Who are Vulnerability Reporters? A Large-scale Empirical Study on FLOSS

Nikolaos Alexopoulos
alexopoulos@tk.tu-darmstadt.de
Technical University of Darmstadt
Germany

Andrew Meneely
andy@se.rit.edu
Rochester Institute of Technology
USA

Dorian Arnouts
dorian-benedikt.arnouts@stud.tu-darmstadt.de
Technical University of Darmstadt
Germany

Max Mühlhäuser
max@informatik.tu-darmstadt.de
Technical University of Darmstadt
Germany

ABSTRACT

(Background) Software vulnerabilities pose a serious threat to the security of computer systems. Hence, there is a constant race for defenders to find and patch them before attackers are able to exploit them. Measuring different aspects of this process is important in order to better understand it and improve the odds for defenders.
(Aims) The human factor of the vulnerability discovery and patching process has received limited attention. Better knowledge of the characteristics of the people and organizations who discover and report security vulnerabilities can considerably enhance our understanding of the process, provide insights regarding the expended effort in vulnerability hunting, contribute to better security metrics, and help guide practical decisions regarding the strategy of projects to attract vulnerability researchers.
(Method) In this paper, we present what is, to the best of our knowledge, the first large-scale empirical study on the people and organizations who report vulnerabilities in popular FLOSS projects. Collecting data from a multitude of publicly available sources (NVD, bug-tracking platforms, vendor advisories, source code repositories), we create a dataset of reporter information for 2193 unique reporting entities of 4756 CVEs affecting the Mozilla suite, Apache httpd, the PHP interpreter, and the Linux kernel. We use the dataset to investigate several aspects of the vulnerability discovery process, specifically regarding the distribution of contributions, their temporal characteristics, and the motivations of reporters.
(Results) Among our results: around 80% of reports come from 20% of reporters; first time reporters are significant contributors to the yearly total in all 4 projects; productive reporters are specialized w.r.t. the project and vulnerability types; around half of all reports come from reporters acknowledging an affiliation.
(Conclusions) Projects depend both on a core of dedicated and productive reporters, and on small contributions from a large number of community reporters. The generalized Pareto principle (the $(1-p)/p$ law) can be used as a metric for the concentration of contributions in the vulnerability-reporting ecosystem of a project.

CCS CONCEPTS

• **Security and privacy** → **Software and application security**;
• **General and reference** → *Metrics*; *Empirical studies*.

KEYWORDS

vulnerability reporters, empirical study, security metrics

1 INTRODUCTION

Vulnerabilities or security bugs are a unique breed of bug. Rather than a deficiency in functionality, a vulnerability is an unintended feature that can potentially enable attackers to compromise the system. Discovering a vulnerability involves consideration not just of what the system was intended to *do*, but what is possible to *exploit*.

This attacker mindset is not ingrained in every developer, so Free/Libre and Open Source Software (FLOSS) projects must rely on the diverse skillsets of their community to both discover and responsibly disclose these vulnerabilities. Currently, report rates of vulnerabilities are at an all-time high, according to the National Vulnerability Database (NVD) [8], as well as seeing an increase in the diversity of types of vulnerabilities being reported [24]. The most prominent FLOSS projects today, such as those in this study, have had steady a stream of actionable vulnerability reports for well over a decade.

Furthermore, unlike a typical bug, knowledge of a vulnerability may have value for nefarious purposes, such as black-market sales or exploitation. There is motivation to keep vulnerabilities undisclosed.

The unique characteristics of the vulnerability discovery and patching process, coupled with the impact of exploits, have motivated the community to measure different aspects of the process, in order to better understand and improve it. For example, work on Vulnerability Discovery Models (VDMs) [2, 4, 15, 16] focuses on modeling and predicting the discovery rate of vulnerabilities after the release of a software product. Other studies look into general characteristics of vulnerabilities [1, 26], such as their severity and types, while a considerable amount of work focuses on characteristics of the patching process [11, 17, 23], and the effectiveness of bug bounty programs [10, 14, 19]. Despite the great efforts of the

ESEM '21, October 11–15, 2021, Bari, Italy

ACM ISBN 978-1-4503-8665-4/21/10…$15.00
https://doi.org/10.1145/3475716.3475783

community in investigating different aspects of the process, the "who"s of the process have not yet received enough attention, especially outside the closed environment of bug bounty programs. In this paper, we empirically investigate vulnerability reporters[1], trying to answer questions such as the following:

Who is reporting vulnerabilities to FLOSS projects? Are all reporters equally productive? Do more reports mean more reporters? How often are bug bounty programs involved? Are reporters specialized?

Answers to such questions will increase our understanding of the process, and can have important practical implications on security metrics and practices. FLOSS project coordinators need to know their community of vulnerability reporters if they want to maintain the health of the project long-term (since this depends on keeping them involved). FLOSS maintainers, as well, need to understand who they will be working with in the disclosure and fixing process. Regarding security metrics, the amount of effort expended in vulnerability finding is known to significantly influence the number of discoveries, and thus researchers have tried to quantify it. For example, effort-based VDMs [3, 29] assess the amount of this effort by the size of the installed user base of an application, or, integrating over time, by the amount of user hours/days since the release of an application. Although such approximations may be well suited for regular bugs, they are not directly related to the effort expended in the vulnerability discovery process, due to the unique nature of the latter (users will generally not bump into a vulnerability when using their favorite browser to browse the Internet). Investigating the landscape of vulnerability reporters can provide insights leading to better metrics of vulnerability-hunting effort with applications to VDMs and other measurement methodologies.

Overall, the goal of this study is to shed light on the human aspect of the vulnerability discovery process by analyzing historical vulnerability reporting data of mature and successful open source projects, in order to: (a) identify common trends and practices that can act as benchmarks of community engagement for new and existing projects, and (b) examine whether (and to what extent) empirical metrics of community engagement can act as indicators of software quality (w.r.t security).

We approach the problem by performing a large-scale empirical study on four big community-driven open-source projects: the Linux kernel, the Apache HTTP Server Project, the Mozilla suite (including Firefox and Thunderbird), and the PHP interpreter. We chose these projects as they are popular community-driven open-source projects with a large number of reported vulnerabilities and a transparent process for reporting and fixing them. We introduce a methodology to collect information regarding vulnerability reporters from a variety of sources, including the NVD, the projects' bug reporting platforms, and the projects' version control systems. We make the data collection and analysis scripts available under a free software license[2]. We also publish a snapshot of the dataset[3].

[1] Although there is a slight difference between vulnerability reporters and discoverers (we discuss this in Section 6), for the rest of the paper, we assume the terms reporters and discoverers to be interchangeable.

[2] https://github.com/nikalexo/vulnerability_reporters

[3] https://doi.org/10.6084/m9.figshare.14986830.v1 . Data are collected from publicly available sources whose purpose is to credit vulnerability reporters. Thus, the data were manifestly made public, and as such processing of the data does not infringe on the privacy rights of the involved individuals, e.g. see Article 9 of the GDPR.

Analysis focus: The dataset created by our methodology can be used to investigate a wide range of characteristics of the vulnerability discovery process. However, to obtain insights relevant to our above-stated goal, we structure our investigation focusing on the following areas of interest:

Distribution and temporal characteristics. We investigate whether contributions to the vulnerability discovery process (reported vulnerabilities) are evenly distributed among the contributing entities (decentralized) or significantly concentrated in a small number of reporters (centralized). This investigation can provide insights as to whether "many eyeballs" (a reference to Linus's law) is the primary contributing factor behind vulnerability discoveries, or if the proliferation of automated tools during the last decade has reduced the need for a large community of bug reporters, and thus made dedicated security resources (translating to a large number of vulnerabilities reported by a small number of entities) the primary contributing factor.

Another issue we investigate is whether productive reporters are specialized in specific types of vulnerabilities. Apart from reporter specialization, we also investigate temporal characteristics of the dataset. Do more CVEs mean more reporters over time? For how long do reporters stay engaged?

Motivations We explore the motivations of reporters in the projects of our study. Does a large portion of yearly reports come from reporters that have otherwise (e.g. via code commits or non-security bug reports) contributed to the project? Are most reports coming from reporters internal to the organization behind the project? Are they employed by other organizations? Did they receive bounties? Answers to these questions can provide insights into how established open source projects attract vulnerability reporters, which in turn can be translated to strategies for new and emerging projects.

Summary of results: We found that the distribution of reporter contributions can be described by a power law, meaning there are a few reporters responsible for most reports, while most reporters report only a few vulnerabilities. The number of reports is correlated with the number of reporters over time, while first time reporters account for a significant portion of the reports on a yearly basis. Regarding the period of engagement, for Mozilla, top reporters stay involved for a median of more than 8 years, significantly more compared to the other 3 projects. Also, reporters are specialized w.r.t. the type of vulnerability. Regarding motivations, bug bounty programs contribute significantly, yet bounty-related reports are a minority in all projects. Furthermore, a minority of reporters are affiliated to the organization behind the projects, while 35% of reporters have also committed to the project's repository, and 39% of reporters created a non-CVE bug in the project's bug tracking platform.

Key takeaways: Reports come from two groups of reporting entities. "Dedicated" entities that report a relatively large amount of CVEs, and a large number of other entities that report only a few CVEs. We interpret this result as a showcase of the complimentary nature of contributions resulting from (a) dedicated security planning and testing, and (b) the "many eyeballs" of the open source community. Furthermore, since we see first time reports contribute significantly to the yearly total of reports for a project, and reporters tend to stay involved for a relatively short period of time (except for

Mozilla), not only do projects need to establish an active community of vulnerability reporters, they also need to make sure that the project remains attractive to new vulnerability reporters over time. Finally, although it is inherently difficult to come up with quantitative metrics of the effectiveness of the vulnerability-hunting community of a project, we identify one such metric that can be of value. Since the distribution of reports per reporter can be described by a power law for the projects in our study, the generalized Pareto principle (the $(1-p)/p$ rule) can be used as a measure of the balance between dedicated resources and community engagement. We further discuss the implications of our results in Section 5.

2 RELATED WORK

There is a large number of previous works describing measurements relevant to the bug- and vulnerability-hunting process (e.g. [17, 20, 27]). However, in this section we focus on related work on vulnerability reporters.

User studies on discoverers. The most closely related work on vulnerability discoverers is based on conducting user studies. The work of Fang and Hafiz [9, 12] is especially relevant to ours. They ran an email-questionnaire user study collecting 127 responses from a variety of reporters of buffer overflow, SQL injection, and cross-site scripting vulnerabilities. Their study was the first to target reporters of vulnerabilities. They focused on the approach, tools and techniques of reporters of different types of vulnerabilities affecting a variety of software, as well as their disclosure practices. One of the results of their user study that our analysis corroborates is that reporters seem to be specialized w.r.t. the types of vulnerabilities they report. In their used study, a large proportion of the reporters who responded to the questionnaire, claimed to have reported a large number of vulnerabilities. Our data, on the other hand, indicates that most reporters report a small number of vulnerabilities. This is probably due to the inherent bias induced by the user study (it might be easier to contact a reporter with multiple reports, such a reporter may be more likely to respond to a request, etc.). Our approach of having a largely automated process, using publicly available data from a variety of sources, and striving for completeness of information, allows us to work on a much larger scale and investigate questions that are difficult to investigate otherwise (e.g. regarding differences between projects).

Another interesting user study was performed by Votipka et al. [28]. They used a semi-structured interview technique on a sample of 25 testers (from within a project) and white-hat hackers to investigate differences in the vulnerability-finding processes of the two groups. They concluded that the approaches of the two groups differ significantly and a project would benefit from the engagement of both groups in its community. They also pointed out that hacker engagement can also be achieved via non-financial rewards along with bug bounties. Our analysis confirms these results, as we see a strong contribution from both long-term reporters and "come-and-go" hackers, with bug bounties not necessarily being the incentive.

Bug bounty programs. Most previous large-scale studies providing insights into vulnerability hunting do so from within the bounds of bug bounty programs [10, 14, 19, 22, 30]. Although we show that bug bounty programs are a major contributor to some projects, their effect in the FLOSS ecosystem as a whole seems to be rather limited. Furthermore, none of these studies focused on the specific reporters, more on the structure of the bounty programs themselves.

Investigating Linus' law. Linus' law is the proposition that a large community of contributors and testers improves the quality of open source software. Eric Raymond first formulated the law as "Given enough eyeballs, all bugs are shallow" [25]. Meneely and Williams [20, 21] explored the correlation of developer collaboration metrics and discovered vulnerabilities for popular open source projects. They found that files co-developed by 2 or more independent developer groups were more likely to contain a vulnerability than files developed by collaborating contributors. Also, Linux kernel files with changes from 9 or more developers were 16 times more likely to have a vulnerability. These studies investigated the "developer" aspect of Linus' law. Our study, on the other hand, provides insights regarding the "vulnerability-reporters" aspect of the law.

3 METHODOLOGY

The following sections describe our methodology for constructing what is, to the best of our knowledge, the largest and most complete dataset of vulnerability reporters in existence. To do this, we had to overcome challenges related to collecting and correctly managing data from a large number of disparate sources (e.g. aliases, repetitions, errors). It should be noted that the methodology is encoded in scripts that automate the process (as it will become clearer below). As a result, although we focus on four specific projects, data can be generated for other projects with relatively little additional effort. We also note that the process is best-effort and may still be subject to errors, but the dataset, as well as all our scripts are open and publicly available. We consider the methodology for constructing the dataset of general importance, and as such, an independent contribution.

3.1 Projects in this study

Different software projects/development teams use diverse approaches regarding reporting, patching and documenting bugs. Their processes differ slightly in the way they handle vulnerabilities.

– **Mozilla suite:** The Mozilla suite includes the code for Mozilla products such as the popular Firefox web browser and Thunderbird email client. The suite is handled as a whole since these products share a significant underlying codebase (the *Core* component). Both security and non-security-related bugs are typically reported and handled in the Mozilla Bug Tracking System which is implemented as a Bugzilla instance (although security bugs can also be sent to the security team via email). Commit messages in the repository that fix a bug should include the bug id (identification number). The Mozilla security community also publishes security advisories providing more information on the fixed security bugs[4].

– **Apache httpd:** Apache httpd is a popular web server with a market share of over 33%[5]. The Apache security policy[6] states that security bugs are to be reported in a dedicated private mailing list

[4] https://www.mozilla.org/en-US/security/advisories/

[5] https://w3techs.com/technologies/details/ws-apache. Market share: 33.6% on 14th May 2021.

[6] http://www.apache.org/security/

and are handled differently than normal bugs (which are handled via a Bugzilla bug tracking system). The Apache security team also publishes security advisories for all fixed vulnerabilities affecting released versions.

– PHP: PHP is the most popular server-side programming language in the web at the time of writing[7]. The standard PHP interpreter is a community-driven project written almost entirely in C using git as the version control system. Security and non-security bugs are both handled by the PHP bug-tracking system (albeit in different ways and with different priorities and privacy rules). Developers are requested to add the bug number prepended by a # in the commit message of the fix.

– Linux: The Linux kernel is arguably the most impactful community-driven project in history, thus being the prototypical example of Eric Raymond's bazaar [25] model of community-driven software development. Although most contributors to the project are no longer volunteers (since they are employed by several organizations to work on the kernel), the general concept of the bazaar, i.e. decentralized and lightly coupled development, still generally holds. Security bugs in Linux are to be reported to the kernel security team via email and are handled separately from normal bugs which are handled via a combination of (subsystem-specific and general) mailing lists and a Bugzilla instance.

3.2 Data sources

Pursuing the goals of the study requires the collection of several data points for the four FLOSS projects investigated. We move on to document the data collection process for each data type:

– CVE entries: We use the cve-search tool [8] to maintain a local copy of the CVE information available at the NVD.

– Vulnerability reporters: Information on who first reported a vulnerability is not available in the NVD entries. Upon further investigation, we located the following sources of information about reporters. For Mozilla products, relevant information is available (a) at the "Reporter" field of Mozilla Security Advisories[9], (b) at the "Reporter" field of the associated bug report. For Apache httpd, relevant information is available at the "Acknowledgements" field of the Apache httpd security advisories[10]. For PHP, relevant information is available at the "Reporter" field of the associated bug report. For the Linux kernel, relevant information is available (a) in the description of Debian Security Advisories[11] (for vulnerabilities that affected the kernel version included in the stable Debian distribution at that point in time), (b) at the "Credit" field of the SecurityFocus database[12], as well as in Ubuntu Security Notices and the Red Hat Linux bug tracking system. The four latter sources of information maintain such references in general for all projects, and are therefore considered for all of them. Although the information of the sources above is not included in the NVD, references to these sources are often included, making the mining process easier.

– Bug reports: We mined all bug reports from the projects' bug tracking systems (BTSs). Mozilla, Apache, and the Linux kernel use a Bugzilla BTS, and therefore we used the provided rest API, while PHP uses a custom BTS, so we scraped its html pages. In all cases, since the amount of data is large (several GB), bulk http requests will time out, and therefore we used per-month requests.

– Developer data: To obtain data regarding the developers involved in the respective projects, we cloned the github mirrors of the projects' repositories.

– Social network data: To enhance our dataset w.r.t. the affiliation of reporters, we extracted relevant information from the public profiles of reporters in the LinkedIn professional networking site. More about the approach can be found in the following section.

– Bug bounty data: We mined the publicly visible portion of the Hackerone[13] bug bounty platform. The relevant bounty programs for the chosen projects within Hackerone are the *Apache httpd (IBB)*[14], the *PHP (IBB)*[15] (indefinitely suspended since October 2020), and *The Internet*[16] programs (all within the scope of the *Internet Bug Bounty (IBB)* program). We found bounty information about a total of 161 CVEs in our dataset (4 for Linux, 142 for PHP and 15 for Apache), including reporter information for 1 Linux CVE and 35 PHP CVEs, for which no reporters had been retrieved from the other sources.

All data were collected from publicly available sources and the collection scripts as an open source project, ensuring reproducibility of our results. A graphical representation summarizing the data points collected and the links between them is provided in Figure 6 of Appendix A. A summary of the number of data points collected can be found in Table 1 (due to space reasons and its relatively small impact, the numbers from Hackerone are omitted from the table and are available in the corresponding paragraph above).

Project	CVEs	PA	DSA	USN	RH	sf	BTS	Total
Mozilla	2 195	992	218	369	1 040	1 421	209	2 193
Apache	249	71	27	38	12	177	—	197
PHP	638	—	37	31	18	393	198	603
Linux	2 566	201	533	562	473	1 555	—	1962

Table 1: Breakdown of information sources. PA stands for project-specific advisories, DSA for Debian Security Advisories, USN for Ubuntu Security Notices, RH for the Red Hat Linux BTS, sf for Symantec's securityfocus.com, BTS for the project's bug tracking system. The last column is the total unique CVEs that we could obtain any information for (union of all sources).

3.3 Data cleaning and pre-processing

Data cleaning was a laborious process requiring considerable manual work. We strove to make this manual effort a one-time job by encoding the logic of the process into reusable and extendable scripts.

[7] https://w3techs.com/technologies/details/pl-php. Market share: 79.2% on 14th May 2021.
[8] https://github.com/cve-search/cve-search
[9] See Footnote 4
[10] https://httpd.apache.org/security_report.html
[11] https://www.debian.org/security/
[12] https://www.securityfocus.com/
[13] https://www.hackerone.com/
[14] https://hackerone.com/ibb-apache?type=team
[15] https://hackerone.com/ibb-php?type=team
[16] https://hackerone.com/internet?type=team

3.3.1 CVEs affecting projects. We found out that attributing CVEs to the affected projects (code-bases) is not a trivial task. Although the NVD provides information following the Common Platform Enumeration (CPE) Dictionary, some cleaning was necessary (due to errors in the NVD). First, we filtered CVEs by the CPE identifier for each of the projects. For Mozilla and Apache, some CVEs reported in the vendors' security advisories were missing from the resulting set, and were subsequently added. For the Linux kernel, there were several issues that were resolved:

– We noticed that the set of CVEs returned when searching with the CPE of the kernel (`linux:linux_kernel`) is missing some CVEs. On the other hand, a free text search with the keyword "Linux kernel" in CVE summaries returns quite some noise (CVEs that do not correspond to the kernel but mention it). To overcome this issue, we only added CVEs returned by the keyword search that included a reference to the git repository of the kernel. This yielded a total of 6 additional CVEs, e.g. CVE-2007-6712 that includes a CPE for `kernel:linux_kernel` (probably a mistake). The low number of additional CVEs indicates that the effect of this type of NVD errors in previous studies would be very limited.

– We noticed that some NVD entries utilizing the AND logical operator to specify vulnerable configurations erroneously labeled the operating system part of the description as vulnerable. For example, CVE-2015-0312 describes an Adobe Flash Player vulnerability that affects certain versions of the Player running on Linux (and other versions of the Player running on Windows). This is not a kernel bug, yet the json feed of the NVD wrongly labels the CPE of the operating system as "vulnerable". We filtered out 35 such instances by keeping only CVEs that refer to a kernel git repository when both another application and the kernel are included in the list of affected CPEs.

3.3.2 Reporter information. Cleaning reporter data was a multi-step process, consisting of the following:

(i.) Extracting reporter names via regular expressions, such as `.*(?= discovered an issue)` for each source (different regular expressions may be needed for each source). At the end of this step, we have a list of strings $[s_1, \ldots, s_n]$, where n is the number of reporter sources (in our case the 6 sources of Table 1). E.g. source$_i$: `Alice and Bob discovered an issue in the Linux kernel...` $\rightarrow s_i$ = `Alice and Bob`.

(ii.) Splitting each of the strings to pieces at the `'and'` and `','` predicates. E.g. `Alice and Bob` → [`Alice`, `Bob`].

(iii.) Keeping track of affiliations in a "smart" way using regular expressions. If an affiliation follows after several names (identifiers), then the affiliation corresponds to all of the preceding names. E.g. `Alice and Bob from Mozilla` → aff.(`Alice`) = aff.(`Bob`) = `Mozilla`.

(iv.) Keeping track of email addresses and twitter handles using regular expressions. We keep a list of all emails associated with a reporter as aliases for the same entity, since people are known to use various email addresses when filing reports or committing code [5].

(v.) Collecting reporters from all sources into a list, and removing remaining natural language phrases (that slipped through the initial matching of step (i.)), like `found by...`. Then, removing duplicates, after first stripping the strings of leading and trailing whitespaces and special characters like full stops. At this point we have a list of reporters for each CVE.

(vi.) Removing "reporters" matching generic terms (e.g. `the vendor`, `unknown`) when other reporters (not matching these terms) exist for the same CVE.

(vii.) Merging reporters with the same email address or twitter handle. E.g. [`Alice <alice@alice.com>`, `alice@alice.com`].

(viii.) Merging "similar" reporter names, based on their Levenshtein distance, using the FuzzyWuzzy python library[17]. E.g. [`Michael Jordan`, `Michael Jordon`][18]. We merge entries over 4 characters that are 90% or more similar (based on the similarity ratio metric of FuzzyWuzzy). Manual inspection of all matches produced by this (rather conservative) rule revealed no false positives (according to our best judgment).

(ix.) Manual fixes (**This is the only step with no automation scripts**). E.g. reporter names like `Bug report XYZ` were deleted. Note that manual post-processing is most likely necessary in the general case because reporter nicknames and entries like the one above may be hard to tell apart automatically.

(x.) Mining additional affiliation information. To enhance our dataset w.r.t. the affiliation of reporters and to capture temporal changes in this field, we use information from LinkedIn. Since this step of the process involves significant manual effort to clean and validate the discovered profiles, we narrowed the scope to the top 100 reporters (with most reports) in our dataset. As these top reporters contribute significantly to the total number of reports, we judged this to be a reasonable trade-off between improving our dataset and manual effort. Attempts to fully automate the process and mine information for all reporters resulted in non-negligible noise in the returned data. For each reporter in the top 100 (reporting entities that we had not characterized as teams or organizations), we used the "people search" function provided in LinkedIn's interface, searching for each person's name followed by the key-phrase `computer security`. We then cleaned the returned results by only considering individuals working in related fields (e.g. Computer & Network Security, Computer Software, Internet) and filtering out profiles that advertise Physical Security as a skill. This brought the number of available profiles to 48. We further investigated the accuracy of the mappings by manually searching for characteristics that point to a security researcher, e.g. phrases like "developed fuzzing tools" or descriptions like "Ethical hacker". Additionally, we cross-referenced information from the profiles with information that we already possessed in our dataset regarding the affiliation of reporters, as well as information from associated presentations, blog posts, etc. that were harnessed via web searches. The result was that 40 out of the 48 profiles were classified as correct, while 8 of them as incorrect. For 7 out of the 8 incorrect cases, we were able to manually identify the correct profile, while for the remaining case we could not. At the end, we had manually verified profiles for 47 reporters that were within the top 100 most productive reporters of our dataset.

The cut-off point for the data presented in this study is the 13th of January 2021. At the end of the cleaning process, we ended up with

[17] https://github.com/seatgeek/fuzzywuzzy

[18] The correct spelling is the second. Look closely at https://www.mozilla.org/en-US/security/advisories/mfsa2011-41/ to see the typo.

2 193 unique reporting entities (2 060 human reporters with the rest being "organizations" or "teams") associated with 4 756 CVEs. Considering that the total number of CVEs for the four projects is 5 648, our dataset has at least one reporter for ~ 84% of all CVEs, a figure that surpassed our initial expectations. A breakdown of the final dataset per project is provided in Table 2.

Project	CVEs	w/ reporter(s)	coverage
Mozilla suite	2 195	2 085	95 %
Apache httpd	249	196	79 %
PHP	638	520	81 %
Linux (kernel)	2 566	1 955	76 %

Table 2: Breakdown of reporter coverage

4 RESULTS

In the following sections, we present the analysis of the dataset that we constructed. We set off by investigating several characteristics of the distribution of reporters, both statically and over time. We then proceed to investigate indicators of reporter motivations. As stated earlier, the goal of the analysis is (a) to identify patterns and characteristics that can be used as a benchmark for the health and extent of the vulnerability hunting communities of open source projects, and (b) to provide indications on whether individual community engagement metrics can be used as indicators of health. We use raw numbers (tables), suitable plots and empirical domain-specific reasoning to assess the strength of relationships in our data. Furthermore, we use suitable statistical tests with a typical 5% significance level to assess the statistical significance of our results, when required.

4.1 Distributions and temporal characteristics

(Concentration of reports.) We begin our analysis by investigating the distribution of the total number of reports per each reporting entity. With this investigation we explore whether the vulnerability reporting ecosystem is more centralized (suggesting more dedicated resource allocation) or decentralized (suggesting more community engagement) for the projects under study. Figure 1 shows

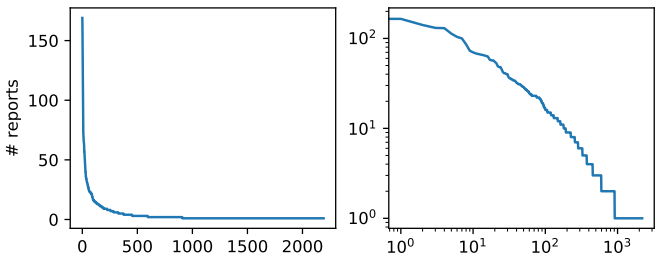

Figure 1: Distribution of reports per reporter in linear and double logarithmic scales (x axis: reporters ordered by number of reports).

the distribution of reports per reporter for all projects. Observing an almost straight line in double logarithmic axes in the second plot of Figure 1, which is an indication of a possible power-law distribution, we moved on to investigate statistically. Power laws are heavy-tailed distributions, indicative of preferential attachment behavior, which have received interest in several fields, including software engineering [18]. We followed the seminal methodology for fitting heavy-tailed distributions of Clauset et al. [7] and used the Kolmogorov-Smirnov statistic to compare possible alternative distributions. Considering a 95% confidence interval, we found that a truncated power law (power law with exponential cut-off) is the most likely statistical fit when considering all the projects together and when considering Mozilla CVEs, while a pure power-law is a good fit for the other projects (Linux, PHP and Apache) when considered individually. Detailed plots of the fits provided by candidate distributions can be found in Figure 7 in the Appendix.

The "80/20 rule" or the Pareto principle, where 80% of the contributions (e.g. wealth, bugs, CVEs) are attributed to 20% of the population is often used as an example to portray the behavior of a power law distribution. However, power laws can be more balanced or imbalanced. The generalized principle can be described by the "$(1-p)/p$ law"[13]. In fact this ratio describes the power law distribution uniquely and provides a metric for the concentration of contributions. The *CVE report concentration metric X/Y* (read as "Y% of reporters contributed X% of reports") for the projects in this study is as follows: Mozilla: 78/22, Apache: 59/41, PHP: 70/30, Linux: 72/28. We see that reports for Mozilla are more concentrated in a "core" group of reporters in comparison to the other projects. On the other side of the spectrum, reports for Apache are more balanced. We discuss how this metric can be used to describe the balance between dedicated reporters and the "many eyes" of Linus' law in Section 5.

(CVEs and reporters over time.) Figure 2 shows several temporal characteristics of the reporting process. The second plot of Figure 2 shows that the number of distinct reporters per year generally follows an increasing trend, as do the number of vulnerabilities reported (first plot). Note that to assign CVEs to years, we used the 'YEAR' portion of the CVE identifier (the YYYY of a CVE identifier that has the format CVE-YYYY-NNNNN). According to MITRE, "the YYYY portion is the year that the CVE ID was assigned OR the year the vulnerability was made public (if before the CVE ID was assigned)"[19]. Since we collected our data in early 2021, the analysis for some 2020 entries had not been published yet in the NVD, explaining the lower number of entries for that year compared to previous ones. The average number of reports per reporter (third plot) shows no noteworthy trend and varies between 1 and 4 reports for all projects, with the Linux kernel having a slightly higher average number during the last years. Note that a CVE can have multiple reporters, thus this number is not bounded by 1. The fourth plot shows an interesting phenomenon: in particular for the Linux kernel and Mozilla there is a constant influx of new reporters ranging from 40 to 90 per year (roughly half of all reporters for each calendar year are first-time reporters). Thus, new reporters are significant contributors to the process.

(Period of engagement.) Since, as we showed, the distribution of reports per reporter is heavy-tailed (most reporters have reported only a few bugs and few reporters have reported most bugs), for this

[19]https://cve.mitre.org/about/faqs.html#year_portion_of_cve_id

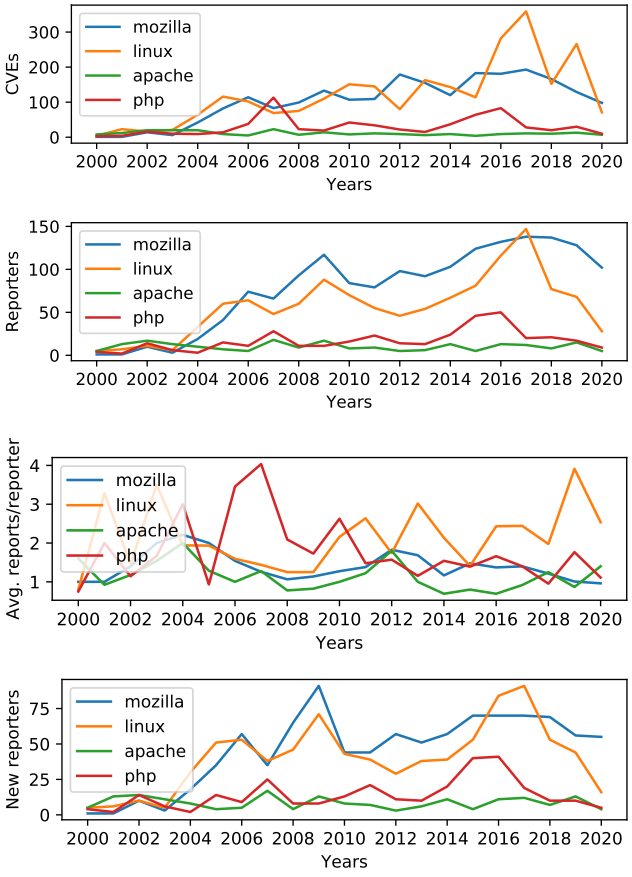

Figure 2: From top to bottom (all per year): # of CVEs, # of reporters, average reports per reporter, # of new reporters. Note that the drop for 2020 observed in all plots is due to the time of data collection being in early 2021 (information for some 2020 vulnerabilities may not be published yet)

part of our investigation we focus on this heavy tail by looking into the top 10 reporters for each project. Figure 3 includes 3 box plots. Observing the first plot, we can say that there is a considerable variation between projects regarding how long their top reporters remained engaged. The median for Mozilla is more than 8 years, whereas for Linux only 3. Note that a reporter's last report stands for their last report until now; we can not know if they make new reports in the future. Therefore, the first plot gives a lower bound on the duration of engagement. In this paper, we do not investigate phenomena regarding potential periodic reporter behavior, but our dataset can be used to study such hypotheses in the future. The second plot shows that, until now, only for Mozilla can we make the argument that there is a learning curve of about 3 years before reporter productivity peaks. Also considering the observations from the first plot, reporters in the other projects seem to generally not remain engaged long enough for effects of a learning curve to manifest themselves. The third plot of Figure 3 shows the time between the year of the peak performance for each of the top reporters and their last report to date. For Mozilla this time is 4

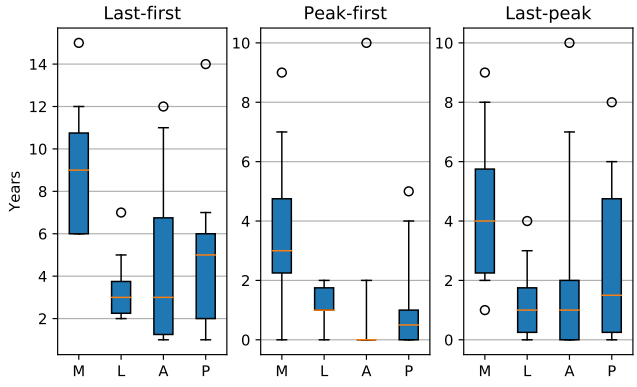

Figure 3: For the top 10 reporters for each project: time in years between (a) their first and more recent report until now (last), (b) their first report and their peak year, and (c) their peak year and their last report until now ([5,95] whiskers). Letters in the x axis are the initials of the corresponding projects.

years (median), while for the other projects, it is one year. Overall, we see that Mozilla has a more stable base of long-term regular contributors of vulnerabilities, while for the other projects, even their top contributors stay engaged only for 2-5 years.

(Reporter specialization.) Only 83 reporters (out of a total of more than 2 000) have reported a vulnerability for two or more of the projects. This number goes down to 14 entities who reported vulnerabilities for three or more of the projects, and only one entity (iDefense, which was a bug-bounty program, so probably includes multiple entities) reported a vulnerability for all four of the projects under investigation. Thus, reporters to multiple of these four FLOSS projects are rare, i.e. reporters are generally specialized w.r.t. the project they are testing.

To investigate reporter specialization w.r.t. CVE types, we extract the Common Weakness Enumeration (CWE) number of each CVE from the information available at the NVD. Since some CWE types are closely related or have changed over time, and since we are interested in a more high-level classification of vulnerability types, we follow the approach of [6] to map each CWE to one of the 6 following high-level categories:

1. Memory and Resource Management (e.g. CWE-119: "Improper Restriction of Operations within the Bounds of a Memory Buffer")
2. Input Validation and Sanitization (e.g. CWE-20: "Improper Input Validation")
3. Code Development Quality (e.g. CWE-369: "Divide By Zero")
4. Security Measures (e.g. category CWE-310: "Cryptographic Issues")
5. Concurrency (e.g. CWE-362: "Race Condition")
6. Other

We provide an overview of the number of different CWE types and categories found by reporters with more than 20 reports, in the box plots in Figure 4. Half of all reporters have reported issues of 10 or more different CWE types, or alternatively 4 or more categories. Half of all reporters have a specific CWE type that accounts for

more than 30% of their reports, or alternatively they have a specific higher-level category that accounts for 45% or more of their reports.

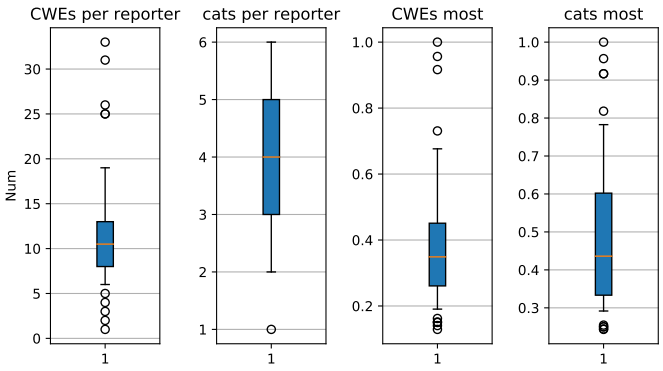

Figure 4: Number of different CWEs and categories (cats) for each reporter. Third column: for each reporter, what portion of their reports falls into the CWE/category with the most reports (dominant).

To investigate whether the distribution of reporters per category varies between reporters, we compared the distribution of categories of each individual reporter with at least 20 reports, with the overall distribution of categories for the project that the reports concern. We only looked into reporters with at least 20 reports in order to be able to make statistical arguments about the distributions, and we looked for reports in each project individually to account for different categories being more common in different projects (e.g. the number of concurrency bugs is negligible in projects other than the Linux kernel). We employed a Chi-squared test to assess whether the distribution of categories corresponding to reports from a given reporter deviates significantly ($p<0.05$) from the expected (taken for the project as a whole). Then, for those reporters whose distribution deviates, we checked if more than half of their reports fall into the same category, signaling a specific focus. The results are summarized in Table 3.

Project	reporters > 20	# deviate	# focused
Mozilla suite	49	33	21
Apache httpd	0	0	0
PHP	4	1	1
Linux (kernel)	18	16	13

Table 3: Deviations from expected categories. From left to right: reporters with more than 20 reports; out of them, reporters with a deviation w.r.t. categories; out of the latter, reporters with a specific focus category.

Due to the low number of observations for Apache and PHP, we focus on Mozilla and Linux. For Mozilla, out of the 21 reporters who exhibited a particular focus, 13 focused on memory-related issues (category 1), 3 focused on issues related to security measures (category 4), while 5 focused on "Other" issues. For Linux, out of the 13 reporters with a significant focus, 3 focused on memory-related issues while 10 focused on issues related to security measures. Overall, results indicate that a significant portion of the most productive reporters are specialized in a specific category of vulnerabilities and that there are two main categories of specialization: memory and security measures. We can conjecture that reporters specialized in memory issues are the ones who develop and operate several fuzzing mechanisms that have become popular during the last years, while reporters specialized in security measures are the ones looking specifically for issues that do not cause crashes (and therefore cannot be detected by usual fuzzing tools); rather these issues have to do with permissions, cryptographic implementations, data leakage, etc.

4.2 Motivations

(Bug bounty programs.) Table 4 shows the number of CVEs associated with a bounty for each project. Because we want to gain insights into the incentives of reporters, we provide a low estimate, which corresponds to the confirmed number of CVEs for which a bounty was awarded, and a high estimate, which corresponds to the number of CVEs reported by a reporter who was awarded a bounty for at least one of their reports.

Project	CVEs	bounties low-high	% low-high
Mozilla suite	2 195	589–1 206	27–55
Apache httpd	249	15-16	6-6
PHP	638	142-208	22-33
Linux (kernel)	2 566	4-50	0-2

Table 4: Reports with bounties per project. Includes a low estimate of confirmed bounties given and a high estimate of reports by a reporter who has received a bounty at least once.

We can observe that the effects of bug bounties vary greatly between the projects, with Mozilla, being one of the pioneers of bug bounties in the FLOSS community, benefiting greatly from the program. However, in each of these projects, bounties were not given to more than half of the reporters. Furthermore, the Linux kernel does not depend on bounties for vulnerability discoveries at all and still maintains a consistent influx of new reports and new reporters. Thus, while bug bounty programs are helpful, reporters have additional motivations beyond financial gain.

(Affiliations.) We linked reporters to affiliations based on (a) information collected from the sources of Section 3.2, (b) additional information from reporters' online professional profiles – when applicable (as described in Section 3.3). A vulnerability is internal iff (a) the reporter *matches* the organization (e.g. "Mozilla" or "Mozilla Corporation", etc. for the Mozilla suite), or (b) at least one of its reporters has an affiliation matching the organization (when temporal data for affiliations is available, e.g. via information from an online profile, then an additional constraint for the time of employment is applied). Looking at Figure 5, we observe differences between the projects. A significant portion of Mozilla CVEs originated from within the organization (33%), while this portion is much smaller for the other projects (<10%). The most significant external contributors to Mozilla reports are Tencent and Google. A number of

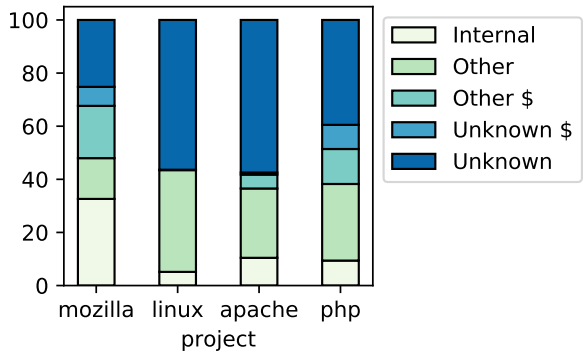

Figure 5: Affiliations associated to CVEs. Internal is for reporters affiliated to the organization behind the project, Other is for reporters affiliated to other organizations, and Unknown is for CVEs which could not be associated to an affiliation. We mark with $ the subsets of the two latter categories that were awarded bug bounties.

companies/organizations contribute significantly to the security of the Linux kernel, with the most notable being Google, Qihoo 360, and Red Hat. Also for the kernel, a comparably large number of CVEs could not be associated with an affiliation, potentially pointing to a higher engagement of hackers working on a volunteer basis.

(Commits and bug reports.) We found at least one commit in any of the projects for 730 out of a total of 2060 human reporters in our dataset (35%). 559 reporters have made 10 or more commits (27%), while 382 (19%) have made 100 or more commits. While reporters are not regular committers in their majority, a significant percentage is actively contributing to the codebase.

Project	Reporters	Bug Reporters (%)	Median
All	2060	810 (39%)	5
Mozilla suite	917	394 (43%)	8
Apache httpd	160	34 (21%)	4
PHP	277	65 (23%)	3
Linux (kernel)	790	151 (19%)	2

Table 5: Percentage of vulnerability reporters with non-cve bug reports and median of such bugs per reporter.

One hypothesis we could make is that most vulnerability reporters find and report other kinds of bugs (non-CVE) as by-products of the process, and that these bugs are somehow different than other non-security bugs. If this hypothesis is true, then related metrics could be used to measure vulnerability-hunting effort. In the following, we investigate this hypothesis.

A significant portion (39%) of vulnerability reporters created at least one non-CVE bug in one of the projects' bug tracking platforms. Between the projects, percentages vary between 19% and 43% (Table 5). For the Mozilla Suite, this value is clearly higher with 43% (19-23% for the other projects). Mozilla reporters also report more non-CVE bugs than others (median for Mozilla is 8 compared to 2-4 for the other projects).

Project	Severe		Resolved		Keywords	
	reps	rest	reps	rest	reps	rest
Mozilla suite	**20%**	13%	**40%**	34%	**23%**	11%
Apache httpd	12%	**20%**	**47%**	32%	**15%**	10%
PHP	-	-	**46%**	11%	**22%**	12%
Linux (kernel)	11%	**16%**	**31%**	27%	**21%**	15%

Table 6: Differences between bugs by vulnerability reporters (reps) and others (rest) as a percentage of the total for each class. All statistically significant ($p < 0.05$ for the Chi-squared test). Severe are bugs with *critical*, *major* or *high* severity (except for PHP for which no severity field exists). Resolved are bugs marked *fixed* in bug reports (except for PHP for which no such field exists, and therefore we considered bugs with associated fixing commits).

To analyze if bugs opened by vulnerability reporters differ from bugs by others, we used the fields *severity* and *resolution* of bug reports (hypothesizing that vulnerability reporters report more severe bugs that are also more likely to be resolved). Additionally we searched for the keywords *memory*, *crash* and *security* in bug descriptions, assuming that these words occur more often for more serious bugs (hypothesizing that vulnerability reporters mostly create such bug reports). The results are summarized in Table 6. We observe a higher percentage of severe bugs only for Mozilla, with the tendency reversed for Apache and Linux. Regarding *resolution*, for each of the projects, a higher percentage of bugs created by reporters are fixed. For keywords, reporters of all projects have created a (slightly) higher percentage of bugs mentioning *memory*, *crash* or *security*. Overall, although we identified statistically significant differences between bugs created by vulnerability reporters compared to bugs created by others, the magnitude of the differences is small. Additionally, as stated earlier, only a minority of vulnerability reporters also reported other bugs, with the percentage varying significantly between projects. As a conclusion based on the two previous statements, we cannot support the hypothesis that by-products of vulnerability hunting in a project's bug tracking platform can be identified and utilized, in order to assess the expended vulnerability hunting effort.

5 DISCUSSION AND IMPLICATIONS

In this section, we discuss some implications of our results.

- **Observations as a benchmark and recommendations for new projects:** We saw that the FLOSS projects in our study depend both on a dedicated set of "core" reporters with many reports, as well as on one-off contributions from a large set of reporters ("many eyeballs"). Therefore, projects should make sure that they attract both types of reporters, e.g. via having dedicated security resources and planning in addition to engaging with volunteer hackers. Furthermore, FLOSS projects depend on attracting a steady influx of new reporters into their communities, therefore projects should find ways to keep attracting new reporters. Depending on the type of the project, the motivations of reporters may vary, and as a result the best ways to attract them may also vary. If the project is used as a core part of the operation of other organizations (e.g. the Linux kernel), then these organizations may contribute to creating the

dedicated "core" of reporters. Otherwise, more investment of own resources or bug bounty programs may be needed (e.g. as in the case of Mozilla). Also, productive reporters seem to be specialized w.r.t. the types of vulnerabilities they are looking for (especially regarding memory-related issues and security issues). Thus, projects should look to attract a diverse set of vulnerability reporters.

- **Community engagement metrics as indicators of quality:** An ideal numerical "vulnerability-hunting effort" metric would provide the community with a powerful tool in order to measure the security of software projects. The complexities and per project peculiarities that we showcased in this paper suggest that a singular such metric may be unattainable. However, some of the metrics we investigated in this paper – in combination with qualitative characteristics discussed in the previous paragraph – may provide useful indications regarding the quality and intensity of the vulnerability-hunting process for a project. Specifically, the $(1-p)/p$ ratio of the distribution of the number of reports per reporter (generalized Pareto principle) can be used as a metric for the concentration of contributions in a project. We saw that all projects in our investigation showed some imbalance w.r.t how many reports are contributed by each reporter. There are many factors (familiarity with a project, use of automated tools, discovery of vulnerability patterns) that could support a preferential attachment mechanism that creates a heavy-tailed distribution of reports per reporter. A very high concentration of reports (e.g. 90/10) would indicate low participation of the "many eyeballs" of a community to the vulnerability reporting process. A very low concentration (e.g. 50/50) would indicate a lack of dedicated resource allocation (or lacking effectiveness of those resources).

Our results suggest that metrics such as the popularity of a project, or the number of developers, do not directly relate to the number of vulnerability reporters, since most reporters were not otherwise involved in those projects. Anecdotal evidence, such as the Linux foundation's executive director's Jim Zemlin commentary on the Heartbleed vulnerability of OpenSSL: "In these cases the eyeballs weren't really looking," seem to support this suggestion. Merely being a popular project does not automatically translate to having an active and adequate community of vulnerability reporters. This would also mean that the use of the popularity of a project (e.g. install base) as a metric for vulnerability-hunting effort (as in some effort-based vulnerability discovery models) may lead to uninformative or even misleading results. A detailed analysis, as the one presented in this paper requires human involvement for the interpretation of its result, yet we believe it provides more actionable and valuable indications regarding the health of the vulnerability-hunting community of a project. Thus, we believe that the methodology presented in this paper can also act as a "general profile" of the health of a project's vulnerability reporting ecosystem.

6 THREATS TO VALIDITY

Below we document some possible sources of bias in our results.

- **Best-effort:** We collected our data from multiple sources and underwent a rigorous cleaning process. Understandably, this process was best-effort and we welcome corrections by the community on our publicly available dataset.
- **Inherent noise and incomplete data:** Our information sources (NVD, security advisories, bug reports, etc.) are manually curated and therefore subject to errors and omissions. Furthermore, we focused our investigation on vulnerabilities that received a CVE identifier. Although the CVE database is supposed to "fully cover" the projects in our study, it is known that a significant amount of vulnerabilities do not get a CVE. Although we do not expect these vulnerabilities to differ significantly (w.r.t. the characteristics we studied) to the ones in our study, future efforts can be driven towards expanding our dataset to include more vulnerabilities.
- **Generalization:** We investigated four popular FLOSS projects. In particular, these are among the largest FLOSS projects with longevity, so these results may not apply to smaller, newer FLOSS projects. We saw various differences in the reporter characteristics for each project, and therefore additional studies are required to have a more general understanding. However, our methodology is largely automated and should be applicable to any other FLOSS projects that keep accurate records of their vulnerability reporters, which can significantly reduce the overhead of future studies.
- **Reporters vs discoverers:** For the discussion of the implication of our results, we assumed that the people/organizations who get credited with reporting a vulnerability are the ones that discovered it. We believe this to be a valid assumption macroscopically, although we note that it may not be universally true.

7 SUMMARY AND CONCLUSION

We performed the first (to the best of our knowledge) large-scale study on FLOSS vulnerability reporters, going beyond bug-bounty programs. We investigated several aspects regarding the trends, backgrounds, motivations, and behaviors of vulnerability reporters. We identified qualitative characteristics that can act as benchmarks for healthy vulnerability-finding ecosystems. We also identified a quantitative metric that can provide indications regarding the health of the vulnerability-hunting ecosystem of a project, although further study with even bigger datasets is required. As an independent contribution, we demonstrated that characteristics of vulnerability reporters can be studied through information mined from several publicly available online sources. Our approach is mostly automated and our data and scripts are open to the public, so future researchers or FLOSS project coordinators can further build upon this study.

In conclusion, based on our results, we believe that individual metrics for vulnerability-hunting effort without context will fail to capture the unique characteristics of the process. On the other hand, comprehensive case studies, shedding light on multiple potentially co-dependent aspects of the process, can provide useful insights. The methodology we present in this paper provides a blueprint for such approaches.

ACKNOWLEDGMENTS

This work has been co-funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

REFERENCES

[1] Nikolaos Alexopoulos, Sheikh Mahbub Habib, Steffen Schulz, and Max Mühlhäuser. 2020. The Tip of the Iceberg: On the Merits of Finding Security Bugs. *ACM Trans. Priv. Secur.* 24, 1, Article 3 (Sept. 2020), 33 pages. https://doi.org/10.1145/3406112

[2] Omar H. Alhazmi and Yashwant K. Malaiya. 2005. Modeling the Vulnerability Discovery Process. In *16th International Symposium on Software Reliability Engineering (ISSRE 2005), 8-11 November 2005, Chicago, IL, USA*. IEEE Computer Society, 129–138. https://doi.org/10.1109/ISSRE.2005.30

[3] Omar H Alhazmi and Yashwant K Malaiya. 2005. Quantitative vulnerability assessment of systems software. In *Reliability and Maintainability Symposium, 2005. Proceedings. Annual.* IEEE, 615–620.

[4] Omar H. Alhazmi and Yashwant K. Malaiya. 2006. Measuring and Enhancing Prediction Capabilities of Vulnerability Discovery Models for Apache and IIS HTTP Servers. In *17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November 2006, Raleigh, North Carolina, USA*. IEEE Computer Society, 343–352. https://doi.org/10.1109/ISSRE.2006.26

[5] Christian Bird, Alex Gourley, Premkumar T. Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006*, Stephan Diehl, Harald C. Gall, and Ahmed E. Hassan (Eds.). ACM, 137–143. https://doi.org/10.1145/1137983.1138016

[6] Manuel Brack. 2020. A large-scale statisticalAnalysis of Vulnerability Lifetimes in Open-Source Software. Bachelor thesis. Technical University of Darmstadt.

[7] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[8] CVE Details. 2020. *Browse vulnerabilities by Date.* Retrieved 2020-20-05 from https://www.cvedetails.com/browse-by-date.php

[9] Ming Fang and Munawar Hafiz. 2014. Discovering buffer overflow vulnerabilities in the wild: an empirical study. In *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, Maurizio Morisio, Tore Dybå, and Marco Torchiano (Eds.). ACM, 23:1–23:10. https://doi.org/10.1145/2652524.2652533

[10] Matthew Finifter, Devdatta Akhawe, and David A. Wagner. 2013. An Empirical Study of Vulnerability Rewards Programs. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*. 273–288.

[11] Stefan Frei. 2009. *Security econometrics: The dynamics of (in) security.* Ph.D. Dissertation. ETH Zurich.

[12] Munawar Hafiz and Ming Fang. 2016. Game of detections: how are security vulnerabilities discovered in the wild? *Empirical Software Engineering* 21, 5 (2016), 1920–1959.

[13] Michael Hardy. 2010. Pareto's law. *The Mathematical Intelligencer* 32, 3 (2010), 38–43.

[14] Hideaki Hata, Mingyu Guo, and Muhammad Ali Babar. 2017. Understanding the Heterogeneity of Contributors in Bug Bounty Programs. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*. 223–228. https://doi.org/10.1109/ESEM.2017.34

[15] HyunChul Joh, Jinyoo Kim, and Yashwant K. Malaiya. 2008. Vulnerability Discovery Modeling Using Weibull Distribution. In *19th International Symposium on Software Reliability Engineering (ISSRE 2008), 11-14 November 2008, Seattle/Redmond, WA, USA*. IEEE Computer Society, 299–300. https://doi.org/10.1109/ISSRE.2008.32

[16] Jinyoo Kim, Yashwant K Malaiya, and Indrakshi Ray. 2007. Vulnerability discovery in multi-version software systems. In *High Assurance Systems Engineering Symposium, 2007. HASE'07. 10th IEEE*. IEEE, 141–148.

[17] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2201–2215. https://doi.org/10.1145/3133956.3134072

[18] Panagiotis Louridas, Diomidis Spinellis, and Vasileios Vlachos. 2008. Power laws in software. *ACM Trans. Softw. Eng. Methodol.* 18, 1 (2008), 2:1–2:26. https://doi.org/10.1145/1391984.1391986

[19] Thomas Maillart, Mingyi Zhao, Jens Grossklags, and John Chuang. 2017. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity* 3, 2 (2017), 81–90.

[20] Andrew Meneely and Laurie A. Williams. 2009. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*. 453–462. https://doi.org/10.1145/1653662.1653717

[21] Andrew Meneely and Laurie A. Williams. 2010. Strengthening the empirical analysis of the relationship between Linus' Law and software security. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, 16-17 September 2010, Bolzano/Bozen, Italy*. https://doi.org/10.1145/1852786.1852798

[22] Nuthan Munaiah and Andrew Meneely. 2016. Vulnerability Severity Scoring and Bounties: Why the Disconnect?. In *Proceedings of the 2nd International Workshop on Software Analytics* (Seattle, WA, USA) *(SWAN 2016)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/2989238.2989239

[23] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 692–708. https://doi.org/10.1109/SP.2015.48

[24] NIST NVD. 2020. *CWE Over Time.* Retrieved 2020-20-05 from https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cwe-over-time

[25] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (1999), 23–49.

[26] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. 2012. A large scale exploratory analysis of software vulnerability life cycles. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. 771–781. https://doi.org/10.1109/ICSE.2012.6227141

[27] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. 2014. Bug characteristics in open source software. *Empirical Software Engineering* 19, 6 (2014), 1665–1705.

[28] Daniel Votipka, Rock Stevens, Elissa M. Redmiles, Jeremy Hu, and Michelle L. Mazurek. 2018. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 374–391. https://doi.org/10.1109/SP.2018.00003

[29] Sung-Whan Woo, Omar H. Alhazmi, and Yashwant K. Malaiya. 2006. Assessing Vulnerabilities in Apache and IIS HTTP Servers. In *Second International Symposium on Dependable Autonomic and Secure Computing (DASC 2006), 29 September - 1 October 2006, Indianapolis, Indiana, USA*. IEEE Computer Society, 103–110. https://doi.org/10.1109/DASC.2006.21

[30] Mingyi Zhao, Jens Grossklags, and Peng Liu. 2015. An Empirical Study of Web Vulnerability Discovery Ecosystems. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015, Denver, CO, USA, October 12-16, 2015*. 1105–1117. https://doi.org/10.1145/2810103.2813704

A SUMMARY OF DATA SOURCES

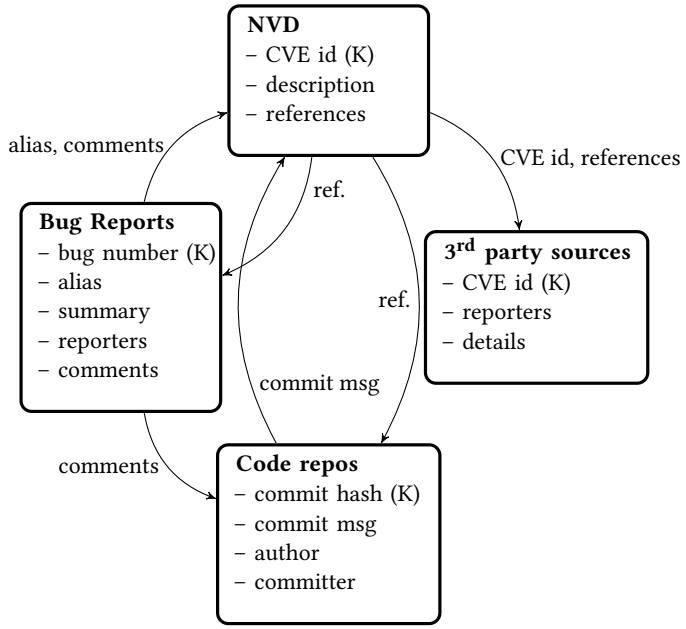

Figure 6: Summary of collected data points and their connections. K stands for the primary (unique) key of the collection.

B ADDITIONAL FIGURES

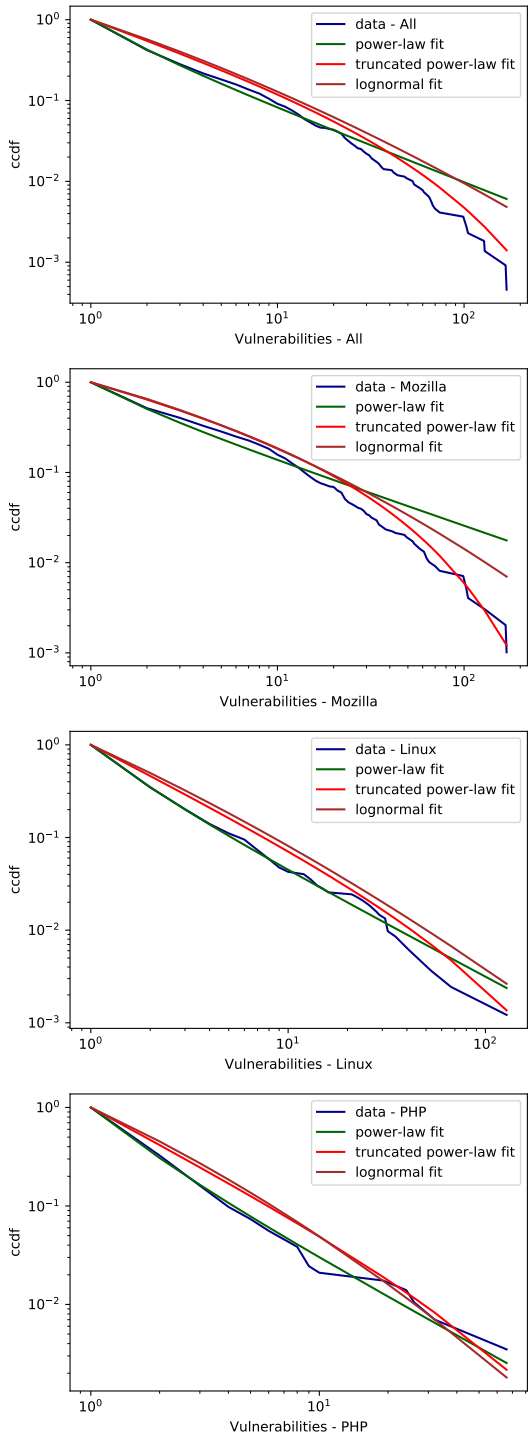

Figure 7: Heavy-tailed distribution fits (complementary cumulative distribution function).