Software Composition Analysis and Supply Chain Security in Apache Projects: an Empirical Study

Sabato Nocera*, Sira Vegas†, Giuseppe Scanniello*, and Natalia Juristo†
*University of Salerno, Fisciano, Italy
†Universidad Politécnica de Madrid, Madrid, Spain
email: snocera@unisa.it, sira.vegas@upm.es, gscanniello@unisa.it, natalia@fi.upm.es

***Abstract*—A software supply chain consists of anything needed to develop and deliver a software project, including (third-party) components. Software Composition Analysis (SCA) allows for managing the security of software supply chains by identifying such components and their (security) vulnerabilities. The main goal of the empirical study presented in this paper is to investigate the effects of adopting/using over time an SCA tool like OWASP Dependency-Check (OWASP DC) in the context of the security of the software supply chain. To this end, following a cohort design, we analyzed the vulnerabilities affecting the components of the open-source (OS) Java Maven projects owned by the Apache Software Foundation (ASF) and publicly hosted on GitHub. These projects could adopt (or not) OWASP DC. The results indicate that the adoption of OWASP DC appears to be causing a significant reduction in the overall number/score of vulnerabilities, including those with a high Common Vulnerability Scoring System (CVSS) severity level. The use of OWASP DC also increased the vulnerabilities with a low severity level. Our results seem to encourage practitioners to adopt SCA to improve the security of their software supply chains.**

***Index Terms*—Cohort Study, Empirical Study, Software Composition Analysis, Software Supply Chain Security, Software Vulnerabilities**

I. Introduction

The software supply chain refers to the entire process and ecosystem involved in the development, sourcing, building, and distribution of software. It includes all components, dependencies, tools, and third-party libraries (simply components from here onwards) that contribute to a software product, from initial development to deployment and ongoing maintenance. Nowadays, software projects are typically made up of several components, which can be affected by a range of (security) vulnerabilities. Software supply chain attacks involve the exploitation of vulnerabilities in components on which a software project depends. In the past few years, security regulations for the software supply chain have been enforced by organizations all over the world, including the United States Government [29] and the European Union [11], [12]. Nevertheless, developers often do not review component security: they usually keep components outdated, are unaware of the vulnerabilities within them, and consider updates as an extra effort (and, therefore, not a priority) [34].

To manage the software supply chain security, developers can leverage Software Composition Analysis (SCA) [57]. SCA is an application security methodology that allows tracking and analyzing the components on which a software project depends; it also produces information about disclosed vulnerabilities within components and possible exploits, as well as information about licenses and deprecated dependencies [55]. SCA is performed through automatic tools, such as those integrated into development pipelines [40]. Although past studies investigated the vulnerability detection capability of these tools (*e.g.,* [27], [64]), there are no studies that compare the effect of using SCA tools over time with respect to not using them at all for software supply chain security. This is also to say that there is a lack of evidence proving that SCA tools effectively enhance the security of software supply chains when adopted in the development process of real open-source and commercial projects. From a pragmatic viewpoint, does using such a tool over time significantly reduce the vulnerabilities in the components these projects depend on?

In this paper, as a first step in bridging this research gap, we present the results of an empirical study analyzing the vulnerabilities affecting the components of Open-Source (OS) Java Maven projects owned by the Apache Software Foundation (ASF) and hosted on GitHub. These projects could adopt (or not) OWASP Dependency-Check (OWASP DC) [19] as an SCA tool. To gather empirical evidence on whether adopting OWASP DC may cause a reduction in the vulnerabilities affecting the components of the software projects, we leveraged a cohort design. This kind of design for empirical studies is well-established in research fields like epidemiology but not in Software Engineering (SE) [52]. Cohort studies allow for investigating cause-effect relationships when controlled experiments cannot be conducted [47], [52]. For example, when the goal is to study how different factors or interventions impact outcomes over time.

The results suggest that the use of OWASP DC seems to be causing a significant reduction in the number of vulnerabilities affecting project components. A significant effect was also observed when studying the score associated with vulnerabilities. The score of a vulnerability indicates its severity level based on the Common Vulnerability Scoring System (CVSS). We observed that the use of OWASP DC caused a significant reduction in the vulnerabilities with a high severity level. Despite that, the use of OWASP DC also caused an increase in the vulnerabilities with a low severity level. Such a difference could be attributed to the smaller impact of vulnerabilities with a low severity level and, therefore, developers' choices to deal

with vulnerabilities with a greater severity level. Our results seem to give credit to recent security regulations [11], [12], [29] and encourage practitioners to adopt SCA to improve the security of their software supply chains.

Paper Structure. In Section II, we present the background and related work. We describe the design of our cohort study in Section III. The results are shown and discussed in Section IV and Section V. In Section VI, we conclude the paper.

II. Background and Related Work

In this section, we first present the SCA tool (*i.e.,* OWASP DC) studied in this paper and then the research related to ours.

A. OWASP Dependency-Check

OWASP DC is an SCA tool primarily conceived to detect publicly disclosed vulnerabilities affecting the components of a project [19]. This tool is owned by OWASP, a nonprofit foundation that works to improve the security of software through education, tools, and collaboration [18]. OWASP DC can be leveraged through a command line interface, an Ant task, and a Maven, Gradle, or a Jenkins plugin [19].

To detect disclosed vulnerabilities, OWASP DC determines whether there is a Common Platform Enumeration (CPE) identifier for a given project component, and, if found, it generates a vulnerability report linking to the associated Common Vulnerabilities and Exposures (CVE) entries [19]. OWASP DC automatically updates itself using the National Vulnerability Database (NVD) Data Feeds hosted by NIST [19]. This process helps developers proactively manage risks associated with third-party software components in their applications.

OWASP DC integrates with various build tools, such as Maven, Gradle, and Jenkins, and can be incorporated into CI/CD pipelines, allowing for automated vulnerability checks throughout development. The tool generates detailed reports that highlight potential vulnerabilities, offering a breakdown of each issue's severity, impacted versions, and remediation advice. This information aids development and security teams in prioritizing and addressing vulnerabilities, fostering a secure software development lifecycle.

For our empirical investigation, we considered OWASP DC as an SCA tool because *(i)* it is popular (more than 6.3k stars, 1.3k forks, and 18.5k dependent software projects on GitHub [28]) *(ii)* it retrieves vulnerability information from remarkable databases (*e.g.,* NVD [38]), and *(iii)* the results of our preliminary analysis, based on the GitHub's Dependency Graph [5], [22] of the software repository hosting the OWASP DC project, indicate that this SCA tool is largely used on the Java Maven projects owned by ASF.

B. Empirical Studies on Vulnerable Components

Past studies investigated the vulnerabilities affecting the components of software projects [1], [8], [9], [44], [63], [65]. For instance, Derr *et al.* [9] conducted an empirical study involving the analysis of Android applications and a survey with developers from Google Play. They found that almost all vulnerable components could be updated without changing the source code and vulnerable components are not updated to avoid concerns that could compromise the functioning software (*e.g.,* incompatibilities, high integration effort) but also for a lack of awareness regarding available updates. Decan *et al.* [8] studied the impact of vulnerabilities in npm components and found the number of new vulnerabilities and affected components growing over time. The authors also observed that it takes a long time to discover and fix vulnerabilities, and a large fraction of components dependent on vulnerable ones are not updated even when a fix is available. Zapata *et al.* [63] manually inspected 60 projects adopting a vulnerable version of the `ws`, `angular`, and `marked` components and found that most of them were safe as they did not use the vulnerable code. Zimmerman *et al.* [65] analyzed security concerns in the npm ecosystem by focusing on the impact that vulnerable components can have. The results indicate that vulnerable components could impact large parts of the entire ecosystem and that lack of security maintenance leads to persistent vulnerabilities. Prana *et al.* [44] analyzed vulnerabilities in OS components used by 450 software projects written in Java, Python, and Ruby. The authors found that the most common vulnerabilities related to "Denial of Service" and "Information Disclosure" were of a medium severity level; moreover, components were rarely updated or changed by project owners, despite the availability of updates. Alfadel *et al.* [1] investigated the propagation and life span of vulnerabilities in the Python ecosystem. Their results seem to suggest that vulnerabilities in the packages increase over time and can take more than three years to be found and seven months to be fixed by the dependent.

The most remarkable difference between the above-mentioned studies and ours is that we did not focus on the impact of vulnerable components, but rather on the extent to which adopting an SCA tool can cause a reduction in the vulnerabilities affecting project components as compared with not using these tools at all.

C. Empirical Studies on SCA tools

A few studies have investigated the vulnerability detection capability of SCA tools [6], [27], [64]. Imtiaz *et al.* [27] conducted a case study to compare vulnerability analysis reports of nine industry-leading SCA tools, including OWASP DC, on a large web app dependent on Maven and npm components. The results indicate that SCA tools vary widely in the reporting of disclosed vulnerabilities. As for OWASP DC, Imtiaz *et al.* [27] observed that this SCA tool detected the highest number of vulnerabilities, although a non-negligible portion were false positives. Bottner *et al.* [6] investigated the vulnerability detection capabilities of two SCA tools, namely Eclipse Steady and OWASP DC, in the context of Java projects. The results indicate that OWASP DC detected significantly more true positives than Eclipse Steady. Zhao *et al.* [64] proposed a model to evaluate the performance of SCA tools in detecting dependencies and vulnerabilities of Java Maven projects. According to this model, the authors assessed six SCA tools and found that none of them supported their model well; however,

among those SCA tools, OWASP DC was, at build time, the best at detecting components and the second best at reporting vulnerabilities (only after a commercial SCA tool). Unlike the studies introduced just before, we did not compare the vulnerability detection capability of SCA tools. We compared projects adopting and not adopting an SCA tool (*i.e.,* OWASP DC) with respect to software supply chain security over time. Moreover, different from the aforementioned studies, our comparison also provided cause-effect evidence on whether adopting OWASP DC over time leads to fewer vulnerabilities affecting project components. Our study provides evidence on whether and to what extent the theoretical benefits of an SCA tool like OWASP DC translate into practical benefits when it is adopted in the software development process by developers. Our contribution is built on previous research, *e.g.,* Zhao *et al.* [64] found that, from a theoretical perspective, OWASP DC was the best free SCA tool among all the compared SCA tools. Comprehensive evaluations need different types of studies focusing on different aspects and our study represents a subsequent step in quantitatively evaluating SCA tools.

In their qualitative study, Pashchenko *et al.* [43] investigated developers' decisions for selecting, managing, and updating software components. To this end, they performed 25 semi-structured interviews. Their results indicate that security is prioritized when selecting components if it is enforced by company policy. In case of a disclosed vulnerability within a component, developers mostly rely on community support and social channels to address that vulnerability. In fact, some developers deemed SCA tools to generate many irrelevant or low-priority alerts. We differ from Pashchenko *et al.* [43] as our investigation aims to quantitatively observe the existence of a cause-effect relationship between the adoption of an SCA tool and the vulnerabilities affecting project components.

III. Study Design

Cohort studies are well-established in other fields (*e.g.,* epidemiology) to investigate cause-effect relationships by analyzing observational data [47]. Observational data are collected by recording phenomena as they naturally occur, without interference or manipulation from researchers. This differs from experiments, which involve intervention and control by the researcher. The information obtained from mining software repositories can be considered observational data [3] and a cohort design can be considered a viable means to investigate cause-effect relationships [47], [52]. Cohort studies specifically analyze observational data to examine whether certain factors contribute to specific outcomes over time. To achieve this, the outcomes of two or more groups of study subjects/participants—each with different levels of the factors being investigated—are compared [47].

In the cohort study presented in this paper, we examined one factor, *i.e.,* the adoption of OWASP DC, to observe as outcomes whether this adoption caused a reduction in the number/score of vulnerabilities (according to the CVSS severity levels) affecting the components on which software projects (also subjects, from here onwards) depended.

To unveil cause-effect relationships, cohort studies must assess the association between the factor and the outcome, ensure the temporal precedence of the factor over the outcome, and control for confounders that may influence this relationship [48], [52]. We instantiated these requirements as follows:

- Empirical association between the factor and the outcome. This association is established through inferential statistics (*i.e.,* by observing statistical significance) according to the planned data analysis (see Section III-C).
- Temporal precedence of the factor over the outcome. We analyzed the difference in the number/score of the vulnerabilities affecting the project components between the start and end of the follow-up period. In cohort studies, the follow-up period is the duration of time over which subjects/participants are monitored to observe the outcome [60]. We obtained temporal precedence because OWASP DC was adopted only by one of the two groups being compared, and the projects of this group adopted OWASP DC after the start of the follow-up period.
- Control for other factors, *i.e.,* the confounders, affecting the association between the considered factor and the outcome. We selected comparable subjects adopting and not adopting OWASP DC by removing those differing too much with respect to the confounders (see Section III-B).

A. Goal and Research Question

The goal of our (retrospective) cohort study is to analyze the adoption of an SCA tool, namely OWASP DC, to evaluate its effect on software supply chain security. The perspective is that of *(i)* practitioners who need to manage the vulnerabilities affecting the components on which their software projects depend and *(ii)* researchers who have been investigating SCA for software supply chain security. The context consists of OS Java Maven projects owned by the ASF and publicly hosted on GitHub.

Based on the aforementioned study goal, we formulated and studied the following high-level Research Question (RQ):

RQ. *To what extent does the adoption of OWASP DC, a well-known and largely used SCA tool, impact software supply chain security?*

This RQ aims to study whether and to what extent adopting OWASP DC (possibly) improves the security of the components on which projects depend. That is, we can speculate that such an adoption improves the security of components and, thus, of the software supply chain if the number/score of the vulnerabilities affecting those components decreases when adopting an SCA (as compared to not adopting it).

B. Study Context and Planning

We considered OS Java Maven projects owned by the ASF and publicly hosted on GitHub. We focused on projects owned by the ASF because they are largely popular, even in industrial settings [15], [62]. We chose Maven and Java because they are the most popular package managers and programming languages for the JVM (Java Virtual Machine) ecosystem [54].

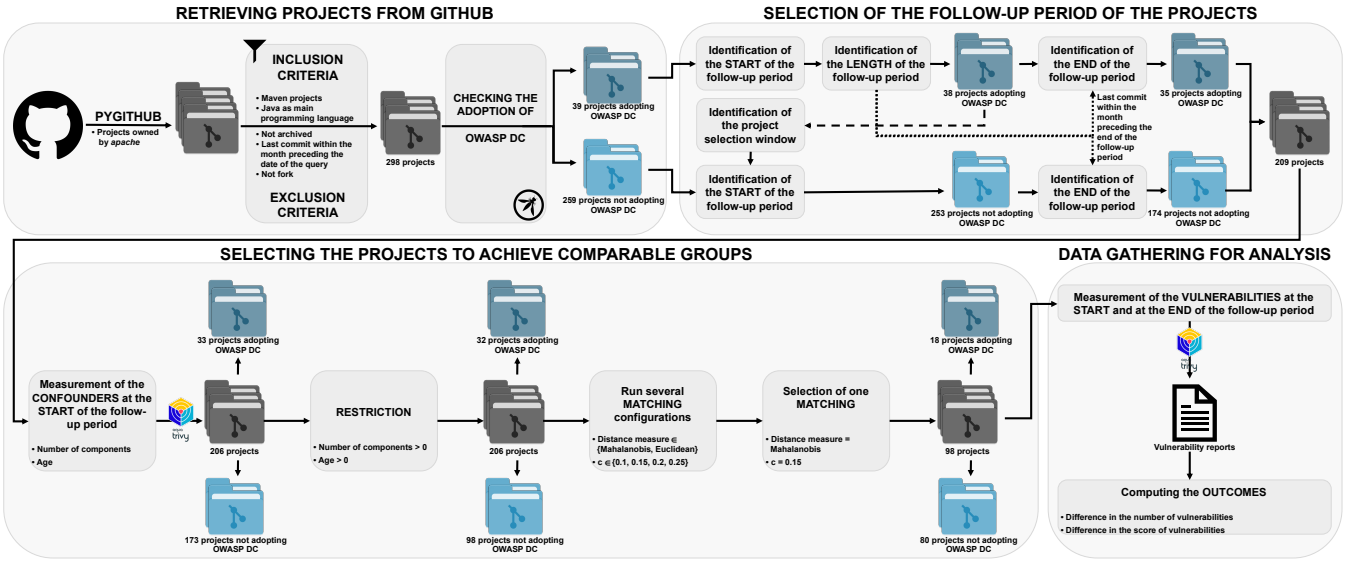

Fig. 1: Process describing the selection of projects.

In addition, according to the 2024 PYPL index, Java ranks as the second most popular programming language [46].

In Fig. 1, we report the process followed to retrieve the projects from GitHub, select those comparable with respect to the confounders, and gather the data about the outcomes. We used PyGithub [45] to retrieve ASF projects publicly hosted on GitHub. PyGithub is a Python library that provides a simplified interface to the GitHub REST API; therefore, it can be used to manage GitHub resources such as repositories, user profiles, and organizations. We used PyGithub to retrieve all repositories owned by `apache`. Among those retrieved, we included in our sample only the repositories that had:

- A `pom.xml` file in the root directory, to select projects with Maven as a package manager;
- Java as the main programming language.

In the enactment of our process, we also took care of the perils of mining GitHub by Kalliamvakou *et al.* [31] and therefore we excluded those repositories that were:

- Archived or without at least one commit in the month preceding the date of the query (May 11^{th}, 2024), to avoid selecting projects no longer active [31].
- Fork, to limit the risk of selecting duplicate projects [31].

We also avoided the risks of selecting personal projects or repositories unrelated to software development (*e.g.,* books, tutorials, assignments) by considering ASF projects [31].

In this way, we retrieved 298 subjects (ASF projects). For each retrieved subject, we checked if it adopted OWASP DC. To this end, for each subject, we analyzed the content of each `pom.xml` file and searched for the string `dependency-check-maven` among the project dependencies. This string identifies the dependency toward OWASP DC in Maven projects and a project that includes this dependency can use OWASP DC for SCA. In case we detected the occurrence of this string in a `pom.xml` file of a project, we deemed that subject as adopting OWASP DC. If we did not find the occurrence of that string in any `pom.xml` file of a given project, we deemed that project as not adopting OWASP DC. Among the 298 retrieved subjects, 39 adopted OWASP DC, while 259 did not.

Afterward, we identified, for each subject, the start and the end of their follow-up period, which was required to gather the data about the vulnerabilities. Indeed, following the cohort design, we had to measure such vulnerabilities at the start and at the end of the follow-up period for each subject. Fig. 2 shows the selection of the follow-up periods in our cohort study. As for the projects adopting OWASP DC, we considered for each subject the commit before the adoption of OWASP DC as the start of the follow-up period (this guarantees the temporal precedence of the factor over the outcome). To decide the length of the follow-up period, which is the same for all the projects (adopting and not adopting OWASP DC), we computed for each subject the time passed from the start of the adoption of OWASP DC till the day we retrieved the repositories with PyGithub. The follow-up period must be long enough to observe a change in the outcome [60]. We observed that the first and second shortest periods from the start of the adoption of OWASP DC till the day we retrieved the repositories with PyGithub was 40 and 264 days, respectively. 40 days might be too short a time because upgrading components to non-vulnerable versions can take some months [1], [44] and, therefore, we chose 264 days as the length of the follow-up period. This resulted in discarding only one subject.

For each project adopting OWASP DC, we mined the commit history of all `pom.xml` files, searching for the first commit introducing the dependency toward OWASP DC. We considered as the start of the follow-up period the commit immediately preceding that of the adoption of OWASP DC. This mining process was facilitated by the use of PyDriller [58].

As for the projects not adopting OWASP DC, we had to

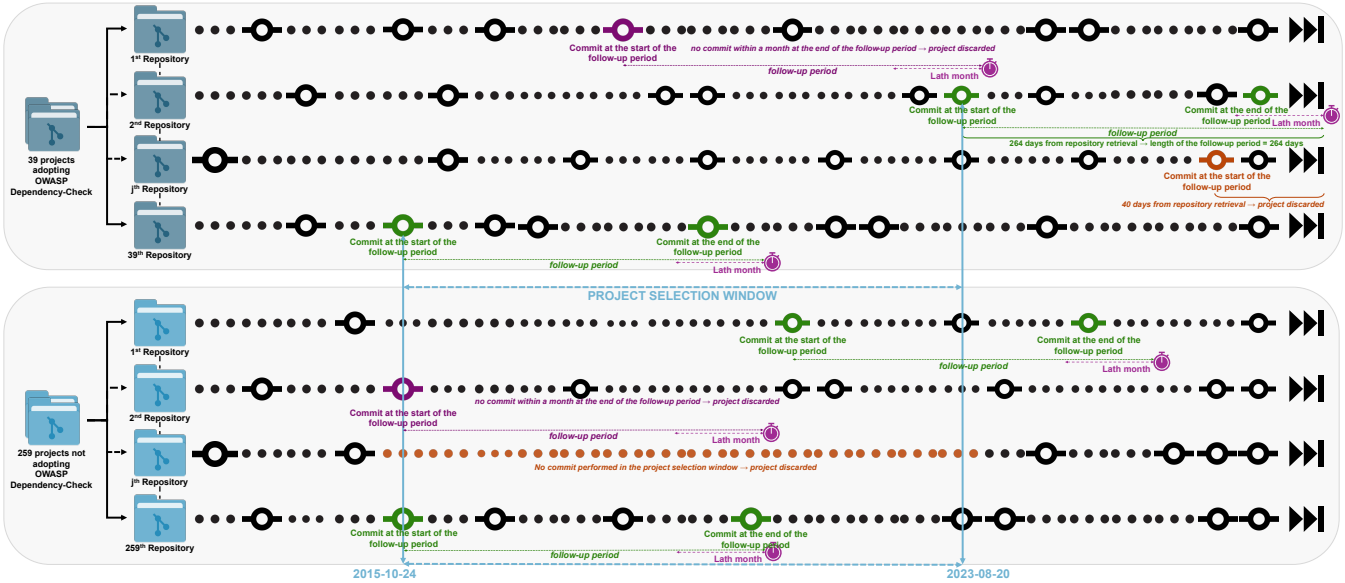

Fig. 2: Selection of the follow-up period of the projects.

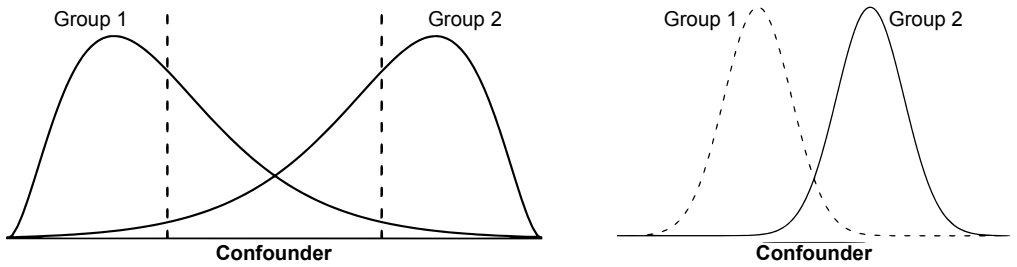

Fig. 3: Distribution of a confounder in two different groups. Example of lack of balance (left) and partial overlap (right).

select subjects that were under development for a period of time similar to those adopting OWASP DC [60]. To this end, we defined a project selection window and selected the projects not adopting OWASP DC having at least a commit in it. As shown in Fig.2, the project selection window starts with the date of the oldest commit of the follow-up periods of the subjects adopting OWASP DC (39^{th} repository in the figure). The project selection window lasts till the date of the commit corresponding to the start of the follow-up period of the subject that lastly adopts the OWASP DC (2^{nd} repository in the figure). The first commit of the projects not adopting OWASP DC within the project selection window represents the start of their follow-up period. We discarded six subjects not adopting OWASP DC because they had no commits within the project selection window.

To avoid considering subjects that were inactive during the follow-up period (both adopting and not adopting OWASP DC), we discarded those subjects whose commit at the end of the follow-up period was not performed within a month [31]. In the end, we obtained 209 subjects, 35 adopted OWASP DC and 174 did not. It is worth mentioning that both projects adopting and not adopting OWASP DC are studied in their follow-up period.

Causal inferences are only possible when the subjects exposed to the factor (the adoption of OWASP DC) are comparable to those not exposed [21]. Comparability between groups can be hindered by the lack of balance and/or partial overlap of confounders (see Fig. 3) [21].

Lack of balance refers to a difference in the distributions of a variable between the groups being compared (left side of Fig. 3). In case such a variable is a confounder, lack of balance could compromise the estimation of the effect of the factor. Partial overlap occurs when the range of values assumed by a variable is different between the groups being compared (right side of Fig. 3). In case such a variable is a confounder, partial overlap compromises the comparison of the groups in the non-overlap regions because there are no subjects in both groups having the same values for the confounder. To assess the comparability between groups with respect to confounders, cohort studies typically rely on the absolute standardized mean difference (SMD) and the Kolmogorov–Smirnov (KS) test [36]. SMD measures the distance between two group means in terms of one or more variables [48]. In cohort studies, these variables are the confounders. The KS test verifies if two groups come from the same distribution [20]. In cohort studies, for each group, every confounder should come from the same distribution. To assess the comparability of groups, we needed to identify confounders and then measure them for

TABLE I: Values of the SMD and the KS test of the confounders, after each strategy for assessing the comparability of groups.

Strategy	Distance measure	c	# Projects	OWASP DC		SMD		p-value of the KS test	
				# Not adopting	# Adopting	# Components	Age	# Components	Age
None	-	-	206	173	33	0.708	0.807	2.17E-11	3.88E-05
Restriction	-	-	130	98	32	0.671	**0.162**	1.88E-06	**0.35**
Matching	Mahalanobis*	0.1	72	56	16	**0.178**	**0.012**	3.53E-04	**0.87**
		0.15*	98	80	18	**0.208**	**0.145**	1.21E-04	**0.59**
		0.2	119	96	23	**0.209**	0.270	1.27E-04	**0.26**
		0.25	123	98	25	0.222	0.268	4.13E-05	**0.30**
	Euclidean	0.1	87	70	17	**0.182**	**0.150**	4.10E-04	**0.58**
		0.15	111	89	22	**0.176**	0.258	3.60E-04	**0.41**
		0.2	122	98	24	0.212	0.331	5.83E-05	**0.17**
		0.25	125	98	27	0.286	0.241	2.10E-05	**0.22**

* indicates the matching configuration chosen among all the performed matching configurations.

each subject at the start of the follow-up period [21]. In the literature, there is no systematic approach to identifying the confounders for a given study. However, guidelines indicate that confounders should be associated with the factor or outcome under investigation, and such association should have been assessed by previous research [51]. In our study, we identified the following confounders:

- *Number of components on which a project depends*. The number of components on which a project depends is correlated to the number of vulnerabilities affecting the components of that project [23]. For each subject at the start of the follow-up period, we computed the number of components using another SCA tool, namely Trivy [56], which we later also used to gather data about the vulnerabilities.[1] We performed a `git-checkout` operation to change the project version to that indicated by the commit at the start of the follow-up period. We failed to measure three subjects (two adopted OWASP DC and the other did not). More in detail, for one subject, we encountered a failure in the `git-checkout` operation, while, for the other two, the analysis process carried out by Trivy had never reached completion, even after hours (so we had to kill these processes manually). As a result, the total number of subjects was 206.
- *Age of a project*. The age of a project is correlated to the number of vulnerabilities affecting the components of that project [4], [23]. In addition, older subjects could keep depending on components with disclosed vulnerabilities because such components cannot be replaced, *e.g.,* due to technological constraints [41]. For each project, we computed the age as the difference between the date of the commit at the start of the follow-up period and its first commit.

[1]We used Trivy rather than OWASP DC to avoid biases influencing the outcomes in favor of the projects adopting OWASP DC. We recall that different SCA tools might detect different vulnerabilities for a given software project [27], [40]. Since OWASP DC was adopted by one of the two groups being compared in this study, using the same tool for measuring the vulnerabilities would have favored that group adopting it with respect to software supply chain security. We opted for Trivy because: *(i)* it is popular (more than 23.1k stars, 2.3k forks, and 400 contributors on GitHub in 10/2024 [2]), *(ii)* previous studies pointed out its vulnerability detection capability [26], [32], [40], and *(iii)* other studies have already leveraged it (*e.g.,* [25], [35], [37]).

Once confounders were identified, we proceeded toward the assessment of the comparability in the adopting and not adopting OWASP DC groups. The first row of Table I shows the values of the SMD and the KS test of the confounders. An SMD below 0.20 indicates that the confounder does not differ much between groups [50], while a p-value higher than 0.05 for the KS test indicates that the confounder comes from the same distribution for both groups [36]. These values for the SMD and the KS test are those desired to deem the two groups of subjects comparable (in boldface in Table I).

Since we did not have desirable values of the SMD and KS test for any confounders, we used restriction. This strategy selects subjects having the same value for one or more confounders [21]. As far as the restriction, we discarded those subjects that, at the start of the follow-up period, *(i)* had no components on which depended (since, to have vulnerabilities affecting components, a project needs to have at least one component) and *(ii)* had just been created (to avoid subjects inactive at the start of the follow-up period [31]). After restriction, we had 130 subjects, 32 adopting and 98 not adopting OWASP DC.

With restriction, we could achieve desirable values for the SMD and KS test just for one confounder, *i.e.,* age, as shown in the second row of Table I. Therefore, to assess the other confounder, *i.e.,* number of components, we used matching. Given the 130 subjects, matching selects similar subjects between groups with respect to the confounders [21]. There are several configurations to perform matching [48]. It is impossible to know which matching configuration leads to the most comparable groups unless such matching configurations are performed and evaluated with the SMD and KS test. For this reason, it is recommended to run several matching configurations and select the one that maximizes *(i)* the number of subjects not discarded and *(ii)* the comparability between groups with respect to the confounders. Our matching configurations involved running optimal full matching with different constraints on how much a pair of subjects (one adopting and the other not adopting OWASP DC) could

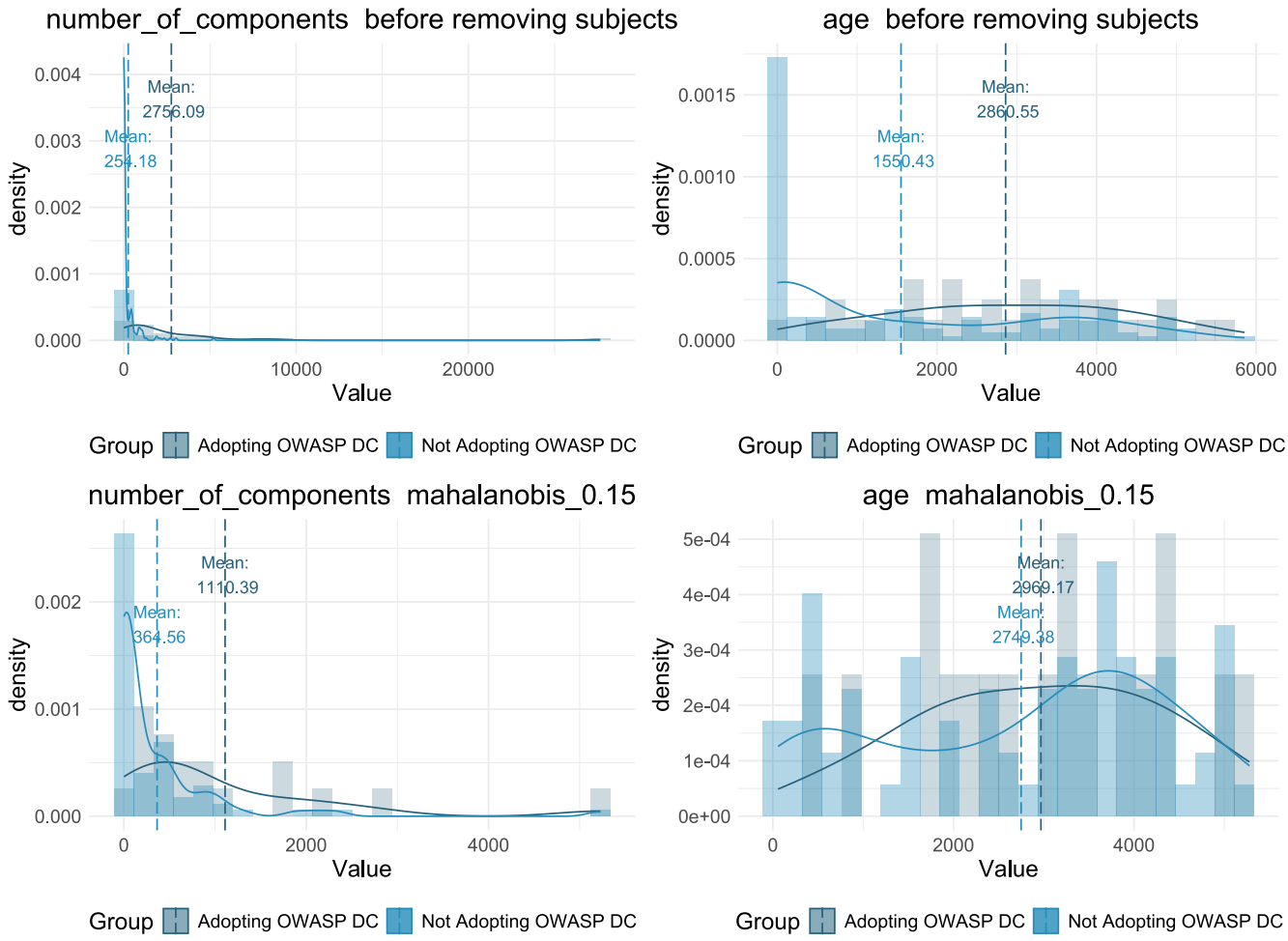

Fig. 4: Kernel density plots of the confounders.

distance with respect to both confounders [48]. These distances were computed, for each pair of subjects, based on the values of their confounders and a distance measure (*i.e.,* Mahalanobis or Euclidean). The constraint on the maximum allowable distance was given by multiplying the standard deviation (SD) of the distance measures with a coefficient c ranging from 0.1 to 0.25 [48]. After running several matching configurations, we had to choose only one matching configuration out of all those performed (see from the third row onwards of Table I). No matching allowed us to obtain a desirable p-value for the KS test of the number of components. We chose the matching with Mahalanobis as a distance measure and 0.15 for c. Fig. 4 shows, for each confounder and group, the kernel density plots before (top) and after (bottom) removing the subjects for assessing the comparability of groups. The plots confirm that, especially for the age, we were able to address the lack of balance and partial overlap (see Fig. 3). We also conducted an a priori power analysis with G*Power [13] to determine if the 98 subjects resulting from the selected matching were enough for the inferential analysis. The results indicated that to achieve 80% power with a significance level of 0.05, the required sample sizes for detecting small, medium, and large effects were, respectively, 395, 55, and 25 for the statistical analysis we planned to conduct (which is described in Section III-C). Thus, our sample size of 98 subjects is adequate for detecting at least a medium effect size and a possible small effect size in the inferential analysis.

Lastly, we used Trivy on the 98 subjects (18 adopting and 80 not adopting OWASP DC) to gather data about the vulnerabilities affecting project components, at the start and end of the follow-up period. Trivy gathers data on all vulnerabilities disclosed up to the present time. Therefore, we excluded those vulnerabilities that, at the time of the start/end of the follow-up period, had not yet been disclosed. We gathered vulnerability data according to different severity levels. Thus, we considered the CVSS score [39] provided by the NVD [38]. The CVSS score ranges from 0 to 10 and such a value also indicates a severity level. From the least to the most severe level (the associated score range in parentheses), we found: info (0), low (0.1-3.9), medium (4-6.9), high (7-8.9), and critical (9-10). For each project and each severity level, we also computed the difference between the number/score of the vulnerabilities at the end and the start of the follow-up period.

C. Data Analysis

To answer our RQ, we first characterized with descriptive statistics (mean and SD) the outcomes of our study, *i.e.,* the difference between the number/score of the vulnerabilities at the end and the start of the follow-up period. Since the data analysis in cohort studies must also address confounder concerns [52], we employed statistical adjustment. Statistical adjustment allows comparing group means by adjusting those means for one or more confounders (which could affect the outcome) [14]. We employed this technique by constructing a statistical model incorporating only one confounder, *i.e.,* the number of components. The reason is that, while we managed to achieve comparability between groups for age, even after matching, the distribution of the number of components was not sufficiently similar between the two groups. Several statistical models can be used to perform statistical adjustments, each of which works according to specific data requirements. Due to the lack of homogeneity of the regression slopes for ANCOVA and the normality of residuals for the Linear Mixed Model (LMM), we had to use the Generalized Linear Model (GLM) [14]. For each of the five CVSS severity levels, we built one GLM for the difference in the number of vulnerabilities and another for the difference in the score associated with that severity. We also built two other GLMs without distinguishing per severity level. This led us to observe 12 possible outcomes. Accordingly, the formula of the (confounder-adjusted) GLMs can be summarized as follows: $outcome \sim adopting\ owasp\ dc + number\ of\ components$. In reporting the results, cohort studies should also provide and compare the confounder-adjusted estimates with the unadjusted estimates for triangulation purposes [59]. To this end, we built another 12 GLMs without incorporating any confounders: $outcome \sim adopting\ owasp\ dc$. For each GLM, we provided the model estimation and p-value of the factor (*i.e.,* adopting OWASP DC) and, if present, the confounder. Note that a p-value below 0.05 would indicate the significance of the factor/confounder in determining the outcome. To assess the quality of each GLM, we computed the Akaike Information Criterion (AIC), which indicates how well a model fits the data it was generated from: the smaller the AIC, the better the fit [14]. As an effect size measure, for each GLM, we computed the Cohen's f^2 [24]. The practical effect of f^2 is: small, if $0.02 \leq f^2 < 0.15$; medium, if $0.15 \leq f^2 < 0.35$; or large, if $f^2 \geq 0.35$ [7].

IV. Results

As reported in Table II, on average, the projects adopting OWASP DC had a reduction in the number (-3.389) and score (-28.394) of all vulnerabilities. Except for vulnerabilities with a low severity level, there was a reduction for all severity levels (ranging from -0.056 to -2 for the number, and from -0.117 to -16.322 for the score). Indeed, vulnerabilities with a low severity level underwent a tiny increase in the number (0.111) and score (0.367). As for the projects not adopting OWASP DC, there was an increase in the overall number (0.75) and score (5.008) of all vulnerabilities. However, such an increase is not consistent across the different severity levels. The number/score of vulnerabilities with info and low severity levels remained unchanged (means equal to zero). As for the vulnerabilities with medium and high severity levels, their number (0.25 and 0.513, respectively) and score (1.176 and 3.954, respectively) increased. The number/score of vulnerabilities with a critical severity level underwent a tiny decrease (-0.013 and -0.123, respectively). Both with and without distinguishing per severity levels, the SD values indicate a higher dispersion of the mean values for projects adopting OWASP DC than those not adopting it. This could have been influenced by the small number of projects adopting OWASP DC (*i.e.,* only 18).

TABLE II: Descriptive statistics on the vulnerabilities.

Difference	Severity	OWASP DC	Mean		SD
			Value	95% CI	
Number	Info	Not Adopting	0	[-,-]	0
		Adopting	-0.056	[-0.173,0.062]	0.236
	Low	Not Adopting	0	[-,-]	0
		Adopting	0.111	[-0.05,0.272]	0.323
	Medium	Not Adopting	0.25	[-0.043,0.543]	1.317
		Adopting	-0.333	[-1.268,0.601]	1.879
	High	Not Adopting	0.513	[0.27,0.755]	1.091
		Adopting	-2	[-6.372,2.372]	8.792
	Critical	Not Adopting	-0.013	[-0.037,0.012]	0.112
		Adopting	-1.111	[-4.898,2.676]	7.615
	All	Not Adopting	0.75	[0.316,1.184]	1.952
		Adopting	-3.389	[-11.941,5.164]	17.198
Score	Info	Not Adopting	0	[-,-]	0
		Adopting	-0.117	[-0.363,0.129]	0.495
	Low	Not Adopting	0	[-,-]	0
		Adopting	0.367	[-0.164,0.897]	1.067
	Medium	Not Adopting	1.176	[-0.371,2.723]	6.951
		Adopting	-1.461	[-6.858,3.936]	10.854
	High	Not Adopting	3.954	[2.072,5.835]	8.456
		Adopting	-16.322	[-51.785,19.141]	71.312
	Critical	Not Adopting	-0.123	[-0.366,0.121]	1.096
		Adopting	-10.861	[-47.976,26.253]	74.634
	All	Not Adopting	5.008	[2.244,7.771]	12.418
		Adopting	-28.394	[-102.623,45.834]	149.267

Table III shows the results of the inferential statistics. Both with and without distinguishing per severity levels, the AIC values are slightly smaller for the confounder-unadjusted GLMs; thus, adjusting the GLMs for the confounder led to GLMs with a slightly worse fit to data. Since the difference in the AIC values for the adjusted and unadjusted GLMs is tiny (at most two, for the difference in the number of all vulnerabilities and for the difference in the score of vulnerabilities with a high severity level), we can postulate that this is because adjusting for confounders was not necessary. In fact, the AIC is a measure of fit that penalizes the model for having more factors [14]. In other words, the restriction and matching had already led to comparable groups for both the confounders.

As for the effects of adopting OWASP DC, we can observe that the relationship between this factor and each outcome is statistically significant (p-values < 0.05), except for the difference in the number/score of vulnerabilities with medium and

TABLE III: Results of the inferential statistics.

Difference	Severity	Adjustment	AIC	Adopting OWASP DC			Number of components			f^2
				Value	95% CI	p-value	Value	95% CI	p-value	
Number	Info	No	-170.82	-0.056	[-0.106,-0.005]	**0.034**	-	-	-	0.048
		Yes	-169	-0.059	[-0.113,-0.005]	**0.033**	4.97E-06	[-1.84E-05,2.83E-05]	0.677	0.049
	Low	No	-108.83	0.111	[0.042,0.181]	**0.002**	-	-	-	0.102
		Yes	-107.01	0.106	[0.032,0.18]	**0.006**	6.81E-06	[-2.52E-05,3.88E-05]	0.677	0.083
	Medium	No	352.54	-0.583	[-1.316,0.149]	0.122	-	-	-	0.025
		Yes	352.64	-0.756	[-1.527,0.014]	0.057	2.32E-04	[-1.02E-04,5.66E-04]	0.176	0.039
	High	No	545.28	-2.513	[-4.471,-0.554]	**0.014**	-	-	-	0.066
		Yes	547.27	-2.541	[-4.621,-0.461]	**0.019**	3.83E-05	[-8.63E-04,9.40E-04]	0.934	0.06
	Critical	No	510.44	-1.099	[-2.738,0.541]	0.192	-	-	-	0.018
		Yes	511.85	-0.881	[-2.618,0.855]	0.322	-2.91E-04	[-0.001,4.61E-04]	0.45	0.01
	All	No	675.72	-4.139	[-7.948,-0.329]	**0.036**	-	-	-	0.047
		Yes	677.72	-4.132	[-8.179,-0.085]	**0.048**	-9.01E-06	[-0.002,0.002]	0.992	0.042
Score	Info	No	-25.40	-0.117	[-0.223,-0.01]	**0.034**	-	-	-	0.048
		Yes	-23.58	-0.124	[-0.237,-0.011]	**0.033**	1.04E-05	[-3.85E-05,5.94E-05]	0.677	0.049
	Low	No	125.18	0.367	[0.137,0.596]	**0.002**	-	-	-	0.102
		Yes	127.00	0.35	[0.106,0.594]	**0.006**	2.25E-05	[-8.31E-05,1.28E-04]	0.677	0.083
	Medium	No	684.36	-2.637	[-6.619,1.344]	0.197	-	-	-	0.018
		Yes	684.62	-3.539	[-7.731,0.653]	0.101	0.001	[-6.08E-04,0.003]	0.195	0.029
	High	No	954.99	-20.276	[-36.113,-4.439]	**0.014**	-	-	-	0.066
		Yes	956.99	-20.350	[-37.173,-3.526]	**0.020**	9.90E-05	[-0.007,0.007]	0.979	0.059
	Critical	No	957.81	-10.739	[-26.805,5.328]	0.193	-	-	-	0.018
		Yes	959.22	-8.613	[-25.629,8.403]	0.324	-0.003	[-0.01,0.005]	0.451	0.01
	All	No	1096.67	-33.402	[-66.031,-0.773]	**0.048**	-	-	-	0.042
		Yes	1098.63	-32.277	[-66.931,2.378]	0.071	-0.002	[-0.017,0.014]	0.844	0.035

critical severity levels. There is also no statistical significance for the confounder-adjusted GLM in the difference of the score of all vulnerabilities. Not having identified a statistically significant relationship between adopting OWASP DC and an outcome does not necessarily mean that such a relationship does not exist [30], [61]. As for the difference in the score of all vulnerabilities for the confounder-adjusted GLM, we can postulate that the loss of significance, compared with the confounder-unadjusted GLM, is due to having adjusted for the confounder, even though it was not necessary. The presence of a statistically significant relationship between adopting OWASP DC and an outcome indicates that such an adoption causes a change in that outcome. To understand whether that change represents an increase or reduction in the number/score of vulnerabilities, we can consider the value estimated for adopting OWASP DC. Negative values indicate that adopting OWASP DC leads to a reduction in the number/score of vulnerabilities, while positive values indicate an increase. We can observe that only vulnerabilities with a low severity level are associated with an increase in number/score when adopting OWASP DC. For all the other severity levels, adopting OWASP DC is associated with a reduction in the number/score of vulnerabilities. This reduction also happens for the vulnerabilities with medium and critical severity levels, for which we did not observe statistical significance when adopting OWASP DC.

The Cohen's f^2 effect sizes are small ($0.02 \leq f^2 < 0.15$) for all the outcomes, except for the difference in the number/score of critical vulnerabilities where the Cohen's f^2 is negligible (0.018 and 0.01 for the confounder-unadjusted and -adjusted GLMs, respectively). The Cohen's f^2 is also negligible for the difference in the score of vulnerabilities with a medium severity level of the confounder-unadjusted GLM (0.018) but not for the confounder-adjusted GLM (0.029). These confounder-unadjusted GLMs do not reach a small effect size for only 0.002. The Cohen's f^2 effect sizes are similar between the confounder-unadjusted and -adjusted GLMs (the maximum difference is 0.019 for the number/score of vulnerabilities with a low severity level). We can also observe that the Cohen's f^2 effect sizes are slightly higher for the confounder-unadjusted GLMs, except for the number/score of both vulnerabilities with info and medium severity levels. Then, we can acknowledge a consistency between the confounder-unadjusted and -adjusted GLMs.

Based on our results, we can answer our RQ as follows:

> The use of OWASP DC causes a significant reduction in the overall number/score of vulnerabilities affecting project components. In detail, a significant reduction was observed for the vulnerabilities with info and high severity levels. As for vulnerabilities with medium and critical severity levels, we did not observe any cause-effect relationship associated with the use of OWASP DC. We also observed OWASP DC causing a significant increase in the number/score of vulnerabilities with a low severity level. The practical effect of using OWASP DC was generally small.

V. Discussion

In this section, we discuss the main findings and the threats that might affect the validity of our study.

A. Overall Discussion and Implications

In frames, we highlight the main findings distilled from the results of our study. Every framed finding is followed by a compelling discussion that deepens understanding.

Finding #1: Using OWASP DC causes a small reduction in the overall number/score of vulnerabilities affecting components on which ASF projects depend. This reduction has also been observed for vulnerabilities with info and high severity levels.

Our results suggest that using OWASP DC can lead developers to reduce the overall number/score of vulnerabilities affecting project components, including those with info and high severity levels according to the CVSS. Therefore, using an SCA tool like OWASP DC can improve the security of the software supply chain, as pointed out by recent regulations [11], [29]. We advise **practitioners** to use an SCA tool to manage the vulnerabilities affecting their project components. This would also allow them to overcome difficulties like the lack of awareness regarding available updates of vulnerable components [8], [9], [44] and facilitate security maintenance activities [65].

We recall that Pashchenko *et al.* [43] pointed out that security is prioritized for selecting components if enforced by organization policy. Therefore, we can postulate that the observed reduction in the vulnerabilities affecting project components may be due to the context of our cohort study, namely ASF projects. These projects share a common set of minimum requirements related to security and then developers should always work with the ASF Security Team when dealing with vulnerabilities [16], [17]. The context of our study could also justify the presence of a small practical effect associated with using OWASP DC. We can postulate that using OWASP DC in projects where software security is not enforced could lead to observing a larger practical effect related to its use. In other words, the use of an SCA tool like OWASP DC in contexts different from that of our study (*e.g.,* a different software ecosystem) could lead to greater improvements in the security of the software supply chain. **Researchers** could be interested in replicating our study to observe whether our results hold in different contexts.

Finding #2: We observed no significant reduction in the number/score of vulnerabilities with medium and critical severity levels when using OWASP DC in ASF projects. The practical effect was small for vulnerabilities with a medium severity level and almost small for vulnerabilities with a critical severity level.

With a proper experimental design (like in the cohort design), statistical significance makes it possible to conclude that a cause-effect relationship is present [52]. The fact that statistical significance is not observed does not imply that a cause-effect relationship does not exist [30], [61]. Therefore, even if such a kind of relationship is not shown, studying the practical effect of a factor on the outcome remains important, and it can be estimated by a proper effect size measure [30], [61]. In our study, we observed that the practical effect in the reduction of the number/score of vulnerabilities was small for vulnerabilities with a medium severity level and almost small for those with a critical severity level. The results on the effect size could motivate **researchers** to investigate the cause-effect relationships related to vulnerabilities with medium and critical severity levels [10]. Our finding also allows **practitioners** to make more informed decisions on adopting SCA tools to address vulnerabilities with medium and critical severity levels.

Finding #3: Using OWASP DC causes a small increase in the number/score of vulnerabilities with a low severity level affecting ASF project components.

Using OWASP DC did not always lead developers to reduce the number/score of vulnerabilities affecting project components in ASF projects. Vulnerabilities with a low severity underwent an increase that, although practically small, can be attributed to the adoption of OWASP DC. In other words, adopting OWASP DC caused an increase in the number/score of vulnerabilities with a low severity level. Since we observed a reduction in the vulnerabilities for all the other severity levels, we can speculate that developers in ASF projects consciously left unresolved vulnerabilities with a low severity level (as alerted by OWASP DC) because they believed them of scant relevance with respect to software supply chain security. For example, the vulnerable code of the components affected by such vulnerabilities might not be executed at runtime [63], or developers might have prioritized the resolution of vulnerabilities with a greater severity level [43].

Researchers could be interested in further investigating vulnerabilities affecting project components, especially with respect to their severity levels. This could involve qualitative investigations with ASF developers about the reasons for deciding whether or not to fix certain vulnerabilities.

B. Threats To Validity

To deal with the threats that might affect the validity of our study, we followed Shadish *et al.*'s guidelines [53]. We ranked the threats from the most to the least sensible for our research goal—*i.e.,* from internal to external validity threats. The rationale behind this decision relies on the fact that we were more interested in studying that cause-effect relationships were correctly identified [53]. This is typical of cohort studies (as well as experiments) in that they seek to unveil cause-effect relationships between factors and outcomes.

Internal Validity. They are concerned with factors internal to a study that might affect its results [53]. In particular, these threats relate to causality. We acknowledge a threat of confounding bias, which affects the association between the factor and the outcome [49]. We controlled confounders by following

a cohort design [52]. It is worth mentioning that there may be confounders that we have not been able to identify and measure. An example of an unmeasured confounder might be the use of tools other than OWASP DC to deal with security—*e.g.,* a developer of a project not adopting OWASP DC might have used static- and/or dynamic- application security testing tools on their computer to detect vulnerabilities before committing the code on GitHub. To strengthen the internal validity of our study (also for unmeasured confounder), we considered OS projects that were similar to each other in the way they were managed, namely ASF projects. These projects share a common set of minimum requirements related to governance, technologies, community, security, *etc.* [16]. In this way, the likelihood that a developer used tools other than OWASP DC is the same for the projects of both groups being compared, and therefore this unmeasured confounder should not represent a threat to the validity of our results.

Construct Validity. They are concerned with the relationship between theory and observation [53]. In a cohort study, a threat to such a kind of validity can be the loss to follow-up [33], *i.e.,* discarding subjects from the study due to concerns that occurred during their follow-up period. In our case, we discarded the inactive projects at the end of the follow-up period. We also verified that all projects using OWASP DC consistently adhered to this SCA tool throughout their follow-up period by analyzing all commits made. Another threat to construct validity could be the tool used to make the measurements, *i.e.,* Trivy. We recognized the possibility that a different tool might detect different components and vulnerabilities. However, the use of Trivy or another tool would equally affect the confounders and outcomes in favor of the projects using either OWASP DC or not. This entails that the accuracy of Trivy affected the measurements of the projects adopting OWASP DC in the same way as the projects not adopting OWASP DC. Therefore, the use of Trivy as a measurement instrument should not represent a threat to the validity of our results.

Conclusion Validity. They are concerned with the statistical conclusions, including sample composition and size [53]. To mitigate the threat of random heterogeneity of subjects, we compared two groups of projects from ASF. By considering projects from the ASF, we also mitigated a threat of reliability of treatment implementation according to which the different projects adopting OWASP DC might have used this tool differently from each other. Although we conducted an a priori power analysis, there could still be the risk that the number of projects could have affected the validity of the conclusion. Finally, we mitigated a threat of violated assumptions of statistical tests by using proper tests (*e.g.,* we employed the GLM since the assumptions for the ANCOVA and LMM did not hold). It is worth mentioning that multicollinearity did not pose a threat because the statistical models used to analyze the data included at most one confounder.

External Validity. They are concerned with the generalizability of results [53]. There could be a threat of interaction of selection and treatment when generalizing our findings to a population different from that being studied. The ecosystem of which the projects are part (*i.e.,* ASF) might represent a threat of interaction of setting and treatment. We acknowledge that our results might not be generalized to other ecosystems. For example, the practical effect of the observed cause-effect relationships might be greater in other ecosystems where requirements related to security are less stringent than those in ASF. However, ASF projects are largely popular, even in industrial settings [15], [62]. We also acknowledge that our results might not be generalized to an SCA tool different from OWASP DC. Our empirical evidence justifies future work on other ecosystems (*e.g.,* npm) and different kinds of SCA tools.

VI. Conclusion

We presented the results of a cohort study investigating whether using OWASP DC (with respect to not using it) causes a reduction in the vulnerabilities affecting the project components. We found that, among the OS Java Maven projects owned by ASF, using OWASP DC as an SCA tool caused a reduction in the overall number/score of vulnerabilities affecting their components. Such a reduction has a small practical effect. The obtained results have several practical implications and complement the current literature. For example, even though past research outlined possible concerns associated with the correctness of the detected vulnerabilities, we provided cause-effect evidence that SCA tools could improve the security of the software supply chain and this was possible thanks to the use of a cohort design. Therefore, our research also has the merit of showing that such a kind of design can be effectively used in the SE field to retrospectively unveil cause-effect relationships on software repositories. In future work, we plan to corroborate our quantitative results with a qualitative investigation of the kinds of vulnerabilities that have and have not been resolved in the software projects considered in our cohort study.

Data Availability

The raw data and R analysis scripts are available online [42].

Acknowledgments

This research has been financially supported by Grant PID2022-137846NB-I00 funded by MCIN/AEI/10.13039/501100011033 and by ERDF A way of making Europe. This research has also been financially supported by the European Union NEXTGenerationEU project and by the Italian Ministry of the University and Research MUR, a Research Projects of Significant National Interest (PRIN) 2022 PNRR, project n. D53D23017310001 entitled ‘Mining Software Repositories for enhanced Software Bills of Materials (MSR4SBOM)’.

References

[1] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical analysis of security vulnerabilities in python packages," *Empirical Software Engineering*, vol. 28, no. 3, p. 59, 2023.

[2] aquasecurity. (2024) aquasecurity/trivy: Find vulnerabilities, misconfigurations, secrets, sbom in containers, kubernetes, code repositories, clouds and more. Last Accessed: 2024-11-08. [Online]. Available: https://github.com/aquasecurity/trivy/

[3] C. Ayala, B. Turhan, X. Franch, and N. Juristo, "Use and misuse of the term "experiment" in mining software repositories research," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4229–4248, 2021.

[4] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the apache community upgrades dependencies: an evolutionary study," *Empirical Software Engineering*, vol. 20, pp. 1275–1317, 2015.

[5] D. Bifolco, S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, "On the accuracy of github's dependency graph," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 242–251. [Online]. Available: https://doi.org/10.1145/3661167.3661175

[6] L. Bottner, A. Hermann, J. Eppler, T. Thüm, and F. Kargl, "Evaluation of free and open source tools for automated software composition analysis," in *Proceedings of the 7th ACM Computer Science in Cars Symposium*, 2023, pp. 1–11.

[7] J. Cohen, *Statistical power analysis for the behavioral sciences*. routledge, 2013.

[8] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th international conference on mining software repositories*, 2018, pp. 181–191.

[9] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on android," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2187–2200.

[10] P. D. Ellis, *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results*. Cambridge university press, 2010.

[11] European Commission. (2022) EUR-Lex - 52022PC0454 - EN - EUR-Lex. Last Accessed: 2024-11-08. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A52022PC0454

[12] European Union Agency for Cybersecurity. (2020) Guidelines for securing the Internet of Things - ENISA. Last Accessed: 2024-11-08. [Online]. Available: https://www.enisa.europa.eu/publications/guidelines-for-securing-the-internet-of-things/

[13] F. Faul, E. Erdfelder, A.-G. Lang, and A. Buchner, "G* power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences," *Behavior research methods*, vol. 39, no. 2, pp. 175–191, 2007.

[14] A. Field, *Discovering statistics using IBM SPSS statistics*. Sage publications limited, 2024.

[15] R. T. Fielding and G. Kaiser, "The apache http server project," *IEEE Internet Computing*, vol. 1, no. 4, pp. 88–90, 1997.

[16] A. S. Foundation. (2024) Apache project minimum requirements. Last Accessed: 2024-11-08. [Online]. Available: https://www.apache.org/dev/project-requirements

[17] ——. (2024) Asf security team. Last Accessed: 2024-11-08. [Online]. Available: https://www.apache.org/security/

[18] O. Foundation. (2024) About the owasp foundation — owasp foundation. Last Accessed: 2024-11-08. [Online]. Available: https://owasp.org/about

[19] ——. (2024) Owasp dependency-check — owasp foundation. Last Accessed: 2024-11-08. [Online]. Available: https://owasp.org/www-project-dependency-check/

[20] M. H. Gail and S. B. Green, "Critical values for the one-sided two-sample kolmogorov-smirnov statistic," *Journal of the American Statistical Association*, vol. 71, no. 355, pp. 757–760, 1976.

[21] A. Gelman, *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2007.

[22] GitHub. (2024) About the dependency graph. Last Accessed: 2024-11-08. [Online]. Available: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph

[23] A. Gkortzis, D. Feitosa, and D. Spinellis, "Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities," *Journal of Systems and Software*, vol. 172, p. 110653, 2021.

[24] J. Groß and A. Möller, "A note on cohen'sd from a partitioned linear regression model," *Journal of Statistical Theory and Practice*, vol. 17, no. 2, p. 22, 2023.

[25] Y. Gu, L. Ying, H. Chai, C. Qiao, H. Duan, and X. Gao, "Continuous intrusion: Characterizing the security of continuous integration services," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1561–1577.

[26] M. S. Haq, T. D. Nguyen, A. Ş. Tosun, F. Vollmer, T. Korkmaz, and A.-R. Sadeghi, "Sok: A comprehensive analysis and evaluation of docker container attack and defense mechanisms," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4573–4590.

[27] N. Imtiaz, S. Thorn, and L. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–11.

[28] Jeremy Long. (2024) jeremylong/dependencycheck: Owasp dependency-check is a software composition analysis utility that detects publicly disclosed vulnerabilities in application dependencies. Last Accessed: 2024-11-08. [Online]. Available: https://github.com/jeremylong/DependencyCheck

[29] Joe Biden. (2021) Executive order on improving the Nation's cybersecurity. Last Accessed: 2024-11-08. [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[30] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[31] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," p. 92–101, 2014. [Online]. Available: https://doi.org/10.1145/2597073.2597074

[32] T. Kim, S. Park, and H. Kim, "Why johnny can't use secure docker images: Investigating the usability challenges in using docker image vulnerability scanners through heuristic evaluation," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 669–685.

[33] V. Kristman, M. Manno, and P. Côté, "Loss to follow-up in cohort studies: how much is too much?" *European journal of epidemiology*, vol. 19, pp. 751–760, 2004.

[34] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies? an empirical study on the impact of security advisories on library migration," *Empirical Software Engineering*, vol. 23, pp. 384–417, 2018.

[35] F. Lombardi and A. Fanton, "From devops to devsecops is not enough. cyberdevops: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline," *Software Quality Journal*, vol. 31, no. 2, pp. 619–654, 2023.

[36] A. Markoulidakis, K. Taiyari, P. Holmans, P. Pallmann, M. Busse, M. D. Godley, and B. A. Griffin, "A tutorial comparing different covariate balancing methods with an application evaluating the causal effects of substance use treatment programs for adolescents," *Health Services and Outcomes Research Methodology*, vol. 23, no. 2, pp. 115–148, 2023.

[37] A. Mills, J. White, and P. Legg, "Longitudinal risk-based security assessment of docker software container images," *Computers & Security*, vol. 135, p. 103478, 2023.

[38] NIST. (2024) Nvd - home. Last Accessed: 2024-11-08. [Online]. Available: https://nvd.nist.gov/

[39] ——. (2024) Vulnerability metrics. Last Accessed: 2024-11-08. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss

[40] S. Nocera, M. Di Penta, R. Francese, S. Romano, and G. Scanniello, "If it's not sbom, then what? how italian practitioners manage the software supply chain," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2024, pp. 730–740. [Online]. Available: https://doi.org/10.1109/ICSME58944.2024.00077

[41] S. Nocera, S. Romano, R. Francese, R. Burlon, and G. Scanniello, "Managing vulnerabilities in software projects: the case of NTT Data," in *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2023, pp. 247–253. [Online]. Available: https://doi.org/10.1109/SEAA60479.2023.00046

[42] S. Nocera, S. Vegas, G. Scanniello, and N. Juristo. (2024) Software Composition Analysis and Supply Chain Security in Apache Projects: an Empirical Study - Supplementary Materials. doi: https://doi.org/10.6084/m9.figshare.27636264. [Online]. Available: https://figshare.com/s/d479d33878a9d4ab16e4

[43] I. Pashchenko, D.-L. Vu, and F. Massacci, "A qualitative study of dependency management and its security implications," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1513–1531.

[44] G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? how vulnerable dependencies affect open-source projects," *Empirical Software Engineering*, vol. 26, pp. 1–34, 2021.

[45] PyGithub. (2024) Pygithub documentation. Last Accessed: 2024-11-08. [Online]. Available: https://pygithub.readthedocs.io/en/latest/index.html

[46] PYPL. (2023) Pypl popularity of programming language index. Last Accessed: 2024-11-08. [Online]. Available: https://pypl.github.io/PYPL.html

[47] M. Robredo, N. Saarimaki, V. Lenarduzzi, S. Vegas, and D. Taibi, "Cohort studies for mining software repositories," in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 569–570.

[48] P. R. Rosenbaum, P. Rosenbaum, and Briskman, *Design of observational studies*. Springer, 2010, vol. 10.

[49] P. R. Rosenbaum and P. R. Rosenbaum, *Overt bias in observational studies*. Springer, 2002.

[50] P. R. Rosenbaum and D. B. Rubin, "Constructing a control group using multivariate matched sampling methods that incorporate the propensity score," *The American Statistician*, vol. 39, no. 1, pp. 33–38, 1985.

[51] K. Rothman, *Modern epidemiology*. Lippincott Williams & Wilkins, 2008.

[52] N. Saarimäki, V. Lenarduzzi, S. Vegas, N. Juristo, and D. Taibi, "Cohort studies in software engineering: A vision of the future," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–6.

[53] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Wadsworth, Cengage Learning, 2002.

[54] Snyk. (2021) Jvm ecosystem report 2021. Last Accessed: 2024-11-08. [Online]. Available: https://res.cloudinary.com/snyk/image/upload/v1623860216/reports/jvm-ecosystem-report-2021.pdf

[55] Snyk. (2024) Guide to software composition analysis (sca) — Snyk. Last Accessed: 2024-11-08. [Online]. Available: https://snyk.io/series/open-source-security/software-composition-analysis-sca/

[56] A. S. Software. (2024) Trivy home - trivy. Last Accessed: 2024-11-08. [Online]. Available: https://trivy.dev/

[57] Sonatype. (2024) What is SCA security — sonatype. Last Accessed: 2024-11-08. [Online]. Available: https://www.sonatype.com/resources/articles/what-is-software-composition-analysis

[58] D. Spadini, M. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 908–911.

[59] E. Von Elm, D. G. Altman, M. Egger, S. J. Pocock, P. C. Gøtzsche, and J. P. Vandenbroucke, "The strengthening the reporting of observational studies in epidemiology (strobe) statement: guidelines for reporting observational studies," *The lancet*, vol. 370, no. 9596, pp. 1453–1457, 2007.

[60] R. Wakabayashi, T. Hirano, T. Laurent, Y. Kuwatsuru, and R. Kuwatsuru, "Impact of" time zero" of follow-up settings in a comparative effectiveness study using real-world data with a non-user comparator: comparison of six different settings," *Drugs-Real World Outcomes*, vol. 10, no. 1, pp. 107–117, 2023.

[61] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer, 2012.

[62] N. Yang, I. Ferreira, A. Serebrenik, and B. Adams, "Why do projects join the apache software foundation?" in *Proceedings of the 2022 ACM/IEEE 44th international conference on software engineering: software engineering in society*, 2022, pp. 161–171.

[63] R. E. Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto, and A. Ihara, "Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm javascript packages," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 559–563.

[64] L. Zhao, S. Chen, Z. Xu, C. Liu, L. Zhang, J. Wu, J. Sun, and Y. Liu, "Software composition analysis for vulnerability detection: An empirical study on java projects," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 960–972.

[65] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security symposium (USENIX security 19)*, 2019, pp. 995–1010.