variables are listed and described in Appendix B.)

Influencing Risk Management Outcomes

Across all the risk management outcomes, the most consistent factors associated with positive risk management outcomes were:

- Having a clear process for adding and removing OSS dependencies
- Remediating known OSS vulnerabilities as a regular part of development
- Updating OSS dependencies regularly
- Using SCA tooling and incorporating this tooling into CI

INFLUENCING RISK MANAGEMENT OUTCOMES:
Mean Time to Detect Vulnerabilities (MTTD)

HYPOTHESIS	RESULT (R2 = 0.37)	DISCUSSION
Practices associated with fast MTTD would involve monitoring and tooling (given the high frequency of new vulnerabilities and large number of dependencies on OSS components) that would be integrated into CI processes.	CONFIRMED. Listed below are the top factors associated with fast MTTD. ▸ Scheduling updating open source dependencies as part of our daily work ▸ Remediation of security issues is addressed as a regular part of development work (i.e., security issues treated as normal defects). ▸ Open source component governance (e.g., security, licensing) is enforced through CI infrastructure.	▸ One of the unexpected and interesting factors that appeared in the survey results was the degree to which OSS is supported within the organization, which we called **"OSS Enlightenment."** We speculate that being involved in the OSS community causes engineers to be more aware of important vulnerability disclosures (i.e., a developer who is active in the Java community will be more likely to hear about important vulnerabilities, and what actions are being taken to address them.) We measured this by asking the following: » For company-sponsored OSS projects, to what degree are external contributions allowed? » To what degree does your organization require that all internal modifications to open source components be contributed back (i.e., "pushed upstream")? » To what degree does your leadership support contributing back to open source components we use (e.g., engineering time, budget, conferences)

INFLUENCING RISK MANAGEMENT OUTCOMES:
Mean Time to Remediate (MTTR)
Measured as the time taken to mitigate a vulnerability across applications once the team becomes aware of that vulnerability.

HYPOTHESIS	RESULT (R2 = 0.32)	DISCUSSION
Informed by last year's work, where we saw a strong correlation between MTTR and general dependency update practices, we predicted that practices would include scheduling updates regularly, automated testing to detect when updates break functionality, and a security-oriented development culture (e.g., addressing security vulnerabilities as a regular part of development work) that would result in improved remediation times.	CONFIRMED. Listed below are the top factors associated with fast MTTR. ▸ Degree of OSS Enlightenment (see above) ▸ Scheduling updating open source dependencies as part of daily work ▸ Our application deployments (including configurations) are fully automated	▸ We were surprised by OSS Enlightenment appearing as the top factor here, tied with scheduling updating dependencies as a part of our daily work (which was what we predicted would be highest) — see the MTTD section for the definition and further discussion. ▸ Security guidance often stresses the importance of having an automated mechanism to deploy updates or patches into production. That automated deployment appears as an important factor here supports this view.

INFLUENCING RISK MANAGEMENT OUTCOMES:

OSS Security

Measured as the level of confidence that applications are not using open source components with known vulnerabilities.

HYPOTHESIS	RESULT (R^2 = 0.35)	DISCUSSION
That some tooling to do centralized scanning of dependencies and an effective approval process would predict confidence in OSS security.	**CONFIRMED.** Listed below are the top factors associated with high security confidence: ▶ Having a clear process for adding and removing dependencies ▶ When selecting new OSS components, the two following factors are considered important: » Security history (e.g. have there been multiple high-risk CVEs) » Rate of fixes (frequency of security and bug fixes) ▶ Scheduling updating open source dependencies as part of our daily work	▶ We asked a series of questions about what criteria were important when selecting new OSS components, which is about being careful and particular about functionality, integrations, ease of use, security, etc. ▶ The primary contributing factors all have to do with controlling what components are brought into the supply chain. The two next most important factors both had to do with monitoring to enforce those policies: » The output of software composition analysis (SCA) tools is integrated into daily development workflows. » Every deployed application is centrally tracked, including its open source dependencies, and it is known who the application team leader is. This practice is critical to building and maintaining SBOMs for each application.

INFLUENCING RISK MANAGEMENT OUTCOMES:

License Compliance

Measured as the level of confidence that the development team is in compliance with the organization's policies regarding open source licenses.

HYPOTHESIS	RESULT (R^2 = 0.29)	DISCUSSION
Practices associated with effective governance (e.g., processes defined, tools to monitor compliance, responsibilities assigned, etc.) would increase confidence in OSS license compliance.	**CONFIRMED.** Listed below are the top factors associated with increased confidence in OSS license compliance. ▶ Having a clear process for adding and removing dependencies ▶ Consistently following open source approval processes ▶ Prioritizing licensing considerations when selecting new open source components ▶ Scheduling updating open source dependencies as part of our daily work	▶ We found it interesting that all these factors relate to process, not technology. ▶ We were surprised that the degree of centralized governance was not associated with increased performance — this likely indicates that there are many organizational approaches to effectively solve compliance problems.

Influencing Productivity Outcomes

Factors which influence software delivery productivity are notoriously elusive, although the State of DevOps Report has wonderfully illuminated its link to continuous delivery, culture, lean product development, etc. Our intent was to further explore other practices that could improve aspects of productivity, which revealed some surprises.

INFLUENCING PRODUCTIVITY OUTCOMES: Developer Portability

Measured as the time required for developers to reach normal productivity when switching teams.

HYPOTHESIS	RESULT (R2 = 0.15)
More centralized and standardized DevOps automation across teams will allow developers to become more productive more quickly when switching between teams.	VALIDATED. The top two factors in explaining developer flexibility were: ▶ Having a centralized record of applications, their dependencies, and the associated development teams ▶ Having automated deployments

INFLUENCING PRODUCTIVITY OUTCOMES: OSS Component Approval Times

Measured as the time it takes for developers to get a new OSS library approved for use.

HYPOTHESIS	RESULT (R2 = 0.16)	DISCUSSION
Automation of governance workflows and monitoring would be a primary factor in decreasing OSS approval times.	CONFIRMED.* The fastest approval times were measured in the Productivity First group, where 72% reported "no approval necessary." Of those who had an approval process, the median approval time was "less than 1 day." *While approvals were fast, their process lacked effectiveness, as demonstrated by the cluster's poor risk management outcomes (SEE FIGURE 4B) **By comparison, the High Performers cluster had the second-fastest approval times overall, with a median approval time of "between 1 day and 1 week."** This demonstrates that you can have great security outcomes using automated governance while maintaining high productivity. When we exclude all "no OSS approval necessary" respondents, the top factors associated with shorter approval times are "OSS Enlightenment", prioritizing commercial or foundation support for dependencies, and centralizing scanning for OSS dependencies	▶ It's surprising to see OSS Enlightenment here, but we suspect it's because having familiarity with the open source community leads to faster research and decision making. ▶ Prioritizing the identification of commercial or foundation support for dependencies is associated with slower approval times, indicating that this takes time to assess and research. ▶ Having automated, centralized scanning of OSS dependencies accelerates approval times, as well as detection and remediation responses enabled through SBOMs.

INFLUENCING PRODUCTIVITY OUTCOMES: Internal Forks

Measured as how common it is for internally modified versions of open source projects to be maintained.

HYPOTHESIS	RESULT (R2 = 0.16)	DISCUSSION
Organizations that take a more active role in open source development will maintain *fewer* internal forks of open source projects.	NOT SUPPORTED.	▶ We found that High Performers were *more likely* to maintain internal forks of open source projects. Upon reflection, we believe this is because internal versions are required to make changes and develop new features, even when these are being regularly contributed back. In a future survey, we will ask about long-lived internal forks that diverge from the original repository in order to better capture the distinction between forking to contribute back (generally good) and forking to avoid keeping up-to-date (generally bad).

Influencing Job Satisfaction

This year's survey measured job satisfaction by five questions about various aspects of work including organizational support, level of fit between skills and tasks, and ability to complete work.

INFLUENCING JOB SATISFACTION: Work Attitudes and Motivation. Measured various aspects of work including organizational support, level of fit between skills and tasks, and ability to complete work.		
HYPOTHESIS	**RESULT (R2 = 0.27)**	**DISCUSSION**
High Performers would not only have better security and higher productivity, but also higher job satisfaction; we didn't hypothesize about any specific factors, but we were curious about what factors were associated with high job satisfaction.	**CONFIRMED.**	▶ We found a surprisingly high correlation approaching the levels we saw with security-related outcomes. The top factors were: » How well an open source risk management initiative was resourced and supported » When test suites were used — and tests passed — there was higher confidence that the application would operate as intended in production. » Where application deployments (including configurations) were fully automated » Where agile or DevOps development practices were in place » When OSS Enlightenment (defined above) was present

Interestingly, **the most predictive question of job satisfaction was "How is your current open source risk management initiative resourced and supported?"** This was the most detailed question we asked regarding general organizational support and included sub-questions about executive support, budget, tooling, and documentation. We suspect that this relationship is highlighting a connection between level of employee support and job satisfaction rather than an effect specific to support of open source risk management initiatives. In future surveys we will ask more general "organizational support" questions to evaluate this hypothesis.

The usage of Agile or DevOps practices, as well as automated deployments and their impact on job satisfaction are very similar to the early results from the State of DevOps Research (cite: Dr. Nicole Forsgren, Jez Humble, Gene Kim, 2015 Puppet Labs State of DevOps Report.[31]

Guidance for Enterprise Development Teams

Our research shows that faster innovation and better risk management are not mutually exclusive. Indeed, High Performance engineering teams are accelerating velocity while simultaneously reducing security and licensing risks.

Our investigation into measures of high performance component-based software development and delivery helped us confirm four overarching, compelling and predictable criteria: **time to update dependencies, deployment frequency, time required for developers to be productive when switching teams, and time to detect and remediate defective components.** Teams striving for productivity and risk management outcomes that improve management of their software supply chains and delivery practices should track performance of these criteria.

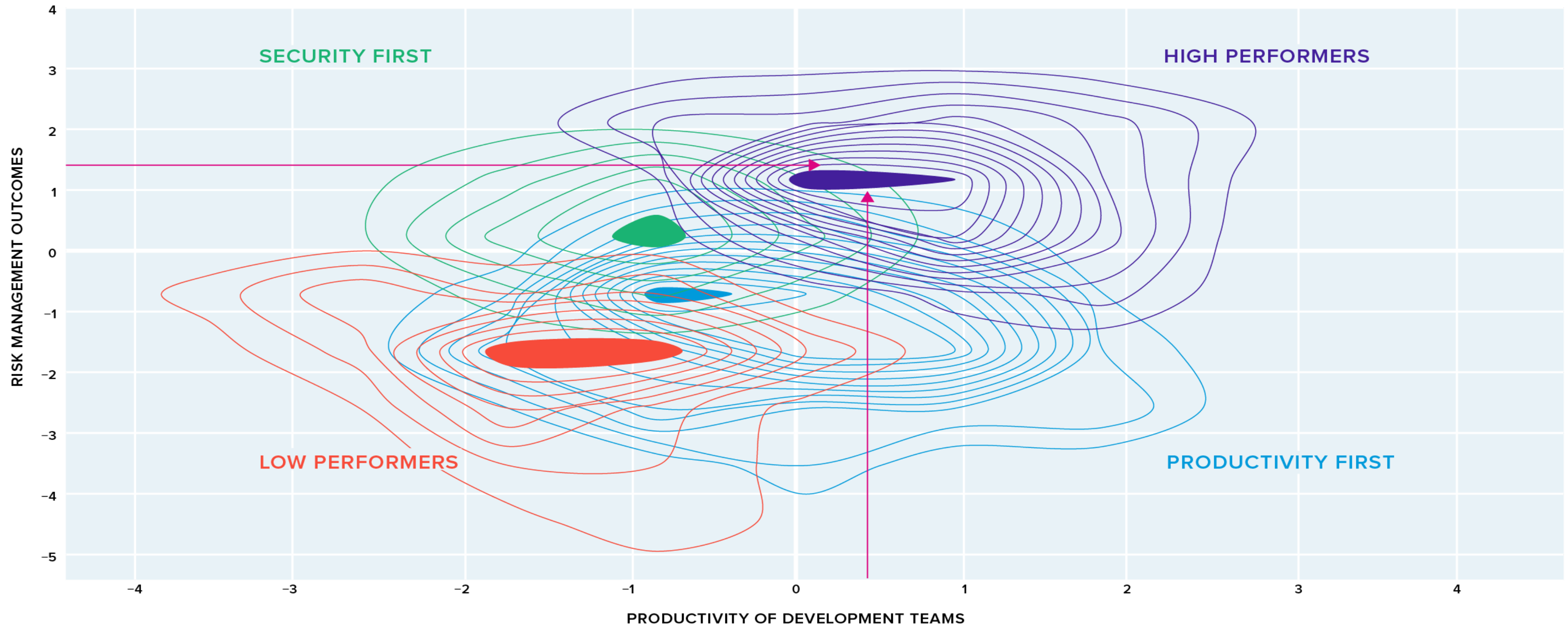
High Performer results are achieved not by implementing a single tool or practice, but through a combination of culture, development practices, policy enforcement, automation, and integrations applied across the development lifecycle. Furthermore, High Performers are not only rewarded with increased productivity and better security (SEE FIGURE 4D), but their employees demonstrate high levels of job satisfaction.

Security First teams desiring to transform themselves into High Performers would benefit from automating their approval, management and analysis of open source components. They should also consider integrating developer friendly SCA tools into their CI process so they can automatically scan build artifacts, easily identify open source security and licensing risk, and benefit from a SBOM for all applications.

FIGURE 4D

Stronger Risk Management and Productivity Outcomes for High Performers

(Comparison of Cluster Centroids)



Productivity First teams wanting to shift up into the High Performer quadrant should prioritize partnering with governance counterparts to integrate automated security scanning into their CI process so they can easily add and remove OSS dependencies and regularly remediate known OSS vulnerabilities.

Patterns Across OSS Component Updates: Easy, Difficult, and Planned

Over the years, we've become increasingly convinced that while updating dependencies is very important for functionality and security, there is a huge economic cost to staying up-to-date. Ideally, dependencies should be updated, simply, safely and painlessly, and as part of the routine development process. But reality shows that this ideal is rarely met.

An astonishing story of how far an organization can stray from ideal update practices comes from Eileen M. Uchitelle, Staff Engineer at GitHub, who described how it took seven years to successfully migrate GitHub from a forked version of Rails 2 to Rails 5.[32] **Even with new tools available to developers that automatically create pull requests with updated dependencies, changes in APIs and potential breakage can still hold back many developers from updating.** We suspect this change-induced break-age is a primary driver of poor updating practices.

Taking a deeper dive into the vast data available to us from The Central Repository, we can better visualize open source project releases and their adoption by enterprise application development teams who migrate from one version to a newer one. **We believe this data shows how OSS component selection can play a major role in allowing for easier and more frequent updates.**

The following graphs show the different stories around OSS update patterns by software development teams. Updates from one version of a library to another are visually depicted by connecting the two versions with an arc. The horizontal axis is an ordered list of library releases, where version numbers increase as you move right.

Consider the graph for the widely used joda-time library **(FIGURE 4E)**, which shows that developers using this OSS component update fairly uniformly between all pairs of versions. This suggests that updates are easy, presenting a seemingly homogenous set of versions to select migrate to and from.

On the opposite extreme, consider the graph for the hibernate-validator library **(FIGURE 4F)**, where there are two sets of communities using it — one favoring version 5 and another preferring version 6. The two communities very rarely intersect. This suggests that **updating to version 6 from version 5 is either too difficult, or the value is not worth the effort.**

Finally, we take a look at the pattern for spring-core **(FIGURE 4G)**, which suggests that **updating is sufficiently difficult that the effort must be planned and some version ranges end up being avoided.**

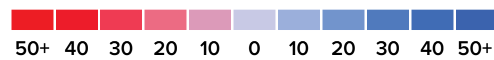
In our future work, we would like to further investigate which dependencies the High Performers and other notable clusters are using and the criteria they use to select them, while measuring the effort and cost required to stay up-to-date. We believe that this could reveal lessons and principles that could help every organization using open source software components.

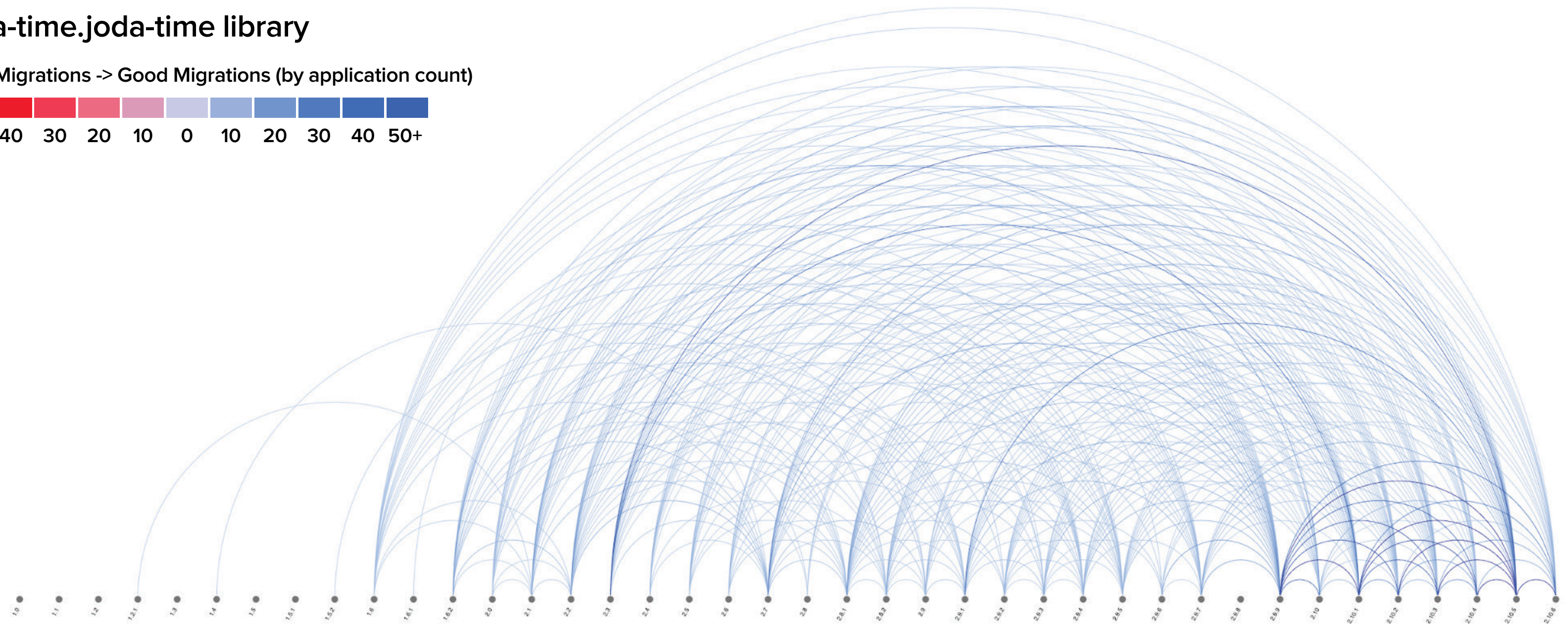
Now that we have explored practices and related outcomes that contribute to successful software supply chain management, let's take a closer look at the volume, quality, and security of open source component consumption in the enterprise. ■

FIGURE 4E

joda-time.joda-time library

Poor Migrations -> Good Migrations (by application count)



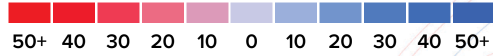


MIGRATION PATTERNS BETWEEN OSS COMPONENT RELEASES

FIGURE 4F

hibernate-validator library

Poor Migrations -> Good Migrations (by application count)



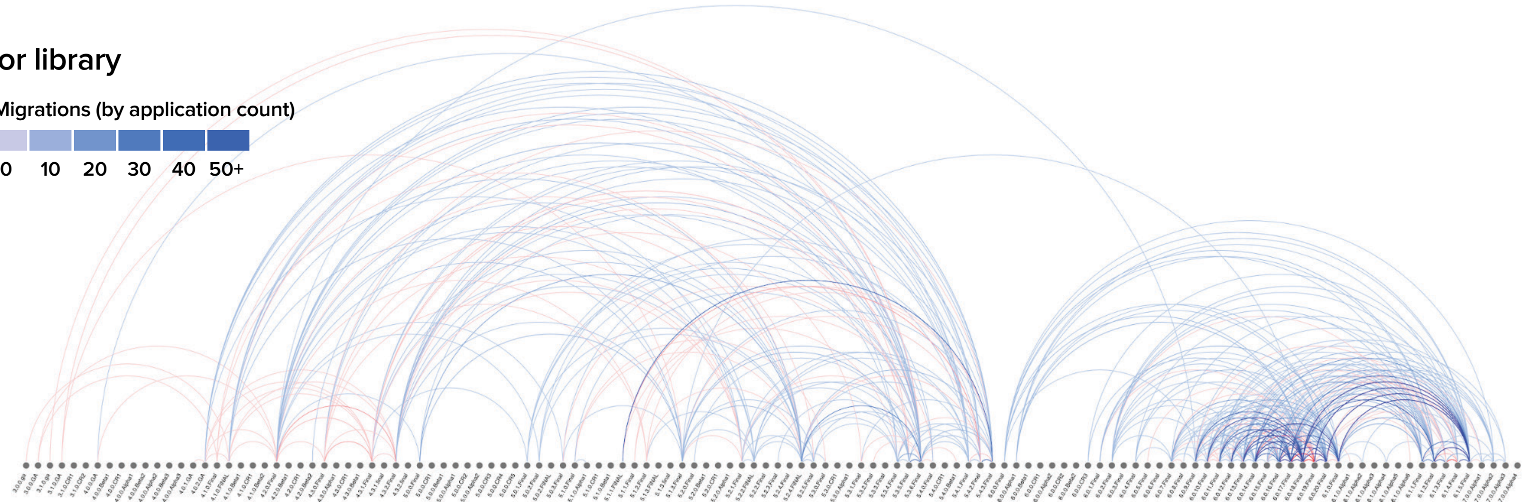
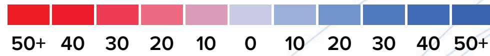


FIGURE 4G

spring.spring-core library

Poor Migrations -> Good Migrations (by application count)



CHAPTER 5

The Trust and Integrity of Software Supply Chains

Enterprise development teams often rely on an unchecked variety of supply from OSS projects where each developer or development team can make their own sourcing and procurement decisions. **Development teams have an inherent trust in their OSS component's authenticity and integrity. Yet the complexity of multi-layered open source software supply chains can obfuscate risk for those seeking to avoid it.**

Choosing open source projects should be considered an important strategic decision for enterprise software development organizations. Just as traditional manufacturing supply chains intentionally select parts from approved suppliers and rely upon formalized procurement practices — enterprise development teams should adopt similar criteria for their selection of OSS components to ensure the highest quality parts are selected from the best and fewest suppliers.

As Jim Zemlin, Executive Director of the Linux Foundation recently remarked, "Open source is an undeniable and critical part of today's economy, providing the underpinnings for most of our global commerce. Hundreds of thousands of open source software packages are in production applications throughout the supply chain, so understanding what we need to be assessing for vulnerabilities is the first step for ensuring long-term security and sustainability of open source software."[33]

FIGURE 5A

Development's Visibility, Awareness and Control of its Software Supply Chain

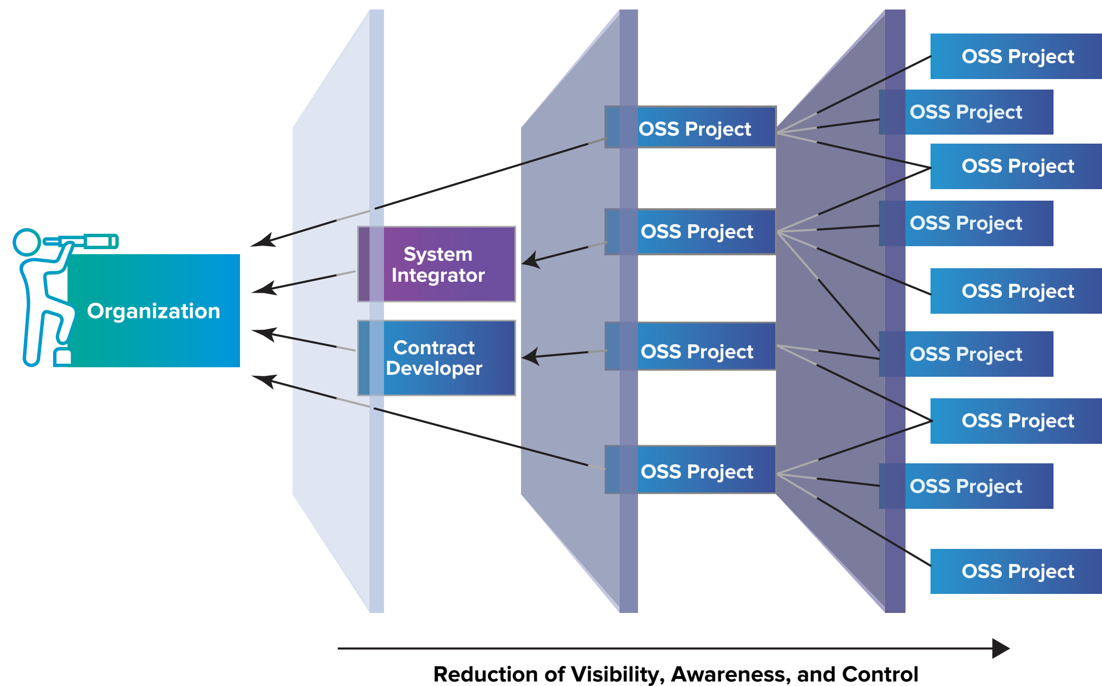


FIGURE 5B

	2013	2014	2015	2016	2017	2018	2019
Percentage of downloads with known vulnerabilities	5.4%	6.2%	6.1%	5.5%	12.1%	10.3%	10.4%

1 in 10 OSS Downloads Are Vulnerable

To better understand how defective and known vulnerable component releases flow through software supply chains, we first have to look at public open source repositories (e.g., Maven Central, npmjs.org, RubyGems.org, NuGet Gallery). Developers download free open source component releases from these internet-based code warehouses in order to build their applications.

For the past seven years Sonatype has analyzed the patterns and practices associated with Java components being downloaded from The Central Repository (FIGURE 5B). **In 2019, 10.4% of the billions of downloads had at least one known vulnerability.'**

Furthermore, research from the University of Darmstadt published in August 2019 revealed that **nearly 40% of all npm packages rely on code with known vulnerabilities**. Perhaps even more concerning is that 66% of security vulnerabilities in npm packages remain unpatched, leaving developers who want to use secure packages with no safe alternatives.[34]

FIGURE 5C

Construct of a Modern Application

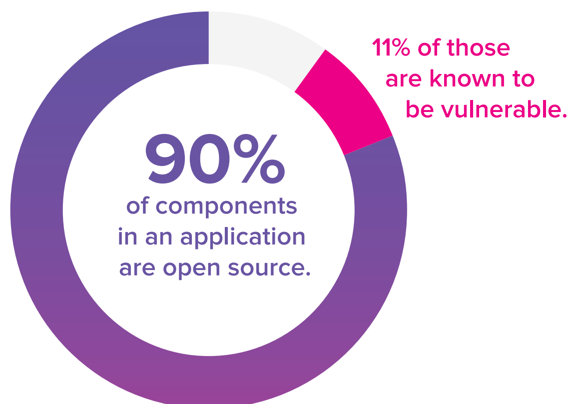
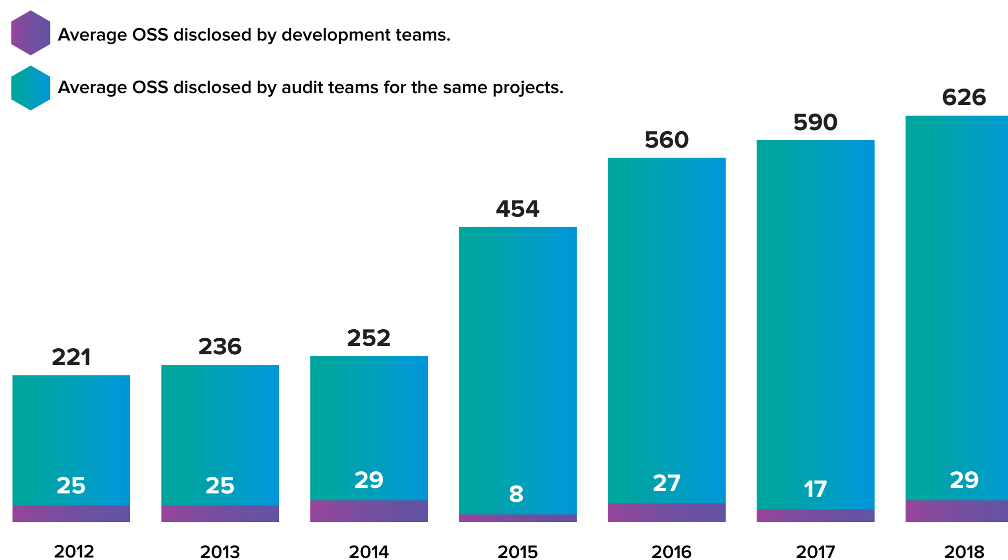


FIGURE 5D

The OSS Knowledge Gap Widens Over Time

SOURCE: Revenera, The Maturity of Open Source Software



Enterprises Rely on Code From 3,500 Suppliers, But Quality Varies

Developers build applications with someone else's code. **Our study of 15,000 enterprise software development organizations revealed an average of 373,000 open source component downloads annually. The downloads represent an average of 3,552 OSS projects** — the external supplier network for code serving modern enterprise development. These downloads represent 11,294 component releases from those projects.

Further analysis of downloads from those organizations reveals that **30,862 (8.3%) included at least one known security vulnerability.** Just as well, not all security vulnerabilities are created equal. Of the 30,862 vulnerable downloads, 68% had Common Vulnerability Scoring System (CVSS) at 7.0 or above on a 10 point scale. Thirty percent (30%) had CVSS scores above 9.0 on a 10 point scale. Minor fluctuations in the percentage of vulnerable downloads were seen on a country by country basis: United States (8.6%), France (8.3%), United Kingdom (8.6%), and Germany (7.81%).

OSS Components Make Up 90% of a Modern Application

Just because a developer downloaded a component does not mean that it was used in an application. To better understand how many open source components were used by developers, we investigated and analyzed 1,700 applications for this year's report. We found that development teams use an average of **135 software components of which 90% are open source.** It was not uncommon to see applications assembled from 2,000 – 4,000 OSS component releases.

Furthermore, **11% of the open source components had at least one known security vulnerability.** On average, the applications contained 38 known vulnerabilities.

While any developer knows that open source components are used to build an application, the enterprise does not carry the same awareness. An analysis of open source component use in organizations by Revenera is telling of software supply chain awareness. **In 2018, development teams using open source in development disclosed their awareness of 29 OSS being used while audits of their environments revealed 626 components — a 22x difference!** (FIGURE 5D)[35]

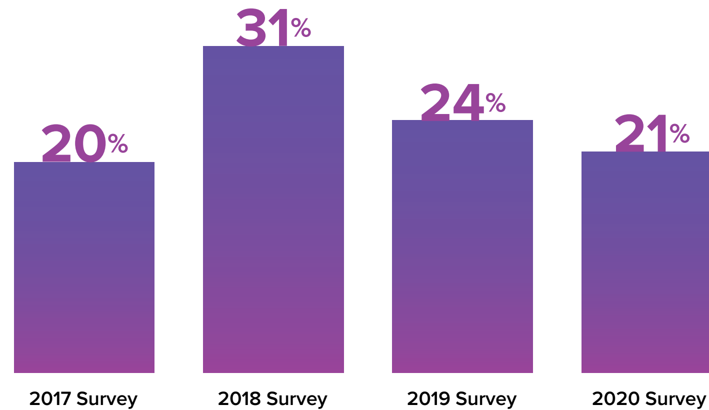
21% of Enterprises Experienced Open Source Breaches

According to the X-Force Threat Intelligence Index attacks on known vulnerabilities increased to 30% in 2019, up from 8% the previous year.[36] Development teams relying on open source components that sometimes contain known vulnerabilities were not immune to these attacks. The 2020 DevSecOps Community Survey of over 5,000 development professionals revealed that **21% had experienced an open source component related breach in the past 12 months** (FIGURE 5E). ■

FIGURE 5E

Open source component related breaches continue to drop, but still occur much too often.

SOURCE: 2020 DevSecOps Community Survey, Sonatype



The Changing OSS Landscape: Social Activism and Government Standards

Social Activism and Open Source Software

Social activism has been high on the agenda of many in the tech community. Developers at Google, WeWork, Kickstarter, Amazon, and other companies across the tech industry have been more active at protesting employer decisions, petitioning them to abstain from doing business with government agencies, and denouncing unfair treatment of employees.

The open source community has not been immune from protests. **In 2017, a developer harassed by corporate lawyers pulled his left-pad code from the npm repository temporarily "breaking the internet"** as numerous automated build environments relying on the code failed.

In September 2019, in an effort to protest his former employer's commercial relationship with the U.S. Immigration and Customs Enforcement (ICE), Seth Vargo removed his "Sugar" code from GitHub and the RubyGems repository. The missing code was eventually replaced, but not before a significant portion of Chef's customers were impacted without warning. Addressing the community as to why he pulled his code, Vargo wrote **"I have a moral and ethical obligation to prevent my source [code] from being used for evil."**[37]

January 2020 surfaced another protest o f sorts when Nikolay Kim deleted his actix-net and actix-web open source project from their public repos after being harassed too many times by his user community. He declared "Being a maintainer of a large open source project is not a fun task... **I am done with open source"** and "I moved actix-net and actix-web project to my personal github account." His action immediately impacted automated builds relying on the code causing many in the community to panic. Kim's code was eventually restored to public repos after he transferred ownership to another developer in the community.[38]

As discussed in Chapter 4, successful productivity outcomes for High Performers were tied, in part, to **keeping a centralized record of applications, their dependencies, and the associated development teams.** Given its association with good outcomes, we recommend the use of repository managers to proxy public OSS repositories and host OSS components locally. Locally hosting any components needed by developers will help improve business continuity during future protests or actions by activists.

Governments Apply New Standards to Secure Software Supply Chains

Secure software practices extend from early development through the active life of an application in the market. With an ever increasing number of application breaches occurring, standards bodies and governments are stepping in to hold development organizations accountable for the quality and security of the code they assemble and build.

United States

OPEN CHAIN PROJECT — LINUX FOUNDATION

In April 2019, the Open Chain Specification, version 2.0, was published to define the key requirements of a quality open source license compliance program. The objective was to provide a benchmark that builds trust between organizations exchanging software solutions composed of open source software.

> "I have a moral and ethical obligation to prevent my source [code] from being used for evil."
>
> — SETH VARGO

Section 3.1 of the specification **called for creating a Software Bill of Materials (SBOM). The SBOM would be used to identify, track, review, approve, and archive information about the open source software components used in a software application, middleware, firmware or operating system.** The specification maintains that an SBOM is needed to support the systematic review and approval of each component's license terms to understand the obligations and restrictions as it applies to the distribution of software.[39]

CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

In May 2019, CISA's Supply Chain Risk Management (SCRM) published a guide for detailing actionable steps on how to start securing software supply chains. Steps recommended building a list of the software components organizations procured, **mapping supply chains to better understand what components were being procured,** determining how organizations would assess the security culture of suppliers, and establishing systems for checking supply chain practices against guidelines.[40]

U.S. CONGRESS

In July 2019, Sen. Mike Crapo (R-ID) and Sen. Mark Warner (D-VA) introduced a bill explaining that software supply chains have proven to be major means through which adversaries seek gain access to weapons systems, IT systems, and communications technology platforms. While not signed into law, the bill had called for **"stronger effort should be placed on securing the vast supply chains of the contractors responsible for developing and producing the defense related capabilities of the United States."**[41]

OASIS OPEN COMMAND AND CONTROL (OPENC2) TC

In October 2019, members of the OASIS Open Command and Control (OpenC2) TC started sharing documents, specifications, lexicons or other artifacts on GitHub aimed to fulfill the needs of cyber security command and control in a standardized manner. Among them, the Department of Defense comply-to-connect use case defined an early step of **querying the new device requesting its "Software Bill of Materials"** and comparing it to policy as part of an acceptance process.[42]

NATIONAL TELECOMMUNICATIONS AND INFORMATION ADMINISTRATION

Over the past year, the National Telecommunications and Information Administration (NTIA) continued its pursuit to establish the **definition formats and standards for a Software Bill of Materials (SBOM).** This multi-year, non-partisan initiative aims to define SBOM concepts and related terms, offers a baseline of how software components are to be represented, and discusses the processes around SBOM creation. The initiative has also detailed the benefits of building and managing SBOMs from the perspective of those who make software, those who choose or buy software, and those who operate it — characterizing security, quality, efficiency, and other organizational benefits.

FOOD AND DRUG ADMINISTRATION

Working hand in hand with the U.S. Food and Drug Administration (FDA), the NTIA produced a report documenting the successful execution and lessons learned of a proof-of-concept exercise led by medical device manufacturers (MDMs) and healthcare delivery organizations (HDOs). The exercise **examined the feasibility of SBOMs being generated by MDMs and used by HDOs** as part of

In July 2019, Sen. Mike Crapo (R-ID) and Sen. Mark Warner (D-VA) introduced a bill explaining that software supply chains have proven to be major means through which adversaries seek gain access to weapons systems, IT systems, and communications technology platforms.

operational and risk management approaches to medical devices at their hospitals.[43]

NATIONAL DEFENSE AUTHORIZATION ACT FOR FISCAL YEAR 2020

In December 2019, the NDAA — now signed into law, called for the U.S. Secretary of Defense to establish pathways for the efficient and effective acquisition, development, integration, and timely delivery of secure software. The Act included the requirement for software security testing that includes vulnerability scanning and also asks for the establishment of DevSecOps practices inside the Department of Defense.

Section 800 of the Act required "assurances that cybersecurity metrics of the software to be acquired or developed, such as metrics relating to **the density of vulnerabilities within the code of such software, the time from vulnerability identification to patch availability, the existence of common weaknesses within such code,** and other cybersecurity metrics based on widely-recognized standards and industry best practices, are generated and made available to the Department of Defense and the congressional defense committees."[44]

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

In April 2020, NIST released new standards for improving software security aimed at helping "software producers reduce the number of vulnerabilities in released software, mitigate the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences."[45]

NIST's Secure Software Development Framework offers several practices to improve the management of open source software supply chains, including:

- **Create and maintain a software bill of materials (SBOM)** for each OSS component used and every proprietary software package created.
- **Securely archive a copy of each release and all of its components** (e.g., code, package files, OSS and third-party libraries, documentation), and release integrity verification information.
- See if there are publicly known vulnerabilities in the OSS software components and services that the vendor has not yet fixed.
- **Ensure each software component is still actively maintained**, which should include new vulnerabilities found in the software being remediated.
- Determine a plan of action for each third party and OSS software component that is no longer being maintained or available in the future.
- Use the results of commercial services for vetting OSS software components.
- **Establish an organization-wide software repository** to host sanctioned and vetted OSS components.
- Maintain a list of organization-approved commercial OSS components and component versions.
- Have a security response playbook to handle a generic reported vulnerability, a report of zero-days, a vulnerability being exploited in the wild, and a major ongoing incident involving multiple parties.[46]

United Kingdom

THE NATIONAL CYBER SECURITY CENTRE: SECURE DEVELOPMENT AND DEPLOYMENT GUIDANCE

The Centre recognized that software development practices are becoming increasingly automated and reliant on open source and third party

New guidance released by the Centre advised that "third party coding frameworks and libraries also need to be considered in the same light as the code you author. If third party components are themselves vulnerable, this is likely to also impact your system."

components. New guidance released by the Centre advised that "third party coding frameworks and libraries also need to be considered in the same light as the code you author. **If third party components are themselves vulnerable, this is likely to also impact your system."**[47]

In an effort to help development teams evaluate their OSS components and reduce security risk, the Centre provided the following eight questions:

- If there is a security vulnerability in the third party components of your code, what security impact may this have on your system?
- **Is the dependency actively developed and maintained?**
- **If a vulnerability is found in one of your dependencies, would you know? Who would fix it?**
- Are you using any old versions of third party code known to contain security vulnerabilities?
- **Do you know anything about the author and maintainer of the dependency? How do they view and approach security?**
- Does the dependency have any history of security vulnerabilities? What's important here is not necessarily that issues are discovered, but how they are handled.
- **If third party code is dynamically included into your product during the build or deployment process, can you ensure that it can't be maliciously modified?** You could achieve this by verifying its origin and integrity, for example.
- If the third party dependency you are using is configurable, consider disabling or removing unneeded functionality which may widen the attack surface of your product.[48]

FIGURE 6A

SOURCE: The Australian Cyber Security Centre (ACSC)

MITIGATION STRATEGY	MATURITY LEVEL ONE	MATURITY LEVEL TWO	MATURITY LEVEL THREE
Patch applications	Security vulnerabilities in applications and drivers assessed as extreme risk are patched, updated or mitigated within one month of the security vulnerabilities being identified by vendors, independent third parties, system managers or users. Applications that are no longer supported by vendors with patches or updates for security vulnerabilities are updated or replaced with vendor-supported versions.	Security vulnerabilities in applications and drivers assessed as extreme risk are patched, updated or mitigated within two weeks of the security vulnerabilities being identified by vendors, independent third parties, system managers or users. Applications that are no longer supported by vendors with patches or updates for security vulnerabilities are updated or replaced with vendor-supported versions.	Security vulnerabilities in applications and drivers assessed as extreme risk are **patched, updated or mitigated within 48 hours of the security vulnerabilities being identified** by vendors, independent third parties, system managers or users. An automated mechanism is used to **confirm and record that deployed application and driver patches or updates have been installed, applied successfully and remain in place**. Applications that are no longer supported by vendors with patches or updates for security vulnerabilities are updated or replaced with vendor-supported versions.

Australia

This year, the Australian Cyber Security Centre (ACSC) has developed prioritised mitigation strategies to help organizations mitigate cyber security incidents caused by various threats (SEE FIGURE 6A). The Centre defined mitigation strategies that could be applied along three maturity levels. For updating third party libraries and patching applications, the guidance recommended mitigating actions within a month at the lowest maturity level and within 48 hours at the highest maturity level, while also recommending automated tooling to track where and when cybersecurity updates had been performed.[49]

Summary

We've observed double and triple digit growth in open source component ecosystems for over a decade. The industry eclipsed 10 billion open source component downloads in 2012 and within five years witnessed 100 billion download requests. With no slowdown in sight, 2020 is on pace to surpass 1.5 trillion download requests.

The purpose of our 6th annual report was to share evidence, practices and outcomes we observed across software supply chains — upstream and downstream. **Our findings are clear. Productivity does not have to come at the cost of reduced security.**

On the supply side, we observed that Exemplary open source projects benefit tremendously from more frequent code commits, dependency updates and releases. The more frequent the updates, the generally more secure the OSS project.

On the demand side, we discovered a range of enterprise practices that influenced successful software supply chain outcomes. **High Performers deployed more frequently, detected and remediated vulnerable OSS components more quickly, and approved new OSS components efficiently.** The High Performers also onboarded developers onto new teams faster and their employees demonstrated high levels of satisfaction on the job.

Our deep examination of consumption patterns, development practices, and cybersecurity hygiene revealed:

- 929 next-generation cyber attacks actively targeting OSS projects over the past year (Chapter 1)
- 608x faster median time to update dependencies and 2.9x more frequent releases for large exemplary OSS projects compared to non-exemplar clusters (Chapter 3)
- 26x detection and remediation of open source vulnerabilities by high performance teams (Chapter 4)
- 11% of OSS components used in applications had at least one known security vulnerability (Chapter 5)
- 21% of development teams experienced an open source related breach in the past 12 months (Chapter 5)

It is encouraging to see exemplary OSS projects and innovative enterprise development teams are delivering high quality, security software at a rapid pace. Their dedication and results are not rare and their performance serves as a benchmark for others to strive for and achieve.

Thank you for reading this year's report. Please share it with others who you feel might benefit from its data, perspectives, and insight. We welcome any feedback that would help us improve our future reports.

Sources

1 https://blog.sonatype.com/open-source-software-is-under-attack-new-event-stream-hack-is-latest-proof

2 https://blog.sonatype.com/octopus-scanner-compromises-26-oss-projects-on-github

3 https://arxiv.org/pdf/2005.09535.pdf

4 https://www.usenix.org/system/files/sec19-zimmermann.pdf

5 https://www.usenix.org/system/files/sec19-zimmermann.pdf

6 https://www.zdnet.com/article/the-linux-foundation-identifies-the-most-important-open-source-software-components-and-their-problems

7 https://medium.com/@bertusk/discord-token-stealer-discovered-in-pypi-repository-e65ed9c3de06

8 https://github.com/dasfreak/Backstabbers-Knife-Collection

9 https://www.cvedetails.com/cve/CVE-2019-14282

10 http://dgb.github.io/2019/04/05/bootstrap-sass-backdoor.html

11 https://github.com/rubygems/rubygems.org/issues/2034

12 https://github.com/rubygems/rubygems.org/wiki/Gems-yanked-and-accounts-locked#19-aug-2019

13 http://arstechnica.com/information-technology/2019/08/the-year-long-rash-of-supply-chain-attacks-against-open-source-is-getting-worse

14 https://www.npmjs.com/advisories/1119

15 https://www.npmjs.com/advisories/1308

16 https://www.zdnet.com/article/two-malicious-python-libraries-removed-from-pypi

17 https://gist.github.com/colby-swandale/11dadff435b02f887fc68178cd4fb0dc

18 https://www.theregister.com/2020/04/21/rubygems_bitcoin_malware

19 https://blog.sonatype.com/saltstack-20-breaches-within-four-days

20 https://www.npmjs.com/

21 https://www.daxx.com/blog/development-trends/number-software-developers-world

22 https://www.zdnet.com/article/programming-languages-python-developers-now-outnumber-java-ones

23 https://www.nuget.org

24 https://www.nuget.org

25 https://www.docker.com/blog/introducing-the-docker-index

26 https://devclass.com/2020/02/05/docker-knits-together-hub-stats-says-pulls-over-8-billion/

27 Exploratory clustering was initially done with the SPSS two-step clustering method, and later was performed with k-means using SciKit Learn, starting from random cluster centers and taking the best of 50 runs. Highly correlated variables were first converted to single dimensions with principal components analysis.

28 We used the SciKit Learn (version 0.21.1) implementation of elastic net regression with alpha=0.1 and an L1 ratio of 0.7.

29 Higher r^2 is better and indicates that the model explains more of the change in outcome.

30 A caveat: our survey relies on self-reported data and we did not have access to direct measures of the behavior.

31 https://puppet.com/resources/report/2015-state-devops-report

32 Eileen M. Uchitelle, "The Past, Present, & Future of Rails at GitHub": https://speakerdeck.com/eileencodes/railsconf-and-balkan-ruby-2019-the-past-present-and-future-of-rails-at-github and "RailsConf 2019 — The Past, Present, and Future of Rails at GitHub," 25 minutes in, https://www.youtube.com/watch?v=vIScxVu00bs

33 https://www.zdnet.com/article/the-linux-foundation-identifies-the-most-important-open-source-software-components-and-their-problems/

34 https://www.usenix.org/system/files/sec19-zimmermann.pdf

35 https://info.flexerasoftware.com/SCA-Ebook-Maturity-Open-Source-Software

36 https://securityintelligence.com/posts/x-force-threat-intelligence-index-reveals-top-cybersecurity-risks-of-2020/

37 https://github.com/sethvargo/chef-sugar

38 https://devclass.com/2020/01/20/rust-framework-dev-says-im-done-with-open-source-has-second-thoughts

39 https://wiki.linuxfoundation.org/_media/openchain/openchainspec-2.0.pdf

40 https://www.cisa.gov/sites/default/files/publications/ict_scrm_essentials_508.pdf

41 https://www.congress.gov/116/bills/s2316/BILLS-116s2316is.pdf

42 https://github.com/oasis-tcs/openc2-usecases/blob/master/Cybercom-Plugfest/uc-A-comply-to-connect.md

43 https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf

44 https://www.congress.gov/116/bills/s1790/BILLS-116s1790enr.pdf

45 https://csrc.nist.gov/publications/detail/white-paper/2020/04/23/mitigating-risk-of-software-vulnerabilities-with-ssdf/final

46 https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04232020.pdf

47 https://www.ncsc.gov.uk/collection/developers-collection/principles/produce-clean-maintainable-code

48 https://www.ncsc.gov.uk/collection/developers-collection/principles/produce-clean-maintainable-code

49 https://www.cyber.gov.au/acsc/view-all-content/publications/essential-eight-maturity-model

Appendix A

Acknowledgments

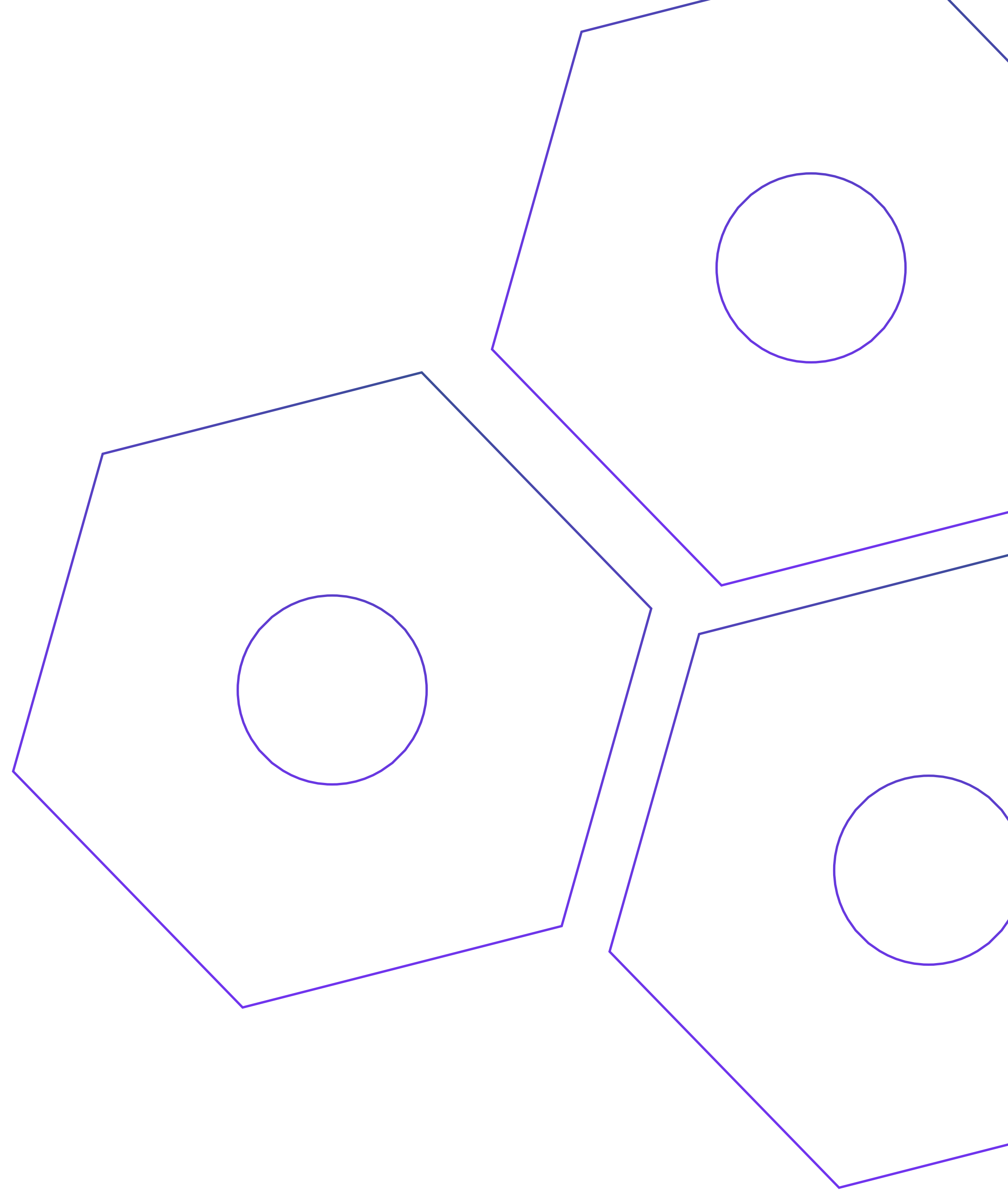
Each year, producing the State of the Software Supply Chain report is labor of love. It is produced to shed light on the patterns and practices associated with open source software development. We began collecting data for our 2020 report from the moment our 2019 report was published.

The report is made possible thanks to a tremendous effort put forth by many team members at Sonatype, including: Derek Weeks, Matt Howard, Joel Orlina, Bruce Mayhew, Gazi Mahmud, Dariush Griffin, Brian Fox, AJ Brown, Ember DeBoer, Mike Donovan, Cameron Townsend, Ilkka Turunen, Alexis Del Duke, Elissa Walters, Adam Cazzolla, Keith Sprochi, and Neil Donewar.

We would also like to offer thanks for contributions big and small from: Hasan Yasar (Carnegie Mellon University Software Engineering Institute), DJ Schleen (Rally Health), and others across the DevOps and open source development community.

A very special thanks goes out to Melissa Schmidt who created the incredible design for this year's report.

Finally, we could not have produced this report without the amazing contributions and countless hours of deep analysis from our research partners Gene Kim from IT Revolution and Dr. Stephen Magill, CEO of MuseDev.



Appendix B

About the Analysis

The authors have taken great care to present statistically significant sample sizes with regard to component versions, downloads, vulnerability counts, and other data surfaced in this year's report. While Sonatype has direct access to primary data for Java, JavaScript, Python, .NET and other component formats, we also reference third-party data sources as documented.

Design of the Survey Questions Used to Analyze Open Source Component Use in Enterprises

Questions were designed to enable quantitative analysis. Most questions were built on a 7-point Likert scale measuring extent of agreement ("strongly agree" to "strongly disagree") or time scales (e.g. "How frequently do you deploy to production?" with options such as "with every change," "multiple times per day," "multiple times per week," "once per week," etc.). Where there were multiple ways to ask about a particular attribute (e.g. "Job Satisfaction"), multiple questions were included and combined into a single dimension for analysis (e.g. "I am satisfied with my job," "I would recommend this organization as a good place to work," "I have the tools and resources I need to do my job," etc.). When multiple questions were combined into a single measure, we verified that the question responses were strongly correlated and used principal components analysis to perform the dimensionality reduction.

Independent Variables Measured When Analyzing OSS Component Use in Enterprises

In our survey of over 600 development professionals to assess how practices and outcomes related to their use of open source components, we measured the following factors to test their effects on the independent variables described above:

DEVELOPMENT PRACTICES

Development philosophy: the general philosophy of development practice used by your team on a spectrum from "waterfall" to "agile / DevOps"

Deployment automation: to what degree are your application deployments (and configurations) automated.

BUILD, TEST, AND RELEASE

Confidence in automated testing: To what degree are you confident that when the automated tests pass the application will operate as intended in production.

Scheduled dependency updates: To what degree is updating open source dependencies scheduled as part of your regular work.

Scheduled patching: To what degree is remediation of security issues treated as a regular part of development work (i.e., security issues are treated as normal defects).

Static analysis tools: To what degree are the output of static analysis tools (e.g., Checkmarx, Coverity, Fortify, etc.) integrated into your daily development workflows.

Artifact repository centralization: To what degree can you centrally analyze all your deployed artifacts (e.g., executable binaries, Docker containers, infrastructure as code, etc.) for open source governance compliance.

OSS SUPPLIERS

OSS selection criteria: What factors are considered when you decide whether to use an OSS component, specifically popularity, feature set, ease of integration, security history (e.g. have there been multiple high-risk CVEs), rate of fixes (frequency of security and bug fixes), OSS license, commercially available support, and foundation/corporate sponsorship.

OSS PHILOSOPHY

Process to add OSS components: The degree to which you use a well-defined process to add new dependencies to an application (e.g., evaluate, approve, standardize, etc.).

Process to remove OSS components: The degree to which do you use a well-defined process to proactively remove problematic dependencies.

OSS enlightenment: The degree to which OSS is supported within the organization, as measured by the following:

- For company-sponsored OSS projects, to what degree are external contributions allowed?
- To what degree does your organization require that all internal modifications to open source components be contributed back (i.e., "pushed upstream")?
- To what degree does your leadership support contributing back to open source components we use (e.g., engineering time, budget, conferences)?

ORGANIZATION AND POLICY

Centralization of asset management: The degree to which there is centralized tracking for every deployed application, its open source dependencies, and ability to contact the application team members.

Centralized OSS governance: The degree to which there is a centralized committee/group/team that is responsible for monitoring and enforcing open source component governance (e.g., security, licensing).

OSS enforcement via automated CI: The degree to which you enforce open source component governance (e.g., security, licensing) through your CI infrastructure.

OSS governance enforcement: The degree to which the open source approval process is consistently followed.

Sonatype is the leader in software supply chain automation technology with more than 300 employees, over 1,000 enterprise customers, and is trusted by over 10 million software developers. Sonatype's Nexus platform enables DevOps teams and developers to automatically integrate security at every stage of the modern development pipeline by combining in-depth component intelligence with real-time remediation guidance.

For more information, please visit Sonatype.com, or connect with us on Facebook, Twitter, or LinkedIn.

Headquarters
8161 Maple Lawn Blvd, Suite 250
Fulton, MD 20759
USA • 1.877.866.2836

European Office
199 Bishopsgate
London EC2M 3TY
United Kingdom

APAC Office
5 Martin Place, Level 14
Sydney 2000, NSW
Australia

Sonatype Inc.
www.sonatype.com

Copyright 2020
All Rights Reserved.