Open Source Software Computed Risk Framework

1st Jon W Chapman
Department of Computer Science
University of California, Davis
Davis, Ca
jwchapman@ucdavis.edu

2nd Hari Venugopolan
Department of Computer Science
University of California, Davis
Davis, Ca
hvenugopolan@ucdavis.edu

***Abstract*—The increased dissemination of open source software to a broader audience has led to a proportional increase in the dissemination of vulnerabilities. These vulnerabilities are introduced by developers, some intentionally or negligently. In this paper, we work to quantify the relative risk that a given developer represents to a software project. We propose using empirical software engineering based analysis on the vast data made available by GitHub to create a Developer Risk Score (DRS) for prolific contributors on GitHub. The DRS can then be aggregated across a project as a derived vulnerability assessment, we call this the Computational Vulnerability Assessment Score (CVAS). The CVAS represents the correlation between the Developer Risk score across projects and vulnerabilities attributed to those projects. We believe this to be a contribution in trying to quantify risk introduced by specific developers across open source projects. Both of the risk scores, those for contributors and projects, are derived from an amalgamation of data, both from GitHub and outside GitHub. We seek to provide this risk metric as a force multiplier for the project maintainers that are responsible for reviewing code contributions. We hope this will lead to a reduction in the number of introduced vulnerabilities for projects in the Open Source ecosystem.**

***Index Terms*—Big Data, Computer Security, Prediction Methods, Data Analysis**

I. Introduction

Advanced software that controls much of everyday life has increasingly been created in an open source, collaborative manner. Many in the open source software world believe that given enough eyeballs, all bugs (and vulnerabilities) are shallow. This idea begins to break down at the scale of modern software projects. We see this in the fact that despite there being unlimited access to the code base of major software projects such as Apache, the Linux Kernel, cURL, etc, vulnerabilities still proliferate. The rapid growth in the amount of code requires a different paradigm to proactively prevent vulnerabilities before bad actors are able to exploit them.

In the past decade, GitHub has emerged as the preeminent platform for social software engineering [1], both open and closed source. GitHub provides an online collaborative platform for software development projects, acting primarily as the central repository for the projects' code. Not only is Github used in the internal development of projects by the developers of the various projects, but also acts as a public facing point of dissemination for software projects to end users.

This also means that GitHub has unwittingly become a platform for the spread of security vulnerabilities. A lack of security awareness among developers [2], [3] has led to the inclusion of vulnerabilities. Even security conscious developers can be responsible for creating vulnerable projects owing to their use of third-party libraries which could have vulnerabilities. [**?**] The lack of attention given to security, coupled with the difficulty in managing large scale projects that may have hundreds (or thousands!) of contributors increases the opportunity for the inclusion of vulnerabilities. There are also the cases of large scale projects that often underpin the working of the internet may only have a handful of developers working on them, but consist of many thousands of lines of code spread across hundreds of files, such as OpenSSL. These various challenges could could even set the conditions for malicious users to purposely include vulnerable code in a project that has a large audience in order to exploit later, such as we saw with NPMs most recently. [7]

The creation of a Developer Risk Score (DRS), combined with the Computational Vulnerability Assessment Score (CVAS) for projects are in support of two closely related research questions:

- **RQ1:** Can an accurate predictive model be created to give insights into future risk of a given contributor?
- **RQ2:** Can a derived, composite risk index be used to accurately predict which open source software projects will be assigned a CVE in the near future?

II. Prior Work

In this section we will explore the various efforts that other researchers have made in this field, as well as the foundational concepts that inform this work. This is a non-exhaustive overview that should serve to motivate our work as well as provide some context to our efforts. As we have seen no evidence of other researchers attempting to quantify the risk of specific developers using a multi-source data pooling approach we feel that this work is new and novel.

There are some tools out there that have similar aims to our project. They seek to prevent vulnerabilities before being distributed to users. These primarily center around static analysis techniques. Some advocate using code scanning tools [8] and others recommend increasing the skill and experience level of those doing the code review [10]

Security in the software development process is often an afterthought [11] [12]. In too many cases, security measures are taken after an exploit has been carried out, resulting in

large amounts of damage [13]. Even for the tools that attempt to mitigate the vulnerability before it effects users require code integration. There is little in the way of predictive tools that try to get a head of the vulnerability before it makes it into the code base.

There are differing opinions of why this situation persists, from lack of awareness [14], to poor education [15], and everything in between. We feel that by providing some additional information to developers, and making it easy to create an assessment of code that is potentially to be incorporated into a project will allow developers to make better (and more secure!) choices.

Our approach is designed to give simple metrics by which decisions to integrate (or not integrate) code, or at least conduct a more thorough review of the code, can be made. This results in a lowering of the "cognitive load" of the project maintainers in the pursuit of increasing security. [16]

III. Method

There exists a bipartite many-to-many relationship between contributors and projects, since a particular contributor can contribute to multiple projects, and since multiple contributors can contribute to the same project. We use this mapping, as shown in Figure 2 as the starting point of our analysis.

Each contributor is assigned a score for each project based on the number of vulnerabilities in the project weighted by their relative contribution to the project. This score is then aggregated for all projects the contributor has been involved with to calculate an effective score for the given contributor.

The sum of the scores of all contributors involved in a project would serve as the base score for the project itself. In future work we discuss this as a possible point of extension. In our study, we define the relative contribution of a developer to a project at a given point in time as the ratio of the number of commits the contributor has been involved in at that point in time(either as the author or as the committer) to the total number of commits that have been made to the project at that point.

We chose to go with this definition for the study, since the number of commits do serve as a direct proxy for the extent of a contributor's involvement with the project and also since the commit data is readily available on GHTorrent. Including the time component in this definition is important and helps make the score more accurate since it ensures that vulnerabilities that existed in a project before a given developer started contributing to it are not attributed to the contributor. An example of this is shown in Figure 3. In the figure, Developer A is the sole contributor to the project till point t_2 when Developer B comes in, and they both contribute equally to the project from $t2$ onwards. As per our metric, at any point in time up to t_3, the riskiness score of Developer A to the project will be 10, and that of Developer B will be 0. After point t_3, Developer A has contributed to 75% of the project on the whole, and Developer B has contributed 25%. As a result, after $t3$, Developer A's riskiness score would be 17.5 and that of B would be 2.5

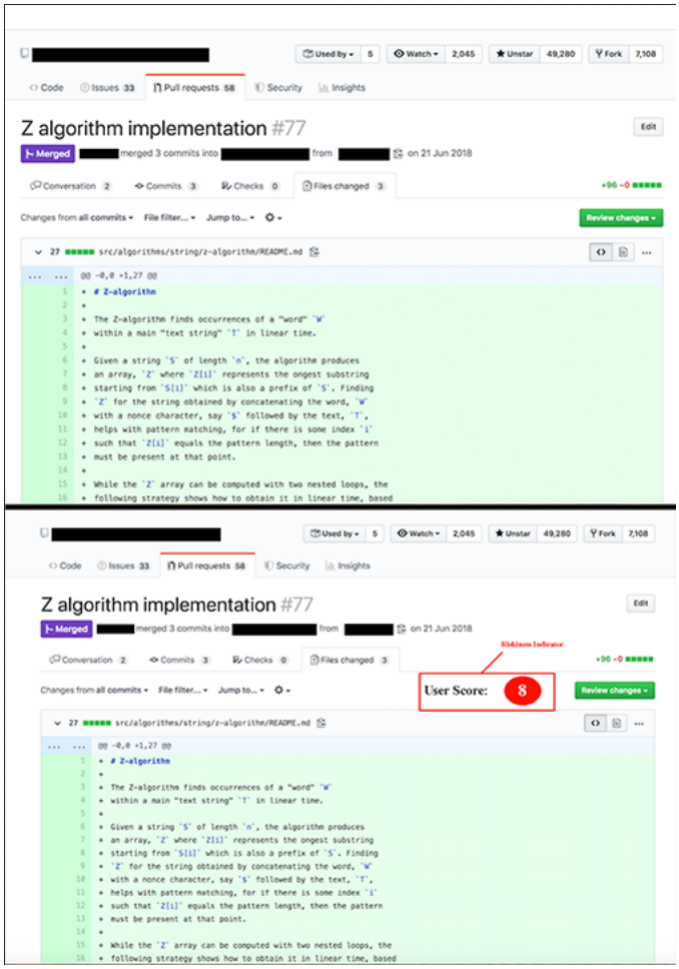

Fig. 1. The image on top shows the current pull request screen in GitHub. As can be seen, the pull request screen informs the user of only the contributors code. A link to the contributor's GitHub profile is also part of the pull request which the reviewer can visit to get additional information about the contributor if required. However, doing so is tedious, and does not quantify the risk associated with the contributor in any way. The image below shows our proposed new pull request review screen with an associated user riskiness score. This score serves to indicate the riskiness associated with the contributor, which the reviewer can now use to decide the extent to which review is required for the pull request in the context of better security.

With the understanding so far, and an acknowledgement that an iterative design process to refine this scoring method is still required, we propose a risk scoring metric that is tentatively defined as follows:

Let,

$$c_i(u_i, t_i, p_i) = \text{Relative contribution of Contributor } u_i, \text{ to Project } p_i \text{ at time } t_i$$

$$r_i(u_i, t_i, p_i) = \text{Partial Risk Score of Contributor } u_i, \text{ derived from Project } p_i \text{ at time } t_i$$

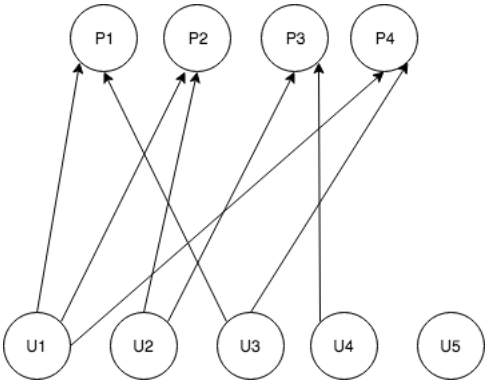

Fig. 2. Graph showing mapping between users and projects. The bottom nodes denote users, and those on top denote projects. Each user contributes to multiple projects and each project contains contributions from multiple users.

Based on our definitions:

$$c_i(u_i, t_i, p_i) = \sum_{t=0}^{t_i} \frac{num\_commits(u_i, p_i, t)}{\sum_{u \in p_i \text{ at time } t} num\_commits(u, p_i, t)} \quad (1)$$

$$r_i(u_i, t_i, p_i) = c_i(u_i, t_i, P_i) \times (cve\_score(p_i, t_i)) \quad (2)$$

Here, $num\_commits(u_i, pi, ti)$ denotes the number of commits that have been made by contributor u_i to project p_i at time t_i, and $cve\_score(p_i, t_i)$ is the CVE score assigned to the project p_i at time t_i.

We can use this to construct an overall risk score of a given contributor at a specific time, in general terms:

$$R_i(u_i, t_i) = \text{Risk Score of Contributor } u_i \text{ at Time } t_i$$

We then use the prior definition to construct the formal definition of the individual developer risk score at a given point in time as the ratio of the sum of partial risks for the user across all their projects at that point in time to the total number of commits made by them at that point. More formally this can be expressed as:

$$R_i(u_i, t_i) = \frac{\sum_{\text{all } p_i \text{ that } u_i \text{ has contributed to}} r_i(u_i, t_i, p_i)}{\sum_{\text{all } p_i \text{ that } u_i \text{ has contributed to}} [\sum_{t=0}^{t_i} num\_commits(u_i, p_i, t)]} \quad (3)$$

This risk score essentially conveys the risk associated with each commit made by the user, with a higher score indicating possibility of greater risk.

IV. Results

In this section we describe our experiments and preliminary results. We first describe and report the results of a simple experiment we ran, where we analyzed the correlation between the average number of commits made per user per project against the number of CVEs in the project. We worked on this with the hypothesis that the more number the number of commits made per user on average, the more responsibility each user takes up individually, resulting in less collaboration, thereby leading to more vulnerabilities. The resulting plot is shown in Fig 4. While we had to do significant cleanup of the plot to remove outliers, the plot does seem to indicate the presence of a correlation. We use this plot as motivation to inform ourselves of the possibility of drawing meaningful inferences from data. Our further, more detailed experiment to quantifying risk is described below. While the results of that experiment do seem to work towards our goal towards quantifying risk, we used them to further refine our metric to present that mentioned in Section III.

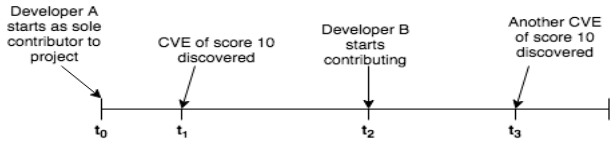

Fig. 3. This representative timeline demonstrates an example life cycle of a software development project on GitHub. This shows when CVEs were assigned to a project, and the points in time when different developers contributed to a particular project. As can be seen, assigning a score to Developer B for the CVE discovered before his/her contribution to the project would be inappropriate. Our model fully takes into account these sort of chronological considerations to ensure appropriate attribution. In this toy example, our model would assign Developer A with a risk score of 17.5 and Developer B with a risk score of 2.5.

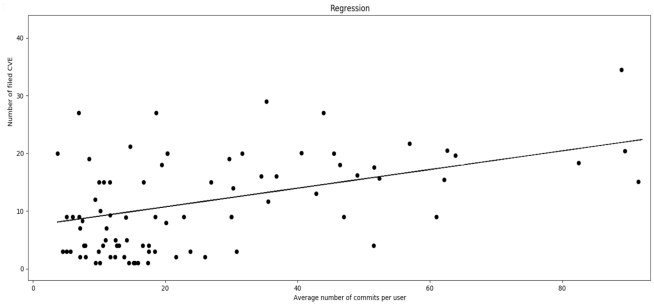

Fig. 4. Simple regression plot of the average number of commits per user per project against the number of CVEs present in the project. The plot works in line with our intuition that the more responsibility individually taken up by a developer, the larger the potential to introduce vulnerabilities. While the plot was heavily cleaned up to remove outliers, it serves to motivate that meaningful inferences can be drawn from the GitHub data towards analyzing risk.

A. Experiment

We first curated a list of approximately the top 1000 users, these were the users with the most contributions on GitHub, which we obtained from [27]. We then obtained a list of all projects these users had contributed to from GHTorrent and queried them against [28] to obtain their associated CVEs. Of the 57322 projects we obtained from the 1000 users, 618 projects had CVE scores associated with them. We then computed riskiness scores for each of the 1000 contributors with a simpler version of our riskiness metrics that did not

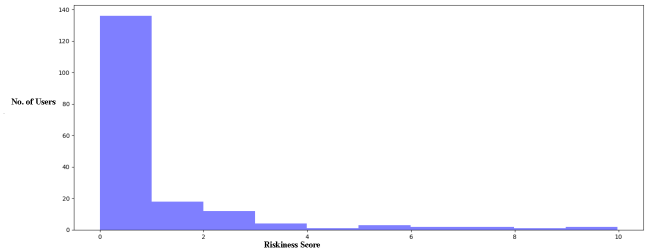

Fig. 5. Histogram showing the distribution of user scores. As can be seen from the figure, most user scores seem to cluster between 0 and 1. The plot only shows the counts for those users who had a non-zero score, and also does not show scores for a tiny portion users with abnormally large scores (over 300).

incorporate the time aspect, and then computed the relative contribution as the ratio of the total number of commits made by the user to the total number of commits made to the project. The resulting distribution of user scores is shown in Figure **??**.

B. Results

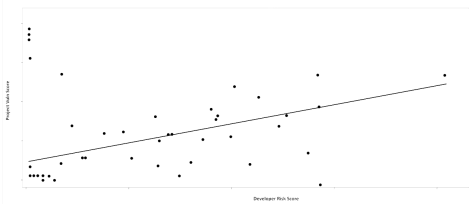

Fig. 6. Regression plot showing the user risk scores vs the vuln scores for a project over time.

We saw a distinct correlation between higher risk scores from development teams and projects having a vulnerability at a later point in our data set. This was an approximate .63 correlation between increasing score and vulnerability. This correlation increases as the risk score drives higher as seen in figure 7. Our thought is that this correlation would be even higher with cleaner data. It was difficult to parse the disparate data sets and ensure the project was correlated to the NVD.

We anticipate greater correlation on future iterations of this work.

REFERENCES

[1] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian The Promises and Perils of Mining GitHub 2014

[2] K. Watkins, S. M. Kywe Unsecured Firebase Databases: Exposing Sensitive Data via Thousands of Mobile Apps 2018

[3] C. Zuo, Z. Lin, Y. Zhang Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps 2019

[4] Gousios, Pinzger, van Deursen An Exploratory Study of the Pull-based Software Development Model 2014

[5] Common Vulnerabilities and Exposures https://cve.mitre.org/

[6] National Institutes of Standards and Technology National Vulnerability Database https://nvd/nist.gov

[7] Github Commitment to NPM ecosystem security https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/

[8] R. Kannavara, "Securing Opensource Code via Static Analysis," 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012.

[9] C. Cowan, "Software security for open-source systems," in IEEE Security & Privacy, vol. 1, no. 1, pp. 38-45, Jan.-Feb. 2003

[10] Payne, C. "On the security of open source software." Information Systems Journal, 12: 61-78. 2002

[11] Devanbu, Premkumar T., and Stuart Stubblebine. Software engineering for security: a roadmap. In Proceedings of the Conference on the Future of Software Engineering, pp. 227-239. ACM, 2000.

[12] Mouratidis H, Giorgini P. Integrating Security and Software Engineering. Idea Group Inc, Hershey, PA, USA. 2007.

[13] Cate FH. Information security breaches: looking back & thinking ahead. Published by the Centre for Information Policy Leadership. 2008.

[14] McGraw G. Software security. IEEE Security & Privacy. 2004

[15] Banerjee C, Pandey SK.https://www.overleaf.com/project/63292d80dc12b22eac8239d6 Research on software security awareness: problems and prospects. ACM SIGSOFT Software Engineering Notes. 2010

[16] Pfleeger SL, Caputo DD. Leveraging behavioral science to mitigate cyber security risk. Computers & security. 2012

[17] Lyu MR. Handbook of software reliability engineering. CA: IEEE computer society press; 1996

[18] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. In2014 IEEE Symposium on Security and Privacy 2014

[19] Lint Software https://en.wikipedia.org/wiki/Lint_(software)
[20] GitHub v3 API https://api.github.com
[21] GitHub GraphQL https://developer.github.com/v4/
[22] GitHub.com https://github.com
[23] BeautifulSoup Python Web Scraper https://pypi.org/project/beautifulsoup4/
[24] Selenium with Python https://selenium-python.readthedocs.io/
[25] Google Puppeteer https://developers.google.com/web/tools/puppeteer/
[26] Gousios G. The GHTorent dataset and tool suite. InProceedings of the 10th working conference on mining software repositories 2013 May 18 (pp. 233-236). IEEE Press.

[27] Most active GitHub users https://gist.github.com/paulmillr/2657075

[28] CVE Details https://www.cvedetails.com/

[29] CIRCL CVE SEARCH https://cve.circl.lu/api