Vulnerabilities in the Core

Preliminary Report and Census II of Open Source Software

The Linux Foundation & The Laboratory for Innovation Science at Harvard

Frank Nagle, *Harvard Business School*
Jessica Wilkerson, *The Linux Foundation*
James Dana, *Laboratory for Innovation Science at Harvard*
Jennifer L. Hoffman, *Laboratory for Innovation Science at Harvard*

www.linuxfoundation.org | www.coreinfrastructure.org | lish.harvard.edu

Contents

Acknowledgments

This report and the research behind it would not have been possible without the leadership of the Core Infrastructure Initiative's co-directors (Karim Lakhani, Jim Zemlin, and Frank Nagle) and Steering Committee, comprised of Josh Corman, Steve Lipner, Audris Mockus, Henning Piezunka, Sam Ransbotham, and David Wheeler. Their deep knowledge and exacting attention to detail have guided and honed this endeavor over the past 18 months. Gratitude and thanks to Michael Dolan and Kate Stewart at the Linux Foundation for their ongoing commitment to this undertaking.

We would also like to thank Brian Warner of the Linux Foundation, who provided the metrics, and Sean Goggins of the University of Missouri, who generated the trend charts in the appendix of this report (also available at **https://chaoss.community**). We are deeply indebted to you all!

CHAPTER ONE

Introduction

CHAPTER ONE

Introduction

Free and Open Source Software (FOSS) has become a critical part of the modern economy. It has been estimated that FOSS constitutes 80-90% of any given piece of modern software,[1] and software is an increasingly vital resource in nearly all industries. This heavy reliance on FOSS is common in both the public and private sectors,[2] and among tech and non-tech companies alike.[3] Therefore, ensuring the health and security of FOSS is critical to the future of nearly all industries in the modern economy.

However, it is difficult to fully understand the health and security of FOSS because 1) FOSS, by design, is distributed in nature so there is no central authority to ensure quality and maintenance, and 2) because FOSS can be freely copied and modified, it is unclear how much FOSS, and precisely what types of FOSS, are most widely used. Therefore, to ensure the future health and security of the FOSS ecosystem, it is critical to understand what FOSS is being used, and how well it is supported and maintained.

In 2014, the Linux Foundation founded the Core Infrastructure Initiative (CII) where its members provided funding and support for FOSS projects critical to global information infrastructure. The CII aims to aggregate support from technology organizations and direct the support to underfunded—but critical—FOSS projects to help ensure the health of the FOSS ecosystem.[4]

In 2015, CII conducted the Census Project (“Census I”) to identify which software packages in the Debian Linux distribution were the most critical to the kernel’s operation and security.[5] Although the Census I project focused on examining the Linux kernel distribution packages, it did not delve deeply into what software was deployed in production applications.[6]

Therefore, in mid-2018, the Linux Foundation partnered with the Laboratory for Innovation Science at Harvard University (LISH) with the goal of conducting a second census to identify and measure how widely open source software is deployed within applications by private and public organizations. This Census II allows for a more complete picture of FOSS usage by analyzing usage data provided by partner Software Composition Analysis (SCA) companies.

In alignment with the ever-evolving nature of the FOSS ecosystem, the CII views the preliminary findings of this second census as a precursor of more exhaustive studies to come in our ongoing efforts to better understand these critical pillars in our information infrastructure. Operating under data constraints, the preliminary findings of this report cannot—*and do not purport to*—be a definitive claim of which FOSS packages are the most critical, but instead represent a starting point upon which future versions will build. The CII plans to release updates and expound upon these insights into private usage of FOSS as more data becomes available.

CHAPTER TWO

Context

CHAPTER TWO

Context

The increasing importance of FOSS throughout the economy became critically apparent in 2014 when the Heartbleed security bug in the OpenSSL cryptography library was discovered. By some estimates, the bug, which was introduced into the OpenSSL codebase nearly three years earlier, impacted nearly 20% of secure web servers on the Internet (almost half a million servers).[7] The vulnerability allowed attackers to obtain access to user passwords and session cookies, essentially rendering ineffective the very security that OpenSSL was built to ensure. Amongst other outcomes, the Heartbleed vulnerability allowed the theft of 4.5 million medical records from a large hospital chain.[8] Operating under the maxim that "with many eyeballs, all bugs are shallow,"[9] many FOSS projects have been able to obtain greater levels of security. Unfortunately, vulnerabilities in other widely-used projects with smaller contributor bases, like OpenSSL, can slip by unnoticed.

Due to Heartbleed and other security issues in FOSS, governmental bodies around the globe have begun to take an expanded interest in the role of FOSS as a type of critical infrastructure that underpins the modern economy. For example, in 2014, the European Commission put into place a FOSS Strategy[10] and a few years later it started sponsoring FOSS auditing by setting up bug bounty programs, hackathons, and conferences.[11]

Compounding the problem is the fact that FOSS is often built into other software and hardware, but precisely what FOSS is being used is not always made clear. This has led to various US government agencies pushing for deeper insights into the software building blocks used to make various packages and devices via a software bill of materials (SBOM), with one working group dedicated to examining the use of FOSS in medical devices.[12] As an outgrowth of these efforts, in April 2018, the leaders of the US Congress House of Representatives Energy and Commerce Committee sent a letter to the Linux Foundation, acknowledging the critical importance of FOSS and exploring the opportunities and challenges related to FOSS, with a particular focus on how sustainable and stable the FOSS ecosystem is.[13]

CHAPTER THREE

Core Infrastructure Initiative's Goals

CHAPTER THREE

Core Infrastructure Initiative's Goals

Similar to physical infrastructure, the critical components of the Internet and modern computing may not always be the most remarkable or the most visible. For example, when the Allies selected strategic bombing targets in an effort to halt the German war machine during World War II, one set stood out from the others: ball bearing factories. The seemingly commonplace products of these factories were crucial components for nearly every aspect of wartime manufacturing, and impacting their production would have significant downstream impacts on the German ability to fight.[14] Similarly, there may be integral FOSS projects whose simplicity or size may belie their vital importance to the modern economy. As such, the overarching goal is to reinforce this infrastructure and guard against systemic vulnerabilities.

Analyzing the usage data from partner Software Composition Analysis (SCA) and application security companies, the Census II project aims to determine how widely FOSS is deployed within applications by private and public organizations. The specific goals of the Census II project are as follows:

1. Identify the most commonly used free and open source software components in production applications.

2. Examine for potential vulnerabilities in these projects due to:
   - Widespread use of outdated versions;
   - Understaffed projects; and,
   - Known security vulnerabilities, among others.

3. Use this information to prioritize investments/ resources to support the security and health of FOSS.

CHAPTER FOUR

Spurring Action

CHAPTER FOUR

Spurring Action

The motivation behind publishing these preliminary findings—as well as anticipated updates in the future—is to not only inform, but also to inspire action by developers and by end-users to support the FOSS ecosystem. While there are many ways to actively support the critical software infrastructure that underpins the world's complex information systems, we offer a few recommendations for unifying action.

Data sharing

In order to tackle a problem, one must first know what may be affected and how. As mentioned above, there is far too little data on actual FOSS usage. Although public data on package downloads, code changes, and known security vulnerabilities abound, the view on where and how FOSS packages are being used remains opaque. Private usage data contributed by partner SCAs and other companies to the CII Census provides a clearer view of which FOSS projects developers built into proprietary software. Additionally, this data enables researchers to trace the dependencies and determine some of the most fundamental—though, perhaps, not the best funded—projects upon which many packages still rely. The insights we can glean from our census efforts will only reach as far as the data sets that FOSS stakeholders (private companies and organizations) share with us. The most critical need for our efforts to support the health and security of the FOSS ecosystem is shared usage data from companies that partner with CII. Any organization that wishes to contribute data to the Census project can visit **https://www.coreinfrastructure.org/programs/census-project-ii/**.

However, usage data only tells one side of the story. The digital infrastructure of FOSS—upon which so much of the economy rests—was built piece by piece, line by line by diligent community contributors. Capturing the contexts in which these developers contribute and the motivations that drive them will help shape more effective interventions and outcomes. In that vein, CII is launching the "FOSS Contributor Survey" in March 2020. This survey aims to poll FOSS contributors annually to further research on the incentives, motivators, and trends driving open source development over time. Any FOSS contributor that wishes to participate in the survey can visit **https://www.coreinfrastructure.org/programs/census-project-ii/**.

Coordination

Beyond predication upon a solid foundation of data, calls to action must coordinate efforts across the whole the FOSS ecosystem. Standardizing terminology and sharing best practices enable the community to build upon previous successes and accelerate progress. Perhaps the largest stumbling block to coordination in the open source sphere is the myriad identifiers used to reference software. FOSS packages live on many different repositories, like NuGet, Maven, GitHub, and npm to name a few. Project names alone may not differentiate between resulting forks of an original project or direct people to the canonical repository. Listing which repository holds the original version of that project (for example, left-pad/npm) can reduce some of the potential confusion. However, identifying a project by the URLs of the repository (location of source code) and the project website (with updated community information and documentation) may be the best solution to ensure clarity. Linking these two URLs as projects will distinguish components more efficiently, even if some FOSS projects move or do not have a public repository.

Accurate project identification impacts not only academia, but the private sector as well. As cyberattacks and security breaches increase, all companies—not just Big Tech—will need to become more cognizant of which components comprise their websites and applications, as well as the origins of those components. In the United States, the federal government is currently creating a Software Bill of Materials which will require all industries to delineate the composition of their software systems. Proactively adopting current standard formats, like Linux's Software Package Data Exchange (SPDX), will put forward-thinking business leaders at an advantage once regulations come into effect.

Investment

Like any critical infrastructure, we must invest in open source if it is to continue to support the demands made upon it. Nadia Eghbal, author of *Roads and Bridges: The Unseen Labor Behind our Digital Infrastructure*, outlined the many sources of financial support available to the FOSS community on her GitHub page "Lemonade Stand."[15] Funding for projects comes in many different forms, including donations, grants, and crowdfunding. Other programs, like Linux Foundation's Community Bridge[16] or GitHub's Bug Bounty Program[17], match open source projects and developers with funding from private companies that rely upon them. While these programs represent a step in the right direction, questions still remain. Without a fuller understanding of what the most critical FOSS projects might be, how do supporters know that sufficient funds will go to where the need is greatest? Are those who benefit most from FOSS projects doing their "fair share" to support the communities behind them?

While money has long been a contentious topic in the FOSS community, investment encompasses more than just financial support. In the open source world, time and talent may indeed be the most important investments. As popularity of particular packages wax and wane, so too do the active contributors. Larger and more established packages tend to attract more contributors[18] than smaller, less visible ones—even if the latter are more heavily depended upon in practice. Companies reliant upon FOSS packages could benefit from supporting them, directly (paying employees to maintain those projects on the clock) or indirectly (hiring contributors to those projects as employees). Similar to financial resources, time and talent need to be carefully considered to ensure that they are directed toward the most critical projects.

CHAPTER FIVE

Methods

CHAPTER FIVE

Methods

The Census II effort benefited from the contribution of private usage data by Software Composition Analysis (SCAs) and application security companies, including developer-first security company Snyk[19] and Synopsys Cybersecurity Research Center (CyRC)[20], who partnered with CII to advance the state of open source research. These SCA partners provided data from automated scans of production systems within their customers' environments, as well as more thorough labor-intensive human audits of software codebases conducted throughout 2018.

In keeping with the spirit of the open source community, CII sought to make the preliminary methodology of this second census effort as transparent as possible. However, in order to ensure the privacy of our data partners and to protect any proprietary aspects of their SCA services, some specific details have been obscured.[21] Ultimately, CII strives to release all future results publicly and transparently, but the commitment to safeguard the sensitive aspects of the data provided must take precedent in this preliminary report.

Data Selection

To better understand the prevalence and overall impact of FOSS in the economy, we chose data that would best reflect actual adoption and usage in businesses. While stars, ratings, and download statistics indicate a package's popularity or reputation, these do not necessarily translate into real-world, day-to-day use. Private usage data from SCA companies' automated scans and human audits from 2018 provides more insight into the inputs to each software package. Instead of the higher-level packages with which end-users would have more contact and familiarity (like Mozilla Firefox or the Apache web server), SCA data focuses on lower level components that act as the building blocks for other software products. This "lower level" focus is important for research, because developers—not end-users—tend to drive the widespread adoption and integration of FOSS projects. As Mike Volpi of TechCrunch noted,

"... the real customers of open source are the developers who often discover the software, and then download and integrate it into the prototype versions of the projects that they are working on.

Once "infected" by open-source software, these projects work their way through the development cycles of organizations from design, to prototyping, to development, to integration and testing, to staging, and finally to production. By the time the open-source software gets to production it is rarely, if ever, displaced."[22]

Peering under the hood, so to speak, of the higher level packages helps to narrow in on the specific components that are most critical. While this data approach does not provide significant insight into end-user facing products (like OpenSSH, for example), it does examine components within those products (like OpenSSH's now infamous lower level library, OpenSSL).

Defining Relevant Terminology

Before delving into the methodology, there is a need to establish consistent terminology when discussing FOSS data. To start, this report relied upon the following definitions laid out in previous CII Census efforts:[23]

- **package:** a unit of software that can be installed and managed by a package manager.
- **package manager:** software that automates the process of installing and otherwise managing packages.
- **repository:** a location for storing and managing the history of information (such as software).

The various methods employed to scan and audit codebases that generated the private usage data, led to analysis which occurred below the **"package"** level. Sometimes a given software project depended on a distinct part of a package, even though it did not appear to depend on the other parts of that package. As a result, CII defined a separate term:

- **component:** a unit of software that can be called by or serves as an input into another piece of software.

The datasets used for this census contained FOSS information at a variety of levels, often treating a package and its subcomponents as separate entities. In order to compare across all datasets, we standardized this **component-level** data first.

Methods Part 1: Parsing

In collaboration with the Linux Foundation, the research team iteratively refined the methodology for combining the private usage data from the SCAs—complex datasets with substantially different means, variance, and schema for identifying unique components. Parsing each dataset generally occurred in three stages.

- **Stage 1:** Cleaning the dataset to remove organizational-specific substrings, whitespace, or other extraneous characters.

- **Stage 2:** Extraction of identifying information from each component in the dataset.

- **Stage 3:** Mapping each component to a project on Libraries.io[24] using that identifying information, if possible.

Not all FOSS packages have a unique identifier. Therefore, to aggregate usage data within and across different datasets, we had to map each component in each dataset to a unique identifier. Here a unique identifier is defined as the combination of the package name (e.g., "lodash") and the package manager which hosts that package (e.g., "npm").

The process of mapping dataset components to a Libraries.io Project relied on several functions in tandem, based on the identifying information found in the dataset.

1. Searching components for embedded unique identifiers (GitHub repository, name and package manager) that can map to Libraries.io projects.

2. Analyzing component text for a naming system that can be translated to the naming system on Libraries.io or the package managers from which it pulls data.

3. Searching components for specified text strings that directly map to a Libraries.io project.

4. Manual matching to a Libraries.io project if all other methods cannot effectively map the component.

While the majority of components provided by SCA datasets automatically matched to Libraries.io in this manner, many components had to be manually mapped. Furthermore, some components in the private datasets did not exist on Libraries.io. These components were still treated as real packages, but could not be used to calculate indirect dependencies.

Methods Part 2: Dependencies

Indirect dependencies are a useful tool for understanding which packages are the most essential to their software ecosystem. If Package A is considered important, then everything that Package A directly uses to function is also important, and all of the packages those packages depend on are important, and so on. Therefore, including indirect values in our resulting dataset was a way to find the "hidden keystones" in the FOSS ecosystem that might be overlooked by a direct audit or scan.

In cases where data partners provided these indirect metrics, we added those calculations to the direct metrics to account for both types. In cases where that data was not provided, we estimated the indirect usage through Libraries.io, which collects dependency information through the package managers from which it pulls data.

Using the SCA datasets provided, we identified indirect usage using the following process:

1. Using the dependency data provided by Libraries.io, filter out non-runtime dependencies, filter out optional dependencies, and filter out self-dependencies.
2. For each component, determine the list of packages that component relies on, directly or indirectly. Each member of the list, including the original component, receives a score equal to the number of times that component was observed being used in the SCA datasets. For example, if "jquery" had a score of 10, each package "jquery" depends on (as well as "jquery" itself) would have 10 added to their score.

3. Extract the top-scoring packages from the network. Notably, if a package is stored under multiple names in its package manager, Libraries.io will give each instance a unique identifier. For example, if a project like ‘JSONStream’ has a deprecated version of that package like “jsonstream’ stored separately in npm, Libraries.io will assign each a unique identifier. For those packages with multiple identifiers in Libraries.io, the average of the scores across all of these identifiers was taken as the final score for that particular FOSS package.

There were a number of challenges associated with this process that are still under consideration and will be dealt with in future versions of this report.

First, as the dependency network was taken from a Libraries.io dataset, only projects appearing on Libraries.io can be indirectly evaluated this way. As such, packages not on a Libraries.io connected package manager did not show up in the top package list.

Second, since dependencies for a given package varied across versions, the preliminary results overcounted the true number of “necessary packages” per component; as a result, packages with a more diverse pool of dependencies over multiple versions had a greater influence over the whole network than others with fewer versions. Without complete versioning data, however, this issue was difficult to avoid. More detailed data will help to address this issue in the future.

Limits to Dependency Network

When using dependency networks like the ones provided by Libraries.io, researchers must select which types of dependencies are considered relevant to the calculation. Not all dependencies are created equal; some inputs to a software component are more essential than others. If researchers have access to this kind of granular information, they can weight dependencies using that information, which would likely create a drastically different result. The section below highlights the reasoning behind the exclusion of certain types of dependencies in the preliminary results.

One of the first choices made was to exclude dependencies which were flagged as being “optional” dependencies. If Component B is not always an input for Component A, then we cannot assume that one instance of A indicates one instance of Component B as well. Therefore, we ignored these “optional” dependency links.

Another decision was to set aside version specifications provided by the Libraries.io dependency dataset. Information on dependencies appears in the format “Project A, Version B depends on Project C”, and then a string that describes the valid versions for Project C (“>=1.3.4, <1.2.1”). Integration of version-specific dependencies would require a standardized format like “Project A, Version B depends on Project C, Version D”. Because we set aside version specifications, “Project A” links to every package upon which “Project A” has ever depended. “Project A” might have depended on a legacy package in older versions, but not in recent versions. Therefore, by ignoring versioning, the probability of overcounting the number of dependencies per project increases and may skew the results.

In future census efforts, CII intends to include version-sensitive dependencies to make these calculations more reflective of the true FOSS ecosystem. Before that information can be included, though, contributed datasets would need to include full or nearly-full versioning information and the dependency data provided by Libraries.io would need to be reconfigured into a version-to-version format.

The third choice made was to exclude non-runtime dependencies. Build software components tend to be massively interdependent, resulting in dependency loops where an extensive chain of FOSS components were all linked together in a circle. If these "build loops" could be eliminated, then future census reports might have better insight into build projects and ensure they receive accurate acknowledgement in the results.

Methods Part 3: Combine

Once the indirect usage was added into each SCA dataset provided, the top ten packages were identified using the following process:

1. Drop the long tail of each dataset.
2. Calculate the average Z-score[25] of the remaining packages relative to the datasets in which that package appears. This approach allows us to proportionally compare the importance of that package across multiple differently-sized data sets.
3. Calculate the rank for each package based on their respective Z-scores.
4. Map each package to its equivalent GitHub repository, if applicable.

Considerations

The final integrated data for this census is unique in that it represents a snapshot of usage by private companies integrated with dependency data. However, like any sample dataset, it has limitations on how fully it can represent the ground truth of all the FOSS projects in use. Analysis of the aggregated data uncovered several considerations to keep in mind when reviewing these preliminary results.

The first consideration to take into account is the fact that FOSS projects exist in many different ecosystems, written in many different languages. The data sources provided snapshots of how companies use FOSS projects, but did not indicate that one FOSS ecosystem or language is any more important than another. The data received from partner SCAs contained a large amount of software from the JavaScript ecosystem. Additionally, small packages are extremely common in the JavaScript npm package system. For example, in npm, 47% of the packages have 0 or 1 functions, and the average npm package has 112 physical lines of code.[26] In contrast, the average Python module in the PyPI repository has 2,232 physical lines of code.[27] These two factors caused the dependency calculations to crowd out non-JavaScript packages. To try to re-capture these crowded-out packages, we created a separate set of results to identify the top packages when JavaScript packages are excluded.

Secondly, FOSS projects exist across time in a multitude of forms. Several instances of deprecated projects or projects which have not been updated for a few years appeared in the usage data provided by SCAs. Codebases often contain “legacy software” like these, but deeper investigation would be needed to differentiate whether these components were still actively called upon or were cached as “gold masters” for use in characterization testing.[28] As a result, a census reliant upon scan and audit data will inherently reflect more older projects, or versions, over newer ones. However, until the role of these legacy packages can be determined, they may warrant more proactive approaches, including efforts to help revitalize these projects or provide assistance for end-users who would like to transition over to newer projects.

A final consideration is the fact that FOSS projects are used by different groups for different purposes. Utilizing FOSS usage information, Census II avoided a previous roadblock: determining which projects are “real” and “relevant”. However, the sample size was limited to the particular customer bases of the respective Software Composition Analysis (SCA) firms who provided data. Furthermore, privacy concerns prevented the provision of data with the level of specificity necessary to undertake representative sampling.

Longstanding roadblocks identified prior to the launch of Census II continue to present challenges. The question of how to incorporate versioning into dependency networks, for example, remains unresolved.

The reliance upon identifying information provided by Libraries.io or GitHub inherently excludes packages that do not appear on either platform, pushing them out of the top ranks during the dependency calculations run for this report. It is unclear whether the inclusion of other datasets from sources like Debian, Wordpress, and Drupal would alleviate or exacerbate these problems.

Under these constraints the preliminary findings of this report cannot—*and do not purport to*—be a definitive claim of which FOSS packages are the most critical.

The calculations provide greater insight into which packages are the most important for the companies and organizations served by CII’s data partners. FOSS software that is essential in one sector may not be used in another. These preliminary results undoubtedly reflect distributions specific to each customer base, but they also provide a rare glimpse into data on private usage of FOSS unavailable to most researchers. CII encourages more companies and organizations to join the Census II effort as data partners, but until more private usage data becomes available, the study must work within this limited set.

CHAPTER SIX

Preliminary Results

CHAPTER SIX

Preliminary Results

The preliminary results of the CII Census II are meant to serve as a “proof of concept” of our current methods. We are committed to providing a clear picture of the end-product that these methods create, especially as the project continues. However, we also want to refrain from making any definitive claims about the packages listed; these lists will undoubtedly shift over time, as we integrate more extensive datasets and refine our dependency analysis algorithm.

The preliminary results take the form of two lists. The first list identifies the **ten most used packages** from our dependency analysis, listed in alphabetical order in **Appendix A** (see page 34). The second list contains the ten most used non-JavaScript packages, also presented in alphabetical order in **Appendix B** (see page 45). Given that JavaScript is heavily represented in our data sources and encourages the proliferation of packages, JavaScript packages will dominate any ranking we create. To account for this, the most used non-JavaScript packages list aims to give a sense of what other kinds of packages are keystones of the FOSS ecosystem. Notably, this second list experiences a similar problem, as Java packages dominate all others.

To give greater context to these packages and how they operate, CII partnered with the CHAOSS[29] project, a Linux Foundation initiative focused on creating metrics and analysis tools to evaluate the health of FOSS communities. The name of each project links to a page containing more detailed CHAOSS metrics, including graphs of commits per week and lines of code added per week. The project description of the project, as it appears on the associated GitHub repository, appears directly after the project name. The size of each project is listed as Total Lines of Code, which were measured in January 2020.[30] Specific data from 2018, such as the number of active contributors[31] and commits, have been included in the accompanying tables to provide a timeframe and context for each project comparable to the private SCA-collected data. In an effort to reveal longer term trends, we have also included graphs to display longitudinal data about project activity. Please note that these graphs show data beyond the 2018 timeframe.

Insights into Top Committers

After identifying the ten most used packages and ten most used non-JavaScript packages from our dependency analysis, more insights emerged from public project data about the communities behind them. By running a query on GitHub data over the life of each of these repositories, we were able to determine the top three committers for each of these FOSS projects. To get a fuller view of total contributions, the query examined not only users who pushed the commits, but also users who authored commits and actually committed. By manually cross-referencing public GitHub profile information with data sources like LinkedIn, Crunchbase, and other publicly-available data from social media and networks, company affiliations for the majority—over 75%—of the top committers could be determined. Some of those contributors may have had multiple affiliations with different companies over the length of the respective FOSS projects. Additionally, some contributors may have had periods of self-employment (approximately 15% of the top committers, labeled as “Independent”). For the remaining 10% of the top committers, there were no known or no found affiliations.

These statistics illustrate an interesting pattern: a high correlation between being employed and being a top contributor to one of the FOSS packages identified as most used. Contrary to popular image in open source discussions of “the overworked and underpaid programmer,” an analysis of 2017 GitHub data found that some of the most active FOSS developers contributed to projects under their Microsoft, Google, IBM, or Intel employee email addresses.[32] Even if the contributors to the projects listed in the appendices do not receive direct compensation from private companies to develop these packages, their status as a member of the FOSS community could have endorsed their qualifications for their current paid employment. However, no conclusions can be drawn without greater visibility into the unique circumstances under which contributors operate and direct data to support those hypotheses. To address this gap, in the coming months, CII will pilot a survey of thousands of contributors associated with the FOSS packages identified by this preliminary census report. The survey will explore the contributor’s level of engagement, employment history, and employer’s policies on developing FOSS in the workplace. A more thorough understanding of the forces at work will help FOSS stakeholders—individual contributors, open source foundations, and companies alike—better allocate resources and support in the future.

CHAPTER SEVEN

Lessons Learned

CHAPTER SEVEN

Lessons Learned

As a separate result of the Census II effort, the CII team identified several "lessons learned" throughout the initial stages of the project. While these lessons learned do not impact the substance of the findings—nor lists of most used packages—we believe these results are important to the broader conversation and merit exploration.

The Need for a Standardized Naming Schema for Software Components

Members of the Census II team and the Steering Committee spent months in the time leading up to the project's acquisition of data attempting to anticipate and prepare for expected obstacles and challenges to the data's use and analysis. The challenges created by the lack of a standardized naming schema for software components that had vexed the Census I effort persisted. The naming conventions for software components across all the data contributed to the Census II effort were unique, individualized, and inconsistent. The effort required to untangle and merge these datasets slowed progress on the current project significantly. Despite the considerable effort that went into creating the framework to produce these initial results for Census II, the challenge of applying it to other data sets with even more varied formats and naming standards still remains.

The struggles with this lack of standardized software component naming schema are not unique to the CII Census projects. The National Institute for Standards and Technology (NIST) has grappled with this issue for decades in the context of software vulnerability management. Stakeholders working with the National Telecommunications and Information Administration (NTIA) Software Component Transparency process have wrestled with the same problem. For some—including the Census II and NTIA software bill of materials (SBOM) projects—the largest consequence of the lack of a naming schema has been lost time. However, as SBOM and other software supply chain transparency and security efforts continue to grow, mature, and become more complex, the lack of a standardized software component naming schema threatens to stymie efforts by industry and government to better protect themselves from software-based incidents.

The bottom line—revealed by the Census II project, the NTIA process, NIST's vulnerability management struggles, and other similar projects—is that there is a critical need for a standardized software component naming schema. Until one exists, strategies for software security, transparency, and more will have limited effect. Organizations will remain categorically unable to communicate with each other on the large-scale—particularly, the global scale—necessary to share such information. Given the increasing frequency and sophistication of cybersecurity incidents in which the software supply chain plays a part, there is precious little time to waste.

The Increasing Importance of Individual Developer Account Security

The next challenge and lesson learned that arose after the data had been analyzed was the criticality of the security of individual developer accounts. Of the top ten most-used software packages in our analysis, the CII team found that seven were hosted under individual developer accounts. The consequences of such heavy reliance upon individual developer accounts must not be discounted. For legal, bureaucratic, and security reasons, individual developer accounts have fewer protections associated with them than organizational accounts in a majority of cases. While these individual accounts can employ measures like multi-factor authentication (MFA), they may not always do so and individual computing environments may be more vulnerable to attack. These accounts do not have the same granularity of permissioning and other publishing controls that organizational accounts do. This means that changes to code under the control of these individual developer accounts are significantly easier to make, and to make without detection.

These potential risks are not hypothetical; developer account takeovers have begun occurring with increasing frequency. "Backdooring" is one popular method used to infiltrate accounts: hackers insert malicious code into seemingly innocuous packages that create a "backdoor" for hackers to enter once the host package is installed. Perhaps the most famous example—though not a "strict" account takeover—involved the backdooring of the popular event-stream JavaScript library. There, a malicious actor gained legitimate publishing rights to the event-stream package, and then wrote a backdoor into the package itself.[33] Separately, in July 2019, a Ruby developer was alerted to the fact that their account with the official Ruby repository had been taken over, and several of their packages backdoored. Later, in August 2019, a similar account takeover was executed once again at the Ruby repository, leading to the backdooring of eleven packages.[34]

While developer account takeovers remain a significant risk to software security, there are other problematic issues that might be less obvious. One example are developers who decide to remove or "delete" their developer accounts. This happened in 2016 with a package called "left-pad," with consequences that stakeholders described as “breaking” the Internet for several hours. There, a developer who was upset with the outcome of a package naming dispute removed their code from the npm repository in protest. It was quickly discovered that hundreds of downstream packages depended upon that seemingly minor piece of code. Without that critical left-pad code, these downstream packages broke.[35] Similarly, in 2019, a developer who disagreed with a business decision undertaken by Chef Software removed their code from the Chef repository with similar downstream impacts to that of left-pad.[36]

Thus, in the contexts of both security and general risk management, it is critical that developer accounts be understood and protected to the greatest degree possible. With this in mind, the Linux Foundation focuses on developer account security in two of its major projects: the Core Infrastructure Initiative badging program[37] and the more recently launched Trust and Security Initiative. Both of these projects wrap developer account security into their controls to mitigate these risks as part of a holistic security program.

The Persistence of Legacy Software in the Open Source Space

The last lesson learned was more subtle than the discovery of the criticality of developer account security. In conversations with JavaScript ecosystem experts about the rankings derived from the Census II data pool, these experts were struck by the relative position of software package “minimist” as compared to software package “yargs”. The two packages performed essentially the same functions, but yargs was generally considered the newer (and better) replacement for minimist. However, minimist showed up several rankings higher than yargs in the Census II rankings.

This suggests that a generally accepted reality exists within the FOSS development space: open source has not escaped the problem of legacy technology. In this specific case, the “legacy tech” is a single software package whose replacement has not yet overtaken its predecessor in terms of sheer usage. Software should arguably be easier to replace within a live system, as it does not involve replacing hardware. In cases where the legacy-to-replacement packages perform generally the same function, the new package could be slotted in with relatively minor disruption to the full product overall. However, in many cases this may not be true: compatibility bugs abound. More likely to be problematic, however, are the financial and time-related costs associated with switching to new software when there is no guarantee of added benefit. For organizations who have not yet experienced a problem with minimist instead of yargs for example, these transition costs may sway an organization against switching to the newer package.

That attitude neglects to take into consideration a separate, related reality of technology in general and FOSS in particular: as technology ages—both software and hardware—it loses support. In the case of FOSS like minimist, the number of developers working to ensure updates—including feature improvements, as well as security and stability updates—decreases over time. Often those developers instead choose to dedicate their time and talents to newer packages. As a consequence, those legacy software packages become more likely to break with each passing day without the guarantee of support on-hand to provide fixes. Although this was not the path that led to the Heartbleed situation discussed above, this path could lead to similar large-scale negative outcomes. Thus, it is equally critical that legacy tech issues be considered in the FOSS space, just as they are in the general technology context. Without this awareness, and especially without processes and procedures in place to address the risks created by legacy FOSS, organizations open themselves up to the possibility of hard-to-detect issues within their software bases.

CHAPTER EIGHT

Conclusion

CHAPTER EIGHT

Conclusion

We understand that these findings are not comprehensive, but with the usage data provided, we hoped to shed a bit more light on what FOSS packages are used—or heavily depended upon—within private companies. Far from being the final word on critical FOSS projects, this census effort represents the beginning of a larger dialogue on how to identify crucial packages and ensure they receive adequate resources and support.

Next Steps

This preliminary report from the Core Infrastructure Initiative Census II effort represents the first steps toward addressing the structural issues that threaten the FOSS ecosystem. CII supports efforts to standardize unique software identifiers (i.e., linking project URLs with repository URLs, SHA checksums, etc.) across the public and private sectors to facilitate better data sharing and aggregation for research. Additionally, we advocate for the inclusion of comprehensive version information in SCA data for both packages observed in scans and audits as well as dependency data. Better standardized and more comprehensive data would enable research efforts, like the Census II, to provide an even clearer picture of which components of the FOSS ecosystem need critical support.

The initial findings of the census have provided valuable insights, but CII also strives to outline the positive impact of FOSS. As the network of usage data contributors to CII grows, we aspire to provide a more accurate estimate of the economic importance of FOSS. To better understand the communities building and maintaining this critical information infrastructure, we plan to launch a longitudinal survey of FOSS developers in March 2020. In addition to capturing the demographic information of contributors in the open source community, the survey will explore the intersection of time spent on FOSS development and employment. By asking deeper questions about time allocation and employer policies toward open source, this endeavor aims to clarify the often blurry lines between direct and indirect support of FOSS projects in the private sector, as well as the sustainability of the FOSS ecosystem. Responses from the people most closely involved in FOSS will inform and guide future FOSS community-building efforts, including funding initiatives, badging programs, and code development norms. Going forward in these efforts, CII welcomes new partnerships with organizations and individuals willing to contribute more comprehensive data and more precise methods to fortify the security and sustainability of the free and open source software community. For more information about Core Infrastructure Initiative's research, or to express interest in partnering with us, please visit our website at **https://www.coreinfrastructure.org/programs/census-project-ii/**.

APPENDIX A

Most-Used Packages

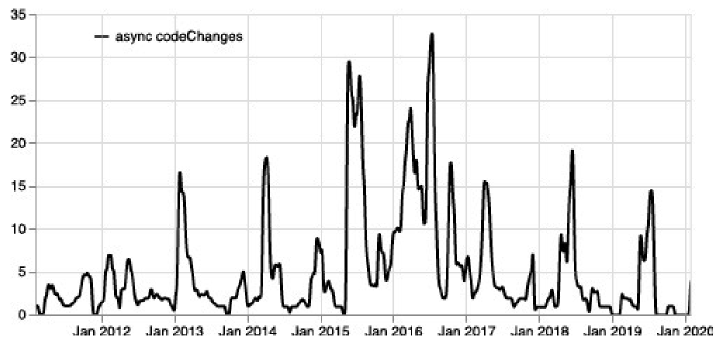
Our dependency analysis identified the following ten packages—listed in alphabetical order below—as the most used FOSS packages among those reported in the private usage data contributed by SCA partners. For further information on how this list was compiled, refer to the Methods section.

async

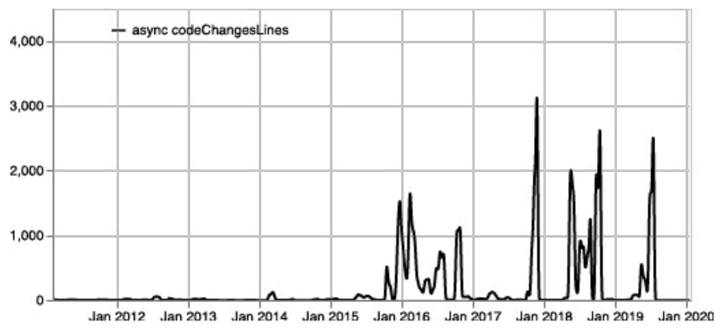
A utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. Although originally designed for use with Node.js and installable via npm install async, it can also be used directly in the browser.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/caolan/async	196,700 Lines	Authors: 22 Commiters: 7	86 total 1.65/week
As of February 7, 2020, this project has 11 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week

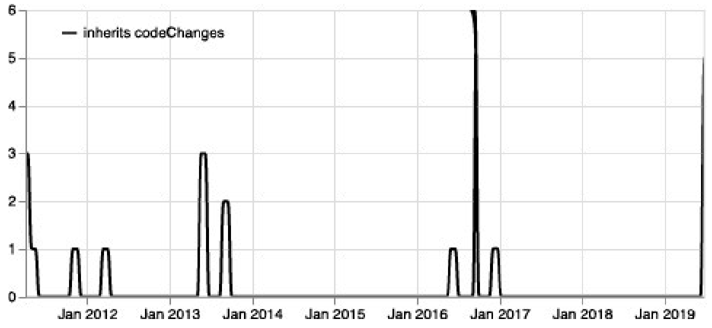


inherits

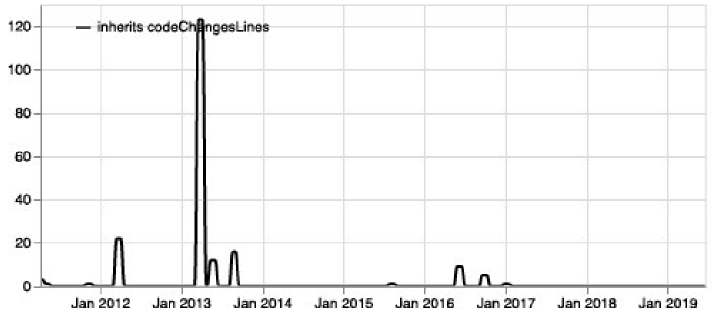
Browser-friendly inheritance fully compatible with standard node.js inherits.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/isaacs/inherits	3,800 Lines	Authors: 3 Commiters: 1	Gap, no commits between December 15, 2016 and June 19, 2019
As of February 7, 2020, this project has 3 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week



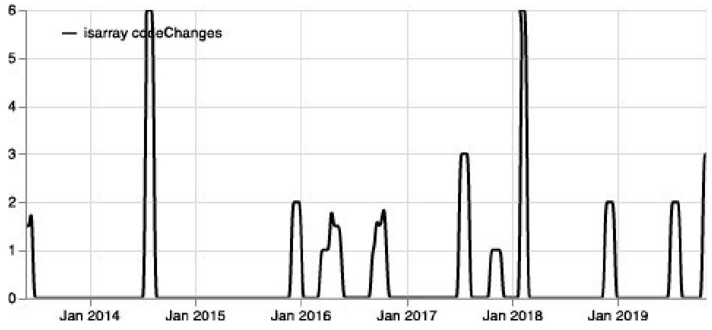
isarray

Array#isArray for older browsers and deprecated Node.js versions.

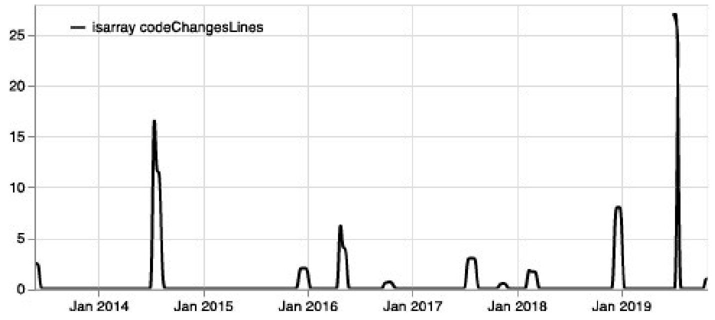
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/juliangruber/isarray	317 Lines	Authors: 3 Commiters: 3	8 total 0.15/week

As of February 7, 2020, this project has 4 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week



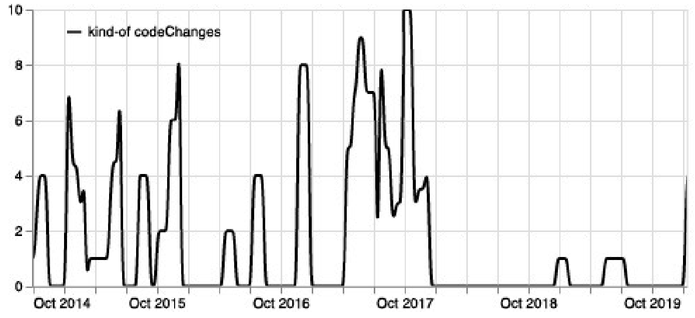
kind-of

Get the native JavaScript type of a value.

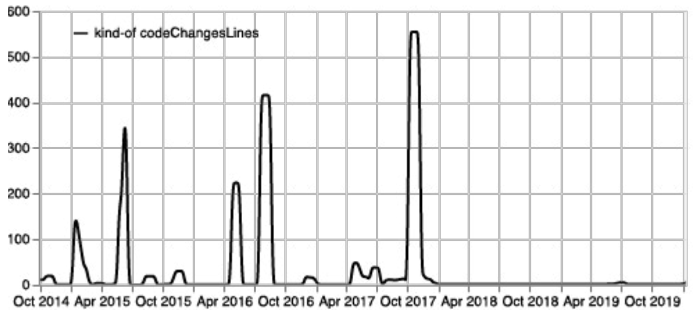
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/jonschlinkert/kind-of	2,000 Lines	Authors: 11 Commiters: 11	Gap, no commits between 2017-12-01 and 2019-05-25

As of February 7, 2020, this project has 3 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week

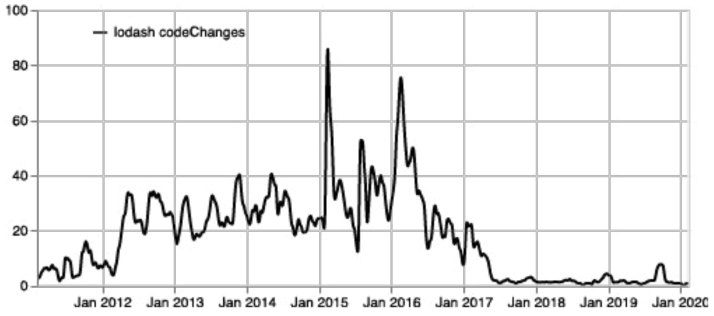


lodash

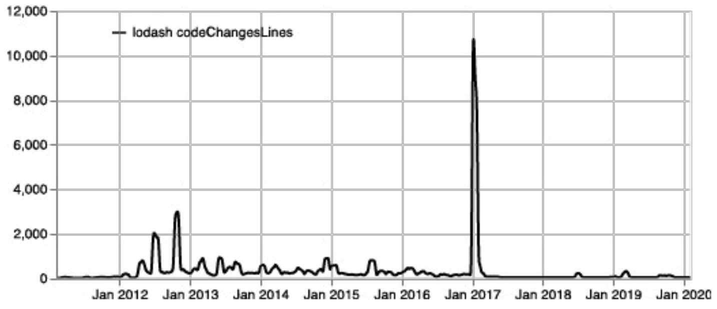
A modern JavaScript utility library delivering modularity, performance & extras.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/lodash/lodash	42,300 Lines	Authors: 28 Commiters: 2	58 total 1.12/week
As of February 7, 2020, this project has 30 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week

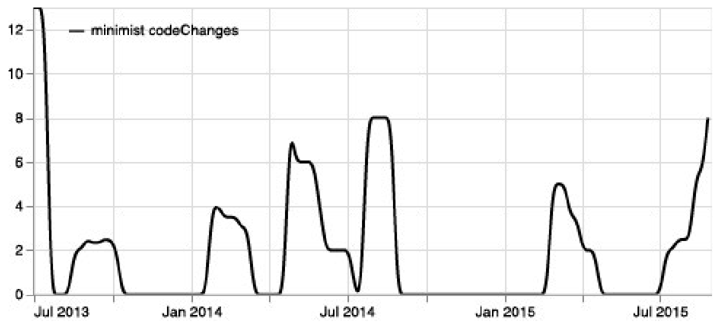


minimist

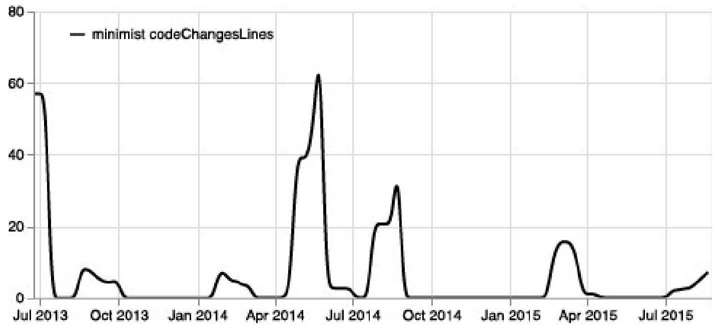
Parse argument options. This module is the guts of optimist's argument parser without all the fanciful decoration.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/substack/minimist	1,200 Lines	Authors: 14 Commiters: 6	Last commit: August 29, 2015
As of February 7, 2020, this project has 38 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week



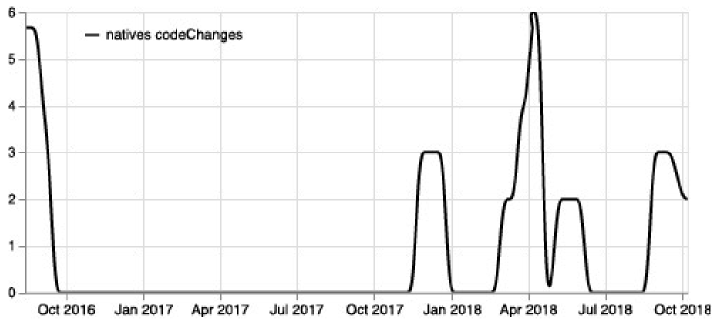
natives

Do stuff with Node.js's native JavaScript modules.

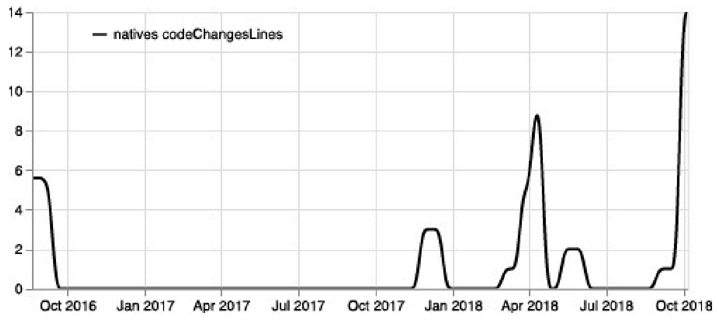
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/addaleax/natives	3,000 Lines	Authors: 2 Commiters: 1	15 total 0.29/week Last commit: October 8, 2018

As of February 7, 2020, this project has 0 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week

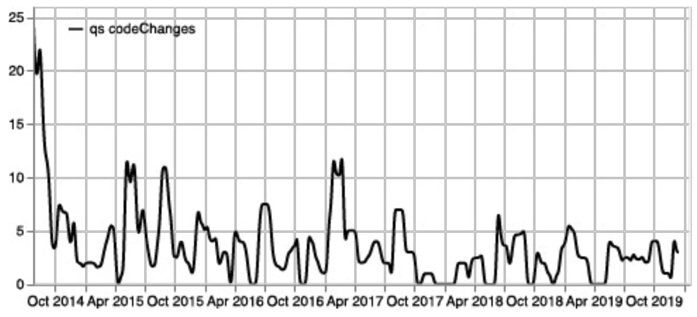


qs

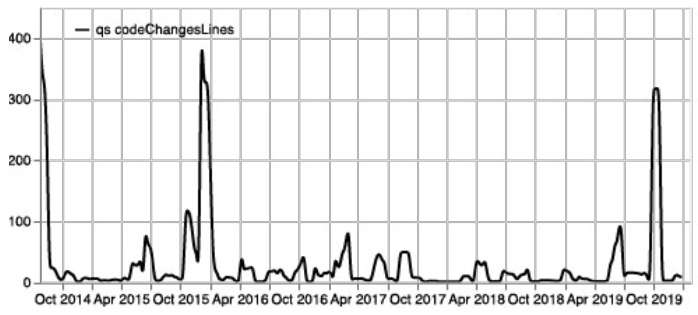
A querystring parsing and stringifying library with some added security.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/ljharb/qs	5,400 Lines	Authors: 5 Commiters: 2	36 total 0.69/week
As of February 7, 2020, this project has 41 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week



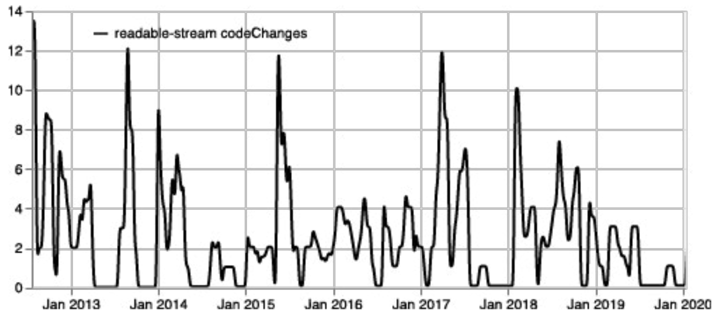
readable-stream

Node.js core streams for userland.

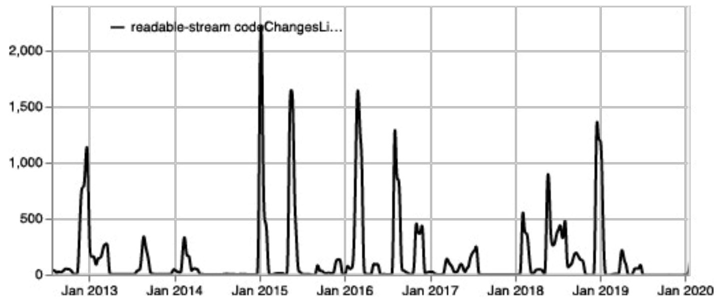
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/nodejs/readable-stream	28,100 Lines	Authors: 10 Commiters: 3	69 total 1.33/week

As of February 7, 2020, this project has 21 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week



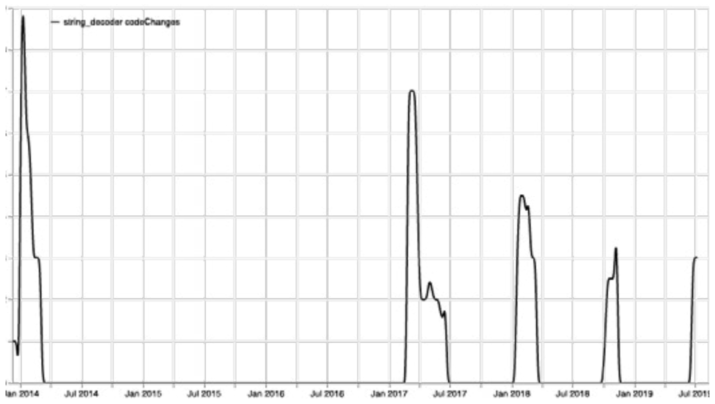
string_decoder

Node-core string_decoder for userland.

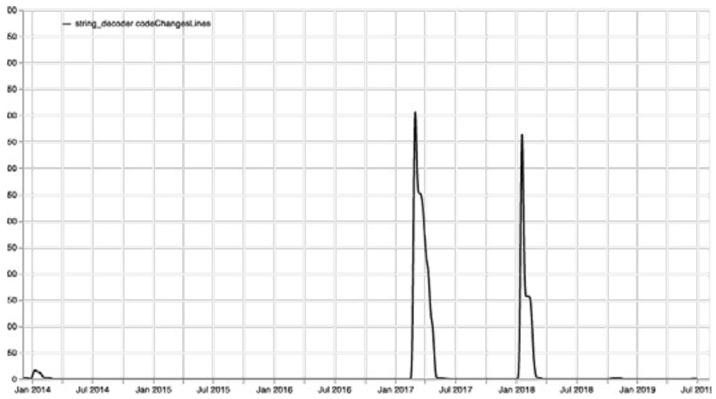
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
npm	github.com/nodejs/string_decoder	4,200 Lines	Authors: 4 Commiters: 3	17 total 0.32/week

As of February 7, 2020, this project has 3 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week



APPENDIX B

Most-Used Non-JavaScript Packages

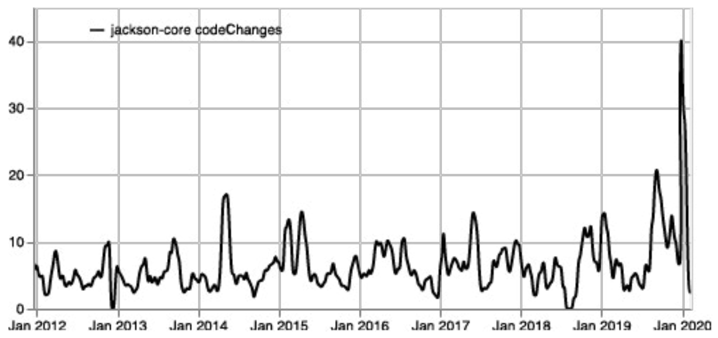
Our dependency analysis identified the following ten packages—listed in alphabetical order below—as the most used, non-JavaScript FOSS packages among those reported in the private usage data contributed by SCA partners. For the rationale behind creating a separate set of results excluding JavaScript packages, refer to “Considerations” (page 21).

com.fasterxml.jackson.core:jackson-core

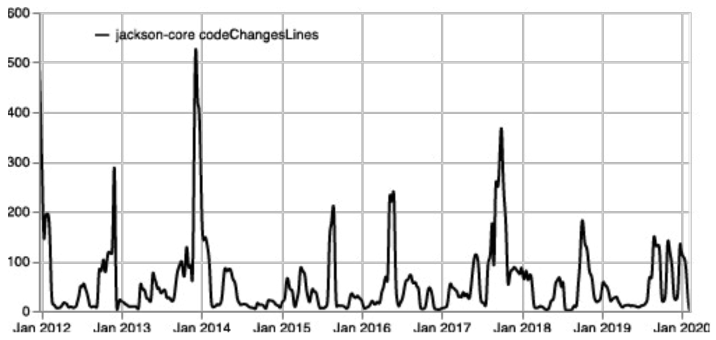
A core part of Jackson that defines Streaming API as well as basic shared abstractions.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/FasterXML/jackson-core	74,400 Lines	Authors: 7 Commiters: 6	183 total 3.52/week
As of February 7, 2020, this project has 40 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week



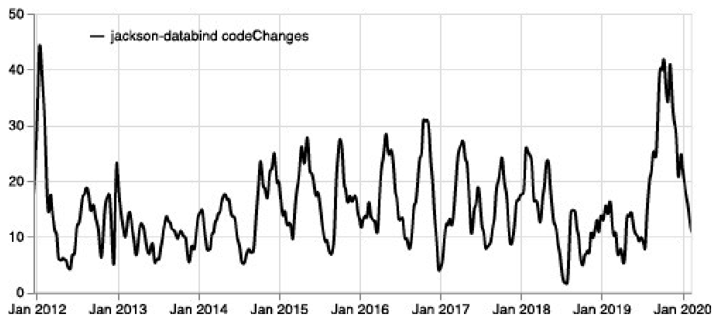
com.fasterxml.jackson.core:jackson-databind

General data-binding package for Jackson (2.x): works on streaming API (core) implementation(s).

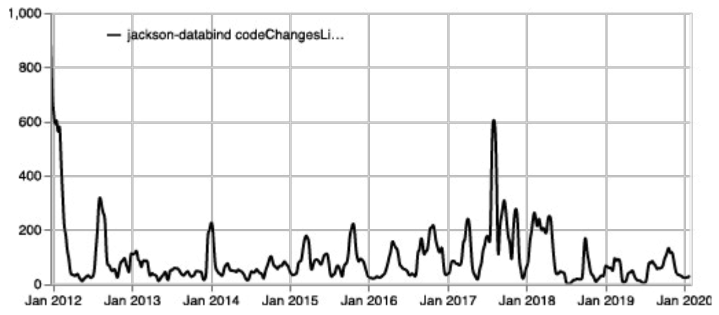
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/FasterXML/jackson-databind	74,400 Lines	Authors: 23 Commiters: 2	594 total 11.42/week

As of February 7, 2020, this project has 363 open issues on GitHub.

Code Changes (Commits)/Week



Lines of Code Added/Week

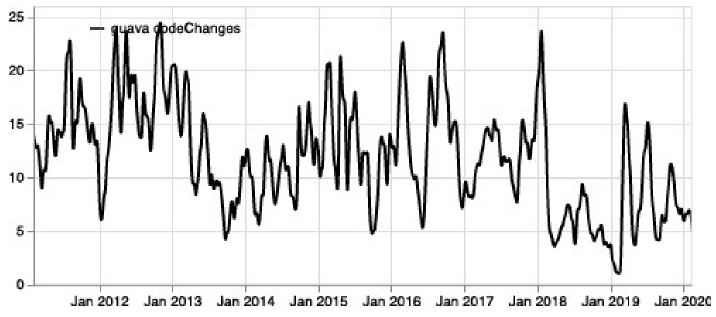


com.google.guava:guava

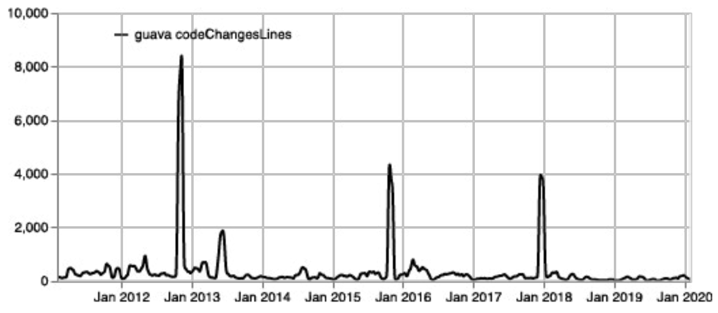
Google core libraries for Java.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/google/guava.git	1 Million Lines	Authors: 83 Commiters: 3	303 total 5.83/week
As of February 7, 2020, this project has 620 open issues on GitHub.				

Code Changes (Commits)/Week



Lines of Code Added/Week

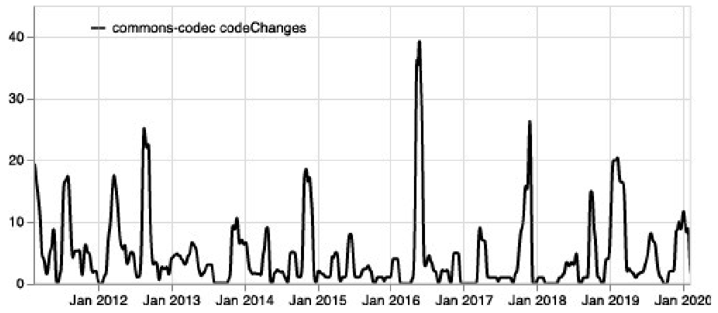


commons-codec

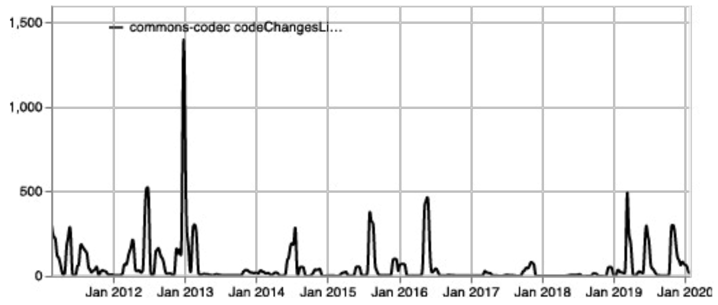
Apache Commons Codec (TM) software provides implementations of common encoders and decoders such as Base64, Hex, Phonetic and URLs.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/apache/commons-codec	51,700 Lines	Authors: 3 Commiters: 3	36 total 0.69/week
As of February 7, 2020, this project has 29 open issues on its JIRA site.				

Code Changes (Commits)/Week



Lines of Code Added/Week



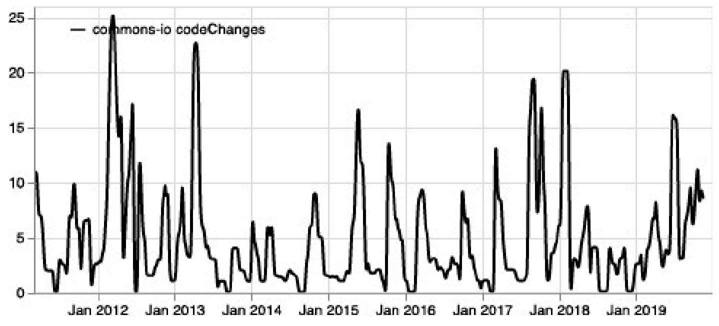
commons-io

Commons IO is a library of utilities to assist with developing IO functionality.

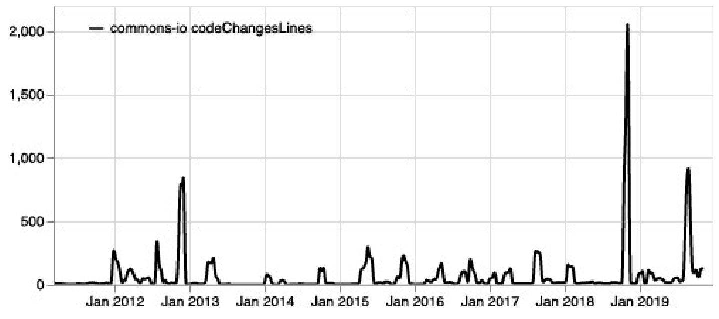
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/apache/commons-io	73,700 Lines	Authors: 10 Commiters: 6	73 total 1.40/week

As of February 7, 2020, this project has 148 open issues on its JIRA site.

Code Changes (Commits)/Week



Lines of Code Added/Week

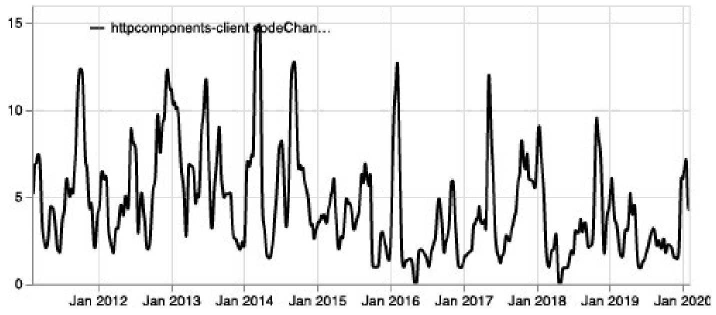


httpcomponents-client

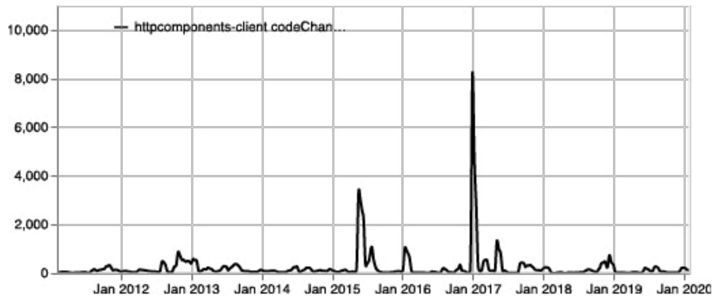
The Apache HttpComponents™ project is responsible for creating and maintaining a toolset of low level Java components focused on HTTP and associated protocols.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/apache/httpcomponents-client	121,700 Lines	Authors: 16 Commiters: 8	133 total 2.56/week
As of February 7, 2020, this project has 47 open issues on its JIRA site.				

Code Changes (Commits)/Week



Lines of Code Added/Week

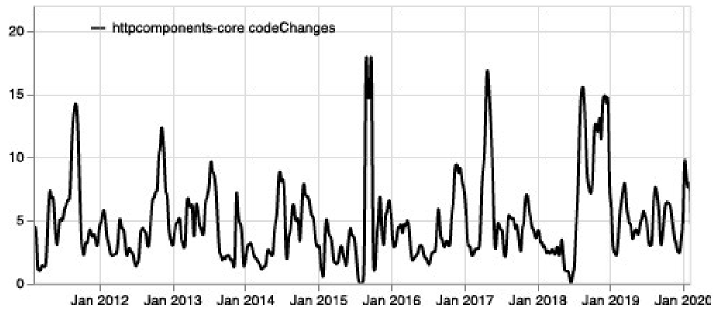


httpcomponents-core

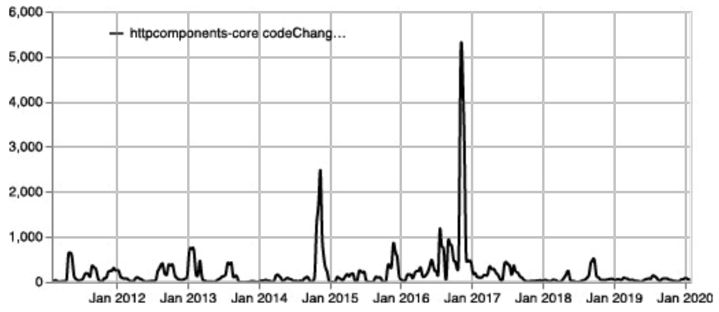
The Apache HttpComponents™ project is responsible for creating and maintaining a toolset of low level Java components focused on HTTP and associated protocols.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/apache/httpcomponents-core	130,900 Lines	Authors: 15 Commiters: 4	302 total 5.81/week
As of February 7, 2020, this project has 7 open issues on its JIRA site.				

Code Changes (Commits)/Week



Lines of Code Added/Week



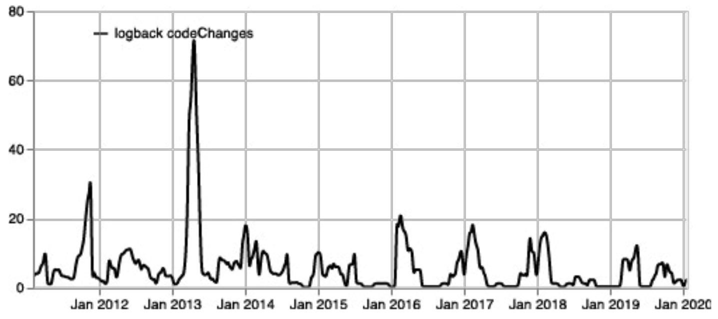
logback-core

The reliable, generic, fast and flexible logging framework for Java.

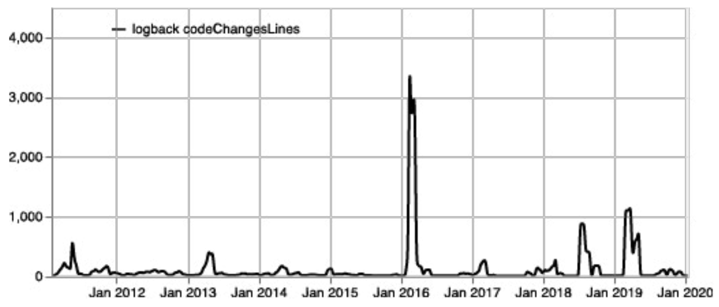
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/qos-ch/logback	154,600 Lines	Authors: 1 Commiters: 2	99 total 1.90/week

As of February 7, 2020, this project has 799 open issues on its JIRA site.

Code Changes (Commits)/Week



Lines of Code Added/Week

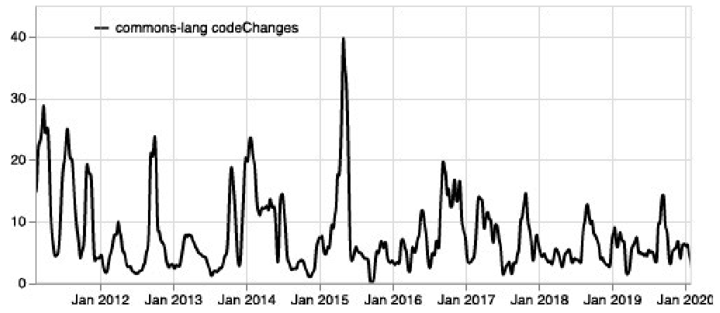


org.apache.commons:commons-lang3

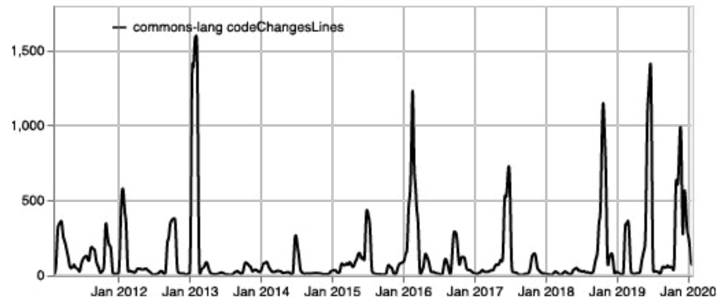
A package of Java utility classes for the classes that are in java.lang's hierarchy, or are considered to be so standard as to justify existence in java.lang.

Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/apache/commons-lang	168,300 Lines	Authors: 28 Commiters: 17	225 total 4.32/week
As of February 7, 2020, this project has 163 open issues on its JIRA site				

Code Changes (Commits)/Week



Lines of Code Added/Week



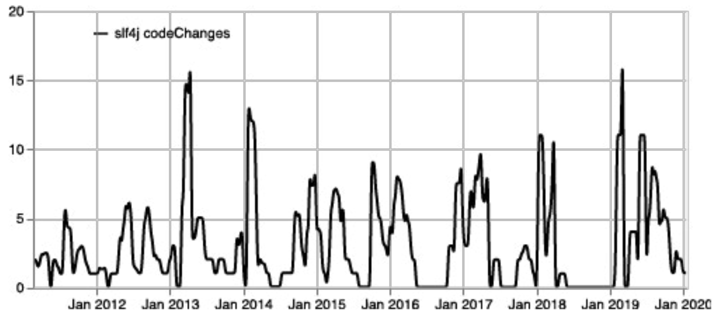
slf4j:slf4j

Simple Logging Facade for Java.

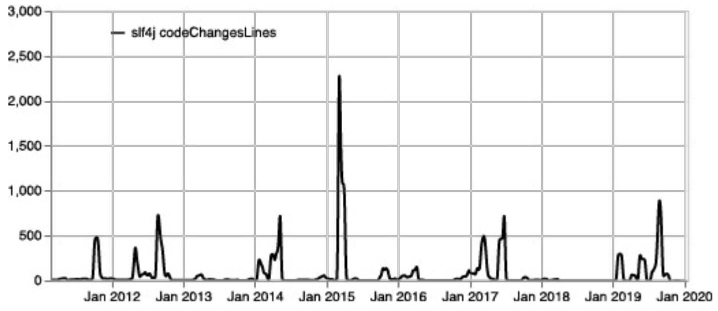
Platform	GitHub	Total Lines of Code	Active Contributors 2018	Commits 2018
maven	github.com/qos-ch/slf4j	38,400 Lines	Authors: 4 Commiters: 4	31 total 0.60/week

As of February 7, 2020, this project has 189 open issues on its JIRA site.

Code Changes (Commits)/Week



Lines of Code Added/Week



Endnotes

1. https://www.sonatype.com/hubfs/SSC/Software_Supply_Chain_Inforgraphic.pdf?t=1468857601884
2. https://www.wired.com/2016/08/open-source-won-now/?GuidesLearnMore
3. https://www.linuxfoundation.org/uncategorized/2018/08/corporate-open-source-programs-are-on-the-rise-as-shared-software-development-becomes-mainstream-for-businesses
4. https://www.coreinfrastructure.org
5. https://www.coreinfrastructure.org/programs/census-project
6. The project did use data on how popular a package was, but this was limited to installations tracked by the Debian Linux distribution and did not cast a wider net due to limited scope.
7. https://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html
8. https://time.com/3148773/report-devastating-heartbleed-flaw-was-used-in-hospital-hack/
9. Often referred to as Linus's Law, named after the creator of Linux, the maxim was formalized by Eric Raymond in his book *The Cathedral and the Bazaar* (1999).
10. https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en
11. https://ec.europa.eu/info/departments/informatics/eu-fossa-2_en
12. https://www.wired.com/story/urgent-11-ipnet-vulnerable-devices/
13. https://web.archive.org/web/20180422034612/https://energycommerce.house.gov/wp-content/uploads/2018/04/040218-Linux-Evaluation-of-OSS-Ecosystem.pdf
14. https://www.businessinsider.com/black-thursday-for-wwii-us-army-air-force-over-schweinfurt-germany-2018-10
15. https://github.com/nayafia/lemonade-stand
16. https://communitybridge.org/
17. https://bounty.github.com/
18. https://www.sciencedirect.com/science/article/pii/S0963868712000340
19. https://snyk.io/
20. https://www.synopsys.com/software-integrity/cybersecurity-research-center.html
21. The addition of new SCA and industry partners with private usage data contributions would enable CII to compile enough data to release an aggregated, de-identified dataset in the future.

22. **https://techcrunch.com/2019/01/12/how-open-source-software-took-over-the-world/**
23. **https://idalink.org/d-8777**
24. Libraries.io, a Tidelift Project licensed under CC-BY-SA 4.0, was used for two reasons: First, because it's an aggregate of many different package managers. Second, Libraries.io was used as the canonical dataset for the Census II Prototype.
25. The Z-score of a package is equal to the package's value minus the mean of the values of the list it comes from, then divided by the standard deviation of that list. This metric captures the relative importance of a package compared to other packages in its dataset. Each dataset is a sample of the greater FOSS ecosystem – larger samples are not inherently "more important" than smaller samples. Z-scores allow us to treat each distinct dataset as equally relevant to the overall result.
26. **https://arxiv.org/pdf/1709.04638.pdf**
27. **https://tomforb.es/how-much-code-is-there-in-the-python-package-index/**
28. **https://blog.thecodewhisperer.com/permalink/surviving-legacy-code-with-golden-master-and-sampling**
29. For more details on the CHAOSS Project and how to contribute, see **https://chaoss.community/about/**.
30. Values were determined using a Chrome extension "GitHub Gloc," which pulls lines of code information from GitHub's API. For more information, see **https://github.com/artem-solovev/gloc**.
31. For projects where there were no commits in 2018, the total number of contributors across the life of the project are given, as well as the dates of the most recent commit.
32. **https://www.vice.com/en_us/article/43zak3/the-internet-was-built-on-the-free-labor-of-open-source-developers-is-that-sustainable**
33. Widely used open source software contained bitcoin-stealing backdoor, Dan Goodin, ArsTechnica (November 26, 2018), **https://arstechnica.com/information-technology/2018/11/hacker-backdoors-widely-used-open-source-software-to-steal-bitcoin/**.
34. The year-long rash of supply chain attacks against open source is getting worse, Dan Goodin, Ars Technica (August 21, 2019) **https://arstechnica.com/information-technology/2019/08/the-year-long-rash-of-supply-chain-attacks-against-open-source-is-getting-worse/**.
35. Rage-quit: Coder unpublished 17 lines of JavaScript and "broke the Internet", Sean Gallagher, ArsTechnica (March 24, 2016), **https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/**.
36. Catalin Cimpanu, Developer takes down Ruby library after he finds out ICE was using it, ArsTechnica (Sep. 20, 2019) **https://www.zdnet.com/article/developer-takes-down-ruby-library-after-he-finds-out-ice-was-using-it/**.
37. **https://github.com/coreinfrastructure/best-practices-badge**.

THE LINUX FOUNDATION

The Linux Foundation promotes, protects and standardizes Linux by providing unified resources and services needed for open source to successfully compete with closed platforms.

To learn more about The Linux Foundation or our other initiatives please visit us at www.linuxfoundation.org

LISH
Laboratory for Innovation Science at Harvard

The Laboratory for Innovation Science at Harvard (LISH) is spurring the development of a science of innovation through a systematic program of solving real-world innovation challenges while simultaneously conducting rigorous scientific research and analysis.

To learn more, please visit lish.harvard.edu