Measuring Enterprise Software Supply Chain Security using Public Repositories

Dmytro Kashchuk
dmytro-kashchuk@utulsa.edu
The University of Tulsa
USA

Tyler Moore
tyler-moore@utulsa.edu
The University of Tulsa
USA

Abstract

Vulnerabilities introduced through outdated third party libraries remain a primary entry point for supply-chain attacks. To quantify this risk at enterprise scale, we examined every public GitHub repository maintained by the Forbes Global 2000. We identified 275 corporate accounts and cloned all 34 368 repositories (1.9 TB of source code). Next, we used two open source tools to identify vulnerable software libraries in the repositories. **Syft** lists every library in a project creating software bills of materials (SBOMs), while **Grype** checks each library for known vulnerabilities (CVEs or GitHub Security Advisories). 11.7% of the 3,791,834 library occurrences have at least one known vulnerability. Surprisingly, we observe active projects are almost as risky as inactive ones. Moreover, some vulnerable libraries affect many disparate projects. For example, one vulnerability in the JavaScript package `semver` appears in 805 projects developed by 89 companies, and 43 firms still publish code that contains the infamous Log4j RCE flaw. We publish our full dataset the largest of its kind to help other researchers. We also note limits of our approach and point to new rules that will soon push more companies to create and share SBOMs.

ACM Reference Format:
Dmytro Kashchuk and Tyler Moore. 2025. Measuring Enterprise Software Supply Chain Security using Public Repositories. In *Proceedings of the 2025 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3733827.3765529

1 Introduction

Many large companies produce software. This software can be used directly by the company, as well as by its customers. However, we know relatively little about the security of such software. This is one important way that information asymmetries manifest in cybersecurity [11]. One of the policy interventions designed to mitigate information asymmetry is the software bill of materials (SBOMs). SBOMs identify software libraries utilized in code, which should make it easier to identify outdated and vulnerable software packages.

In this paper, we seek to measure the security of software produced by large firms. We focus on the open source code that is developed and shared via GitHub. While such code represents only a fraction of the code firms produce, we believe it is worth studying for two reasons. First, open-source code published on enterprise repositories is important because the code can be utilized by others. Second, such code may shed light on software security practices of firms more broadly.

Lodash, a popular JavaScript library for working with objects and arrays, illustrates how libraries can introduce security vulnerabilities to firm-developed software. In July 2020, version **4.17.21** fixed a "prototype pollution" flaw (CVE-2020-8203). Prototype pollution lets an attacker inject properties into JavaScript's base object prototype, potentially leading to crashes or arbitrary code execution. Even years after the patch, many "enterprise" products still use older Lodash versions. For example Oracle telecom appliances (Session Border Controller, Session Router, Subscriber Aware Load Balancer) were still using the vulnerable library as noted in Oracle's April 2021 Critical Patch Update [23]. IBM Watson Machine Learning Community Edition included TensorBoard built on the old Lodash, risking crashes or code execution on GPU clusters if its default port was left exposed [18]. These cases demonstrate that even years after a fix remote code execution paths remain open if system owners do not update their dependencies [26].

To measure such enterprise cybersecurity risks at scale, we inspect all firms in the Forbes Global 2000, a tally of the world's largest global firms. In each case, we determine whether the firm has a public GitHub repository, finding 34 368 repositories in total. We then download all code from these repositories and extract software libraries identified through SBOM tools. We then check the libraries for associated vulnerabilities.

We report on the software security practices of enterprise software developers overall, and look for differences between projects and firms. We distinguish between projects based on activity level. Ultimately, we believe this work presents a valuable first step in measuring the state of software supply chain security at the enterprise level.

Contributions

- We conduct a large-scale study of enterprise software supply chain security.
- We introduce a way to measure firm-level cybersecurity through its public software repositories.
- We leverage SBOM tools to measure the prevalence of potentially vulnerable libraries utilized in software developed by enterprises.

2 Data Collection Methodology

Our goal is to measure software supply chain risk for code developed by large enterprises. Figure 1 visualizes the tool chain we have constructed to collect data for analysis. By identifying each

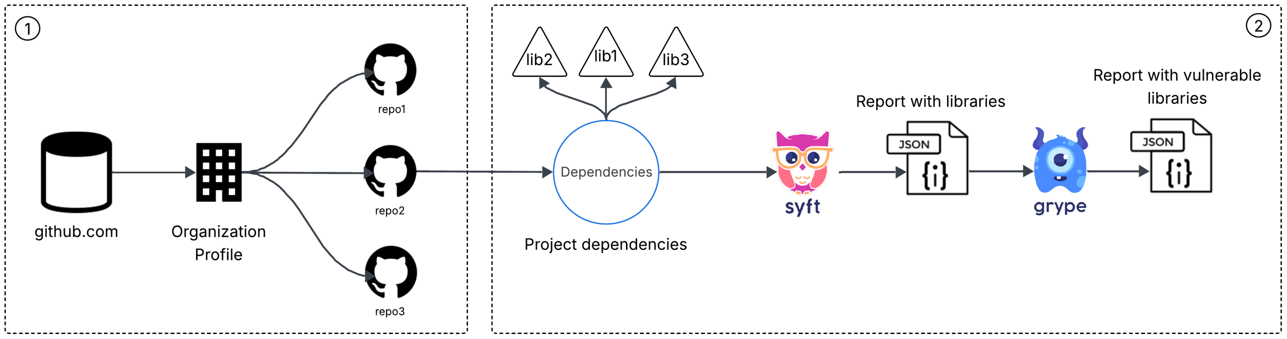

Figure 1: (1) Data Collection and (2) Data Analysis Process.

repository's core attributes, utilized libraries and vulnerabilities, we construct a clear, end to end view of the enterprise software supply chain.

2.1 Identifying Enterprise Software Repositories

We focus on companies in the Forbes Global 2000, which tracks the 2,000 largest publicly traded companies in the world. For each firm, we searched for a matching GitHub profile. We identified public profiles for 275 firms through a manual web search. We then downloaded all 34 368 public repositories for these profiles.

Once we had the official accounts, we used the GitHub API to harvest all available metadata repositories, stars, forks, followers, URLs, email addresses, etc. With our list of repositories in hand, we again tapped the GitHub API to gather detailed repository data and then cloned each project for offline analysis, totaling over 1.9 terabytes of data.

We distinguish between repositories that are actively maintained and utilized compared to those are no longer in active development. The GitHub API reports key usage metrics, notably the creation and last activity date, as well as the number of stars and forks. We deem a project to be *active* if it has more than 40 forks, more than 40 stars and has been updated within the last 30 days. 40 was selected as the cutoff for each because it is the median value for forks among all repositories. Using this criteria, 3 324 of the 34 368 repositories are considered active. Moreover, 117 of the 275 profiles contain at least one active repository.

We observed that, unsurprisingly, many of the most popular public GitHub profiles belong to technology companies. This makes sense, given that they are more likely to create software that others can utilize. Therefore, we sought to distinguish between tech firms and others, so that we might observe differences in their security practices.

To identify tech companies, we checked each companies Standard Industry Classification (SIC) codes. We marked a company as "tech" if its code starts with 35 (computers and related equipment), 36 (telecoms and electronics) or 38 (instruments and controls). We also added companies in semiconductors (SIC 3674) and telephone services (SIC 4813). For software and IT Services, we included all codes from 7371 to 7379. This approach provides a method for distinguishing technology companies from non-technology companies within the dataset.

2.2 Identifying Software Libraries

For each repository we are interested in the software libraries utilized, as well as its version. We utilized Anchore Syft, a command-line tool that scans codebases (or container images) and generates a Software Bill of Materials (SBOM) listing every component and its version. Syft reads operating system package databases for Debian or Ubuntu, rpm-based systemsand for Alpine Linux [5, 6].

Table 1: Collected Data.

Metric	Value
Profiles analyzed	275
Repositories analyzed	34,368
Libraries analyzed	3,791,834
GHSA found	444,439 (11.72%)

Syft also scans language manifest and lock files such as, package-lock.json, yarn.lock, requirements.txt, Pipfile.lock, go.mod, go.sum, Cargo.toml and Cargo.lock to list declared dependencies [7, 25]. Java archives (jar, war, ear, par, sar, nar, native image) and their metadata files such as pom.xml are inspected to describe Java components [10]. The continuous integration descriptors in the github/workflows directory are cataloged to record the usage of GitHub actions [16]. Finally, Syft can start with many types of source, container images, directories, or compressed archives and automatically pick the right catalogers for each [25].

2.3 Identifying Vulnerabilities in Libraries

Once we had that SBOM file, we fed it into Grype, another tool from Anchore that compares each listed package against vulnerability databases (CVE and GHSA) and flags any that are outdated or carry known security issues. Grype then produces a new report highlighting exactly which libraries need attention.

It identifies which parts of software have known risks by matching versions against a vulnerability database. This helps developers find and fix security issues before they can be exploited. Grype can start from many kinds of input. It accepts SBOM files in Syft JSON, SPDX, or CycloneDX format, or it can catalog packages itself

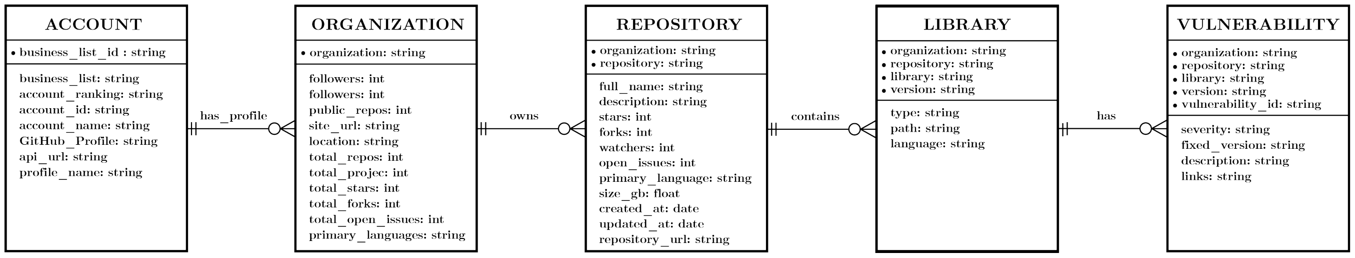

Figure 2: Database schema.

from container images, OCI archives, Singularity images, regular directoriesand single files [9]. Grype's vulnerability database is generated from public feeds such as Alpine SecDB, Amazon ALAS, Debian CVE Tracker, GitHub Security Advisories, the National Vulnerability Database (NVD), Red Hat and SUSE OVAL data, Ubuntu Security, Wolfi SecDB and others [8].

We recognize that Syft's identification of software libraries and Grype's detection of vulnerabilities present in those libraries is incomplete. Consequently, the subsequent analysis can be best interpreted as a lower bound for the number of vulnerabilities present in the enterprise software.

2.4 Final dataset

As a final result, we built an integrated dataset (outlined in Figure 2) to perform high-level analysis. For each project we have: GitHub metadata for profiles and repositories (creation date, last commit, stars, forks, languages), the SBOM generated with Syft, which lists all dependencies and their versions found in the project's manifests and the list of known vulnerabilities (CVE and GHSA) identified by Grype for each component, with CVSS severity, disclosure date with a link to the patch. This information is organized into five main tables (figure 2), linked by foreign keys that allow cross analysis at the project, company and industry levels. This is a nice database of software supply chain security produced by large enterprises and is a key starting point for future comparative and longitudinal research in this critical area.

3 Analysis

We now discuss the data collected from repositories.

3.1 Profile coverage and utilization

We first report summary statistics on profile prevalence and repository coverage. In total, 275 firms out of the Forbes Global 2000 have public GitHub profiles (14%). Of these profiles, 31 have no public repositories. These profiles maintain a median of 22 repositories (mean 125). However, the top ten firms account for 50.70% (17 431 out of 34 368) of all repositories and 72.08% (2 396 out of 3 324) of all active repositories.

Table 2 lists the top 20 firms with the most active repositories, sorted accordingly. It also lists the median number of libraries invoked by each repository, the number and percentage of repositories loading at least one vulnerable library, and the percentage of a firm's active repositories with at least one vulnerability present. The bottom row reports the median figures for all firms with at least ten repositories present. We can see, for example, that while Microsoft has the largest absolute number of vulnerable repositories, as a proportion they fare slightly above average (26% vs 31%). We also observe that active repositories tend to have higher vulnerability rates, which is true for all organizations.

Table 2: Repository characteristics for top 20 organizations with the most active repositories.

Organization	Total Repos	% Active	Median Libraries per Repo	# Vuln. Repos	% Repos w/ Vuln.	% Active Repos w/ Vuln.
Microsoft	7004	14	10	1814	26	74
Alphabet	2786	20	7	621	22	68
Alibaba Group	455	36	17	207	45	82
NVIDIA	587	24	9	115	20	57
Tencent Holdings	217	59	14	76	35	64
Meta Platforms	150	65	9	45	30	80
Intel	1318	7	7	177	13	59
Unity Software	782	12	5	112	14	23
Cloudflare	504	17	4	181	36	96
IBM	3628	2	15	1156	32	63
Shopify	1122	7	14	402	36	93
Apple	333	22	7	71	21	70
Netflix	232	27	5	56	24	87
Oracle	296	16	16	78	26	66
Coinbase	163	25	49	80	49	80
Salesforce.com	404	10	11	173	43	70
Spotify Technology	278	14	12	121	44	82
Uber	166	22	12.5	66	40	92
Adobe	1030	3	30	450	44	66
SAP	296	11	51	138	47	67
Median Organization	143	6	12	40	31	83

Table 3: Top 3 Ecosystems per Organization.

Org	1st	2nd	3rd
Microsoft	npm	python	github-action
Alphabet	npm	rust-crate	python
Alibaba Group	npm	go-module	java-archive
NVIDIA	go-module	rust-crate	npm
Tencent Holdings	npm	python	rust-crate

3.2 Libraries and vulnerabilities

For each repository we obtained the dependency name, version, and the Syft artifact *type*, which we refer to as the *ecosystem*. This set includes both true package managers (npm, python) and other ecosystems detected by Syft, such as GitHub Actions (github-action) or Java archives (java-archive). Consequently, Table 3 reflects the most prevalent *Syft ecosystems*, not only package managers, used by organizations.

Of the 28 different ecosystems identified, npm is most widely used across all organizations.

In total, we identified 3,791,834 libraries (median 13 and mean 110 libraries per repo) across all repositories this is the overall number of library usages. Among these, around 106,503 represent distinct libraries and 324,631 are unique when considering both the library name and version ([library, version]).

We found that 444,439 libraries (11.72%) are affected by at least one known CVE. As shown in Table 4, a total of 1,405 repositories (221 of which are active) contain at least one library with a HIGH severity vulnerability. In total there are 9,426 vulnerable repos with median of 6 vulnerabilities per repo.

Table 4: Vulnerability Severity Counts per Repo

Severity	Active Repos	Inactive Repos
High	221	1,184
Medium	247	1,273
Low	212	893

How do vulnerability rates vary by firm? Table 5 shows the firm profiles with the lowest and highest vulnerability rates. The left columns show the best and worst performing technology companies, while the right columns show non-technology companies. We included any firm profiles with at least ten repositories.

The worst overall performer among tech companies is Vodafone, in which 57.9% of its repositories invoked libraries with known active vulnerabilities. Over half of Zebra Technologies and 47% Intuit repositories also had vulnerabilities, which is well above the median rate of 27.9% for technology companies with at least ten repositories. Among highly popular profiles, Spotify fared worst, with 43.5% of its repositories having at least one vulnerable library present.

On the other end of the spectrum, the three best-performing tech companies Cadence Design, Quanta Computer, Info Edge India had zero known vulnerabilities in repositories introduced by software libraries. Among the most popular profiles, Microchip Technology has a notably low vulnerability rate at 3.8%.

Moving to the non-technology companies, we observe that its worst performers outpaced those in the technology space. Citigroup and ResMed had vulnerabilities in 75.0% of their repositories. But there are also many secure non-tech firms, with two having no observed vulnerabilities. The median vulnerability rate for repositories among non-tech firms with more than ten repositories was higher than for tech firms at 32.3%, but only moderately so.

What can we take away from this analysis? First, there is wide dispersion in security performance among firms based on this metric. Second, the metric itself applies to many different types of firms, which suggests it could be a promising measure to track over time.

3.3 Comparing vulnerability rates of active and inactive repositories

Table 6: Vulnerability Summary for Active vs Inactive Repositories.

	Vuln Repos	Total Repos	% Vuln
Active repositories	1,032	3,324	31.05%
Inactive repositories	8,394	31,044	27.03%

Table 6 shows that 31.05% of *active* repositories and 27.03% of *inactive* repositories contain at least one vulnerability. We had expected inactive repositories to have more vulnerabilities, since they are more likely to run outdated software packages.

A plausible explanation is that many enterprises publish code on GitHub to signal openness or innovation and then largely ignore it. Empirical studies of "abandoned repositories" back up this view. Miller *et al.* observe that 15 % of the most downloaded `npm` packages are abandoned within six years and that 82 % of the affected projects never respond to the issue at all [21]. Avelino *et al.* deepen the picture by analysing 1 932 of the most starred GitHub projects across six languages. They introduce the notion of a project's *truck factor* the smallest set of developers whose loss would stall the project and classify a repository as abandoned when all those key developers cease committing for at least twelve months. In their dataset, 16 % of projects reached this point. Fewer than half of these ever recovered and when recovery occurred it typically hinged on a single newcomer joining within the first year. Most abandonments (59 %) struck during a project's first two years and projects that did bounce back produced roughly four times as many commits after the event as those that did not [12]. A follow up survey found that new maintainers step in mainly because they personally rely on the software, while administrative hurdles such as obtaining push rights are a bigger barrier than writing code. Taken together, these findings show that the label *active* in our own dataset merely indicates recent surface activity; it does *not* prove that security patches are applied or that a healthy maintainer team is in place. Without automated dependency updates and broader maintainer coverage, even high-profile repositories quickly accumulate outdated and vulnerable libraries —- hence the surprisingly similar vulnerability rate for active and inactive projects in Table 6.

Table 5: Firms with the most and least vulnerable percentage of repositories with vulnerabilities, split by technology companies and others.

Technology Companies				*Non-Technology Companies*			
		# Repositories				# Repositories	
Company	**Vuln. %**	**Vuln.**	**Non-Vuln.**	**Company**	**Vuln. %**	**Vuln.**	**Non-Vuln.**
Top 10 Most Vulnerable Firm Repositories							
Vodafone	57.9%	11	8	Citigroup	75.0%	24	8
Zebra Technologies	53.8%	7	6	ResMed	75.0%	9	3
Intuit	47.5%	66	73	US Foods	70.0%	7	3
SAP	46.6%	138	158	Toll Brothers	70.0%	7	3
Swisscom	46.5%	101	116	Farfetch	68.8%	11	5
Synopsys	44.4%	12	15	ZTO Express (Cayman)	61.1%	11	7
Spotify Technology	43.5%	121	157	Societe Generale	61.0%	25	16
Salesforce.com	42.8%	173	231	Goldman Sachs Group	56.5%	39	30
VMware	40.1%	63	94	Kohl's	54.8%	17	14
Naver	37.5%	101	168	General Motors	54.7%	23	19
Top 10 Least Vulnerable Firm Repositories (reversed order)							
Analog Devices	10.4%	31	266	ThyssenKrupp Group	14.3%	5	30
Keysight Technologies	10.0%	6	54	Walmart	13.9%	21	130
Broadcom	7.1%	7	91	Novartis	12.6%	12	83
Marvell Technology	7.0%	4	53	Air France–KLM	10.5%	2	17
Skyworks Solutions	5.4%	2	35	AGCO	8.3%	4	44
Microchip Technology	3.8%	15	376	The Boeing Company	7.7%	3	36
Legrand	3.0%	1	32	Illumina	5.4%	3	53
Info Edge India	0.0%	0	11	Avnet	4.5%	6	126
Cadence Design	0.0%	0	89	Bristol Myers Squibb	0.0%	0	33
Quanta Computer	0.0%	0	25	Chevron	0.0%	0	22

3.4 Measuring systemic risk of library vulnerabilities

We are particulary interested in whether certain vulnerabilities in a software library could simultaneously affect the software utilized by many firms. If so, this would indicate that software supply chain insecurity presents a systemic cyber risk. To check, we studied the most prevalent vulnerabilities present across all analyzed repositories and organizations. Table 7 presents the top ten GitHub Security Advisories (GHSAs), detailing the affected packages, severity ratings, total appearance counts and the number of distinct repositories and organizations impacted. On the one hand, the most commonly occurring vulnerabilities only affect a relatively small number of repositories. The most commonly occurring vulnerability affecting the brace-expansion library affected 1 225 repositories, just 3% of the total across all firms.

On the other hand, many of these vulnerabilities do in fact affect a large share of firms. The same brace-expansion vulnerability affected at least one repository for 39% of firms. This does suggest that many of the most popular vulnerabilities may pose a systemic risk given the wide adoption across firms.

For example, **GHSA-c2qf-rxjj-qqgw** in the *semver* package is classified as *High* severity and corresponds to a Regular Expression Denial of Service (ReDoS) flaw where untrusted input passed to the constructor can trigger catastrophic backtracking and block the event loop. We observed this advisory 2 324 times across 805 unique repositories owned by 89 different organizations. Again, the share of repositories affected is small (2%), but many organizations are affected (32.4%).

While not in the top 10, we also examined the presence of one of the most critical and widespread vulnerabilities ever discovered the Log4j remote code execution (RCE) flaw within corporate repositories. Our analysis revealed that 151 unique repositories and 43 unique organizations remain affected by this vulnerability.

Table 7: Vulnerabilities appearing in the most repositories.

Vulnerability ID	Library	Severity	repos	orgs
GHSA-v6h2-p8h4-qcjw	brace-expansion	Low	1225	108
GHSA-c2qf-rxjj-qqgw	semver	High	805	89
GHSA-968p-4wvh-cqc8	@babel/helpers	Moderate	681	84
GHSA-3xgq-45jj-v275	cross-spawn	High	717	86
GHSA-9wv6-86v2-598j	path-to-regexp	High	665	83
GHSA-grv7-fg5c-xmjg	braces	High	689	85
GHSA-952p-6rrq-rcjv	micromatch	Moderate	686	86
GHSA-hrpp-h998-j3pp	qs	High	574	77
GHSA-3h5v-q93c-6h6q	ws	High	527	75
GHSA-35jh-r3h4-6jhm	lodash	High	517	76

Table 8 lists the libraries with the highest number of GitHub Security Advisories (GHSAs) detected and shows how the vulnerable versions are distributed across repositories and organizations. This is an alternative way of measuring high-impact vulnerabilities, this time by the frequency of occurrence of vulnerabilities in the packages themselves. Note that the three vulnerabilities attracting the most distinct vulnerabilities are variations of the popular tensorflow libraries. Once again, these vulnerabilities affect a small number of repositories but affect a higher share of organizations at least once.

4 Related Work

Several recent studies connect directly to our effort to measure the security of enterprise software supply chains. Lazarine et al. [19] built a *fork graph* of large tech GitHub projects and ran static-analysis tools on every fork. They showed that many forks keep exploitable code long after the upstream patch, highlighting how

Table 8: Top 10 Most Vulnerable Packages and Their Adoption.

Package	Vulnerabilities	Repositories	Organizations
tensorflow	423	183	41
tensorflow-gpu	418	25	9
tensorflow-cpu	399	5	3
ImageMagick	268	1	1
jenkins-core	201	3	3
FFmpeg	195	2	2
go	125	4	3
django	118	65	32
stdlib	117	80	35
FreeType	83	1	1

risk spreads through the fork network, an effect we also observe at company scale. Williams et al. [29] surveyed supply chain attack paths across code dependencies, build systems, and human factors. They outlined research needs such as better metrics, stronger standards (SSDF, SLSA), and new challenges introduced by large language models (LLMs) in the supply chain.

One important limitation of our study is its reliance on imperfect tools for generating SBOMs and linking them to vulnerabilities. O'Donoghue et al. [24] sheds light on the resulting measurement error. They compared SBOMs generated by Syft and Trivy (CycloneDX vs. SPDX) and scanned them with Grype, Trivy, and CVE bin tool. Simply changing the SBOM tool or format produced large swings in the vulnerability count, underscoring the measurement challenges we face when scanning thousands of enterprise repositories. It is worth noting that Syft and Grype performed best among tools evaluated. In a similar effort, Yu et al. performed a largescale differential analysis of four popular metadata based SBOM tools (Trivy, Syft, Microsoft SBOM Tool, and GitHub Dependency Graph) and found that none produces fully accurate manifests: all exhibit inconsistent dependency extraction, omissions, and even parser confusion vulnerabilities that can hide malicious or vulnerable packages [30].

Nocera et al. [22] mined GitHub for projects that already publish SBOMs, finding only 186, fewer than half ship the SBOM in official releases. This low adoption motivates our choice to generate SBOMs automatically so that vulnerability analysis covers all repositories, not just the rare ones with a pre-existing SBOM. However, simply making security information available does not guarantee that it will be critically assessed or properly used. Less experienced developers often feel safe when relying on projects released by large companies such as Google. A very interesting example (although not entirely related to our study) comes from a survey of 214 Google account holders: Balasch et al. observed that 67% of participants had at least one third party application with broad API access, and 79% rarely or never reviewed these permissions, illustrating a trust transference effect toward anything associated with Google [13].

One of the biggest limitations of SBOM-based vulnerability detection is the assumption that all included libraries are equally risky. Recent work on code coverage and reachability analysis shows that this may significantly overestimate actual risk. The work from [31] demonstrates that many flagged vulnerabilities exist in code that is never executed in practice. This "reachability gap" means our 11.7% vulnerability rate may overstate actual exploitable risk, as the vulnerabilities might never be called.

A comparable security gap appears on GitHub. Fischer *et al.* analyzed more than 50 000 repositories and found that dependency alert banners increased manual patching by about 149%, while enabling automated security updates added a further 32%. By contrast, the newer CodeQL code scanning feature reduced critical vulnerabilities by only 2% [17].

These findings show that neither a project's brand nor platform level warnings are enough to guarantee safety. Continuous dependency maintenance and independent verification remain essential for keeping real world projects secure.

4.1 Regulations

Currently, only certain categories of companies are subject to SBOM related laws, such as those operating in government contracting, healthcare, critical infrastructure, financial services, and IoT device manufacturing, but SBOMs are becoming increasingly important over time, and soon we will see all software companies producing SBOMs. In this section, we list all current and upcoming regulations.

EU Cyber Resilience Act (CRA). This EU rule, published in November 2024 and coming into effect on 11 December 2027, states that anyone who makes products with digital parts (hardware or software) must create and keep a machine readable SBOM in their technical documents [3].

United States Executive Order 14028. In May 2021, Executive Order 14028 states that every U.S. federal agencies must require SBOMs from suppliers of critical software. A proposed change to the Federal Acquisition Regulation would extend this requirement to all government ICT contractors [1][4].

PCI DSS v4.0. From 31 March 2025, PCI DSS v4.0 requires any organization that manages payment card data to produce an accurate list of their software components. [2].

US FDA Cybersecurity for Medical Devices. Since 1 October 2025, Section 524B of the FD&C Act and the FDA final guidance demand a machine-readable SBOM for every connected medical device submission [14].

These rules show that, although not every company has to produce an SBOM right now, more industries such as IoT devices and finance will likely face similar or stricter requirements soon.

5 Conclusion

Even though many open source repositories are active and used by thousands of developers every day, they continue to ship outdated dependencies that contain high risk CVEs, exposing both end users and at times, the organizations behind the software.

We have introduced a way to measure firm-level cybersecurity through its public software repositories. We believe the approach shows promise and merits further study, especially given the emphasis policymakers in many jurisdictions have placed on utilizing SBOMs to measure and improve software security.

We have only begun to scratch the surface on the types of analysis that can be conducted. In future, we aim to conduct econometric analysis to search for causal relationships between firm software security practices and observable outcomes. One major limitation of our study is the correctness and completeness of SBOM generation itself. Because we rely on Syft and Grype for SBOM creation and vulnerability flagging, this tool level variability means we may undercount real risks (missed dependencies) and overcount false ones, introducing measurement bias into our enterprise scale supply chain assessment [24, 30]. Another challenge that we defer to future work is to investigate the root causes for why so many repositories load vulnerable libraries, even after patched versions are available.

What are the implications for organizations? First, they should recognize that public source-code repositories reflect on the company's image. They should pay closer attention to the quality of the code on these repositories, since they are associated with their organization and are open source for anyone to inspect. Second, organizations should beware the technical debt incurred by launching open-source projects. Maintaining and updating third-party libraries is a crucial task, but one that is frequently overlooked based upon the data presented here. Organizations should proactively determine whether to sunset projects that are not actively used. It would be unfortunate if firms responded to our findings by making their repositories private. Rather, we hope firms begin to recognize that pushing code to public repositories creates additional responsibilities over the project's lifecycle.

SBOM tools can help identify the presence of outdated libraries and offer guidance on what needs to be patched. The methodology we followed to identify potential vulnerabilities is completely open source and easily replicable. One possible avenue for future research could be to notify companies of vulnerable code. However, the research on the effectiveness of notification strategies suggest that success would be limited [15, 20, 27, 28]. Ultimately, this is not an efficient or scalable approach. Instead, we hope to elevate awareness so that firm behavior can change by incorporating the open-source tooling into their own vulnerability-management processes.

Acknowledgments

We gratefully acknowledge support from the US National Science Foundation Award No. 2452738. Kashchuk acknowledges support from Tulsa Innovation Labs via the Cyber Fellows Initiative.

References

[1] 2021. Executive Order 14028 Improving the Nation's Cybersecurity. The White House, May 2021.

[2] 2022. Payment Card Industry Data Security Standard v4.0. Effective Mar. 31, 2025.

[3] 2022. Regulation (EU) 2022/2556 on Cyber Resilience. Published Nov. 2024, applicable from Dec. 11, 2027.

[4] 2024. Proposed FAR Clause 52.239-ZZ—Software Bill of Materials.

[5] Anchore. 2022. GitHub. https://github.com/anchore/syft/issues/733

[6] Anchore. 2023. GitHub. https://github.com/anchore/syft/issues/1550

[7] Anchore. 2024. Package Cataloger Selection. Syft Wiki on GitHub. https://github.com/anchore/syft/wiki/package-cataloger-selection

[8] Anchore. 2025. Grype README (database location and data feeds). GitHub Repository. https://github.com/anchore/grype

[9] Anchore. 2025. Grype README (supported sources and SBOM formats). GitHub Repository. https://github.com/anchore/grype

[10] Anchore. 2025. Syft README (supported package types). GitHub Repository. https://github.com/anchore/syft

[11] Ross Anderson and Tyler Moore. 2006. The Economics of Information Security. *Science* 314, 5799 (2006), 610–613. https://doi.org/10.1126/science.1130992

[12] Guilherme Avelino, Eleni Constantinou, Marco Túlio Valente, and Alexander Serebrenik. 2019. On the Abandonment and Survival of Open Source Projects: An Empirical Investigation. *CoRR* abs/1906.08058 (2019). arXiv:1906.08058 http://arxiv.org/abs/1906.08058

[13] David G. Balash, Xiaoyuan Wu, Miles Grant, Irwin Reyes, and Adam J. Aviv. 2022. Security and Privacy Perceptions of Third-Party Application Access for Google Accounts. (2022). https://www.usenix.org/system/files/sec22-balash.pdf

[14] United States Congress. 2022. Federal Food, Drug, and Cosmetic Act, Section 524B: Ensuring Cybersecurity of Medical Devices. 21 U.S.C. § 360e-4, added by Pub. L. 117-328 (Food and Drug Omnibus Reform Act of 2022). Enacted 29 December 2022; FDA begins enforcing SBOM provision on 1 October 2025.

[15] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (Vancouver, BC, Canada) *(IMC '14)*. Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755

[16] FirelightFlagboy. 2023. Support cataloging GitHub workflow and GitHub action usages (Issue #1896). GitHub Issue. https://github.com/anchore/syft/issues/1896

[17] Felix Fischer, Jonas Höbenreich, and Jens Grossklags. 2023. The Effectiveness of Security Interventions on GitHub. (2023). https://arxiv.org/abs/2309.04833

[18] IBM PSIRT. 2020. *Security Bulletin: TensorBoard Node.js Lodash module vulnerable to prototype pollution.* https://www.ibm.com/support/pages/security-bulletin-wml-ce-tensorboard-nodejs-lodash-module-vulnerable-denial-service-caused-prototype-pollution-attack IBM WML CE 1.6.2 / 1.7.0, published 17 July 2020.

[19] Ben Lazarine, Zhong Zhang, Agrim Sachdeva, Sagar Samtani, and Hongyi Zhu. 2022. Exploring the Propagation of Vulnerabilities from GitHub Repositories Hosted by Major Technology Organizations. In *Cyber Security Experimentation and Test Workshop (CSET).* https://doi.org/10.1145/3546096.3546114

[20] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. 2016. You've Got Vulnerability: Exploring Effective Vulnerability Notifications. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1033–1050. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/li

[21] Courtney Miller, Mahmoud Jahanshahi, Audris Mockus, Bogdan Vasilescu, and Christian Kästner. 2025. Understanding the Response to Open-Source Dependency Abandonment in the npm Ecosystem. In *Proceedings of the 47th International Conference on Software Engineering (ICSE '25)*. IEEE / ACM, Ottawa, Canada, 1–12. https://doi.org/10.1109/ICSE55347.2025.00004

[22] Simone Nocera, Simone Romano, Massimiliano Di Penta, Rocco Francese, and Giuseppe Scanniello. 2023. Software Bill of Materials Adoption: A Mining Study from GitHub. Technical report, University of Sannio. (2023). https://hdl.handle.net/20.500.12070/67170

[23] Oracle Corp. 2021. *Oracle Critical Patch Update Advisory – April 2021.* https://www.oracle.com/security-alerts/cpuapr2021.html Risk Matrix: multiple Communications components affected by Lodash CVE-2020-8203.

[24] Eric O'Donoghue, Brittany Boles, Clemente Izurieta, and Ann Marie Reinhold. 2024. Impacts of Software Bill of Materials (SBOM) Generation on Vulnerability Detection. (2024).

[25] Alan Pope. 2025. How Syft Scans Software to Generate SBOMs. Anchore Blog. https://anchore.com/blog/how-syft-scans-software-to-generate-sboms/

[26] Snyk Security Research. 2020. *Prototype Pollution in Lodash (CVE-2020-8203).* https://security.snyk.io/vuln/SNYK-JS-LODASH-567746 Disclosure and proof of concept.

[27] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. 2016. Hey, You Have a Problem: On the Feasibility of Large-Scale Web Vulnerability Notification. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1015–1032. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/stock

[28] Marie Vasek and Tyler Moore. 2012. Do malware reports expedite cleanup? An experimental study. In *Proceedings of the 5th USENIX Workshop on Cyber Security Experimentation and Test* (Bellevue, WA) *(CSET'12)*. USENIX Association, Berkeley, CA, USA. https://tylermoore.utulsa.edu/cset12.pdf

[29] Laurie Williams, Giacomo Benedetti, Sivana Hamer, Ranindya Paramitha, Imranur Rahman, Mahzabin Tamanna, Greg Tystahl, Nusrat Zahan, Patrick Morrison, Yasemin Acar, Michel Cukier, Christian Kästner, Alexandros Kapravelos, Dominik Wermke, and William Enck. [n. d.]. Research Directions in Software Supply Chain Security. *ACM Transactions on Software Engineering and Methodology* ([n. d.]).

[30] Sheng Yu, Wei Song, Xunchao Hu, and Heng Yin. 2024. On the Correctness of Metadata-Based SBOM Generation: A Differential Analysis Approach. (2024). https://doi.org/10.1109/DSN58291.2024.00018

[31] Yunze Zhao, Yuchen Zhang, Dan Chacko, and Justin Cappos. 2024. CovSBOM: Enhancing Software Bill of Materials with Integrated Code Coverage Analysis. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Tsukuba, Japan, 228–237. https://doi.org/10.1109/ISSRE62328.2024.00031