

REPORT

# Bug Bounties and FOSS: Opportunities, Risks, and a Path Forward

**Ryan Ellis**  
Associate Professor, Northeastern University

**Jaikrishna Bollampalli**  
Candidate For Master Of Science In Software  
Engineering Systems, Northeastern University

PUBLISHED BY

Sovereign Tech  
Resilience





# **Bug Bounties and FOSS:** Opportunities, Risks, and a Path Forward

**Ryan Ellis**

Associate Professor, Northeastern University

**Jaikrishna Bollampalli**

Candidate For Master Of Science In Software  
Engineering Systems, Northeastern University

## Introduction

### 6 **Log4j: Open-Source Security Under a Microscope**

9 ..... Bounties and Open-Source Security: Charting a Path Forward

9 ..... Structure of the Report

## Section I

### 11 **Methods: Mapping Open-Source Challenges, Documenting Bounties**

11 ..... GitHub Data: Surveying Maintenance and Security

13 ..... HackerOne Data: The Internet Bug Bounty Program

14 ..... Interview Data: Understanding Open-Source Maintainers & Bug Bounty Participants

## Section II

### 16 **Open-Source Security: Structural Challenges**

17 ..... Securing the Commons: Collaborative Debugging in Open-Source Projects

19 ..... Maintaining Security: Time, Neglect, and the Problem of Popularity

20 ..... Life Gets in the Way: When Hobbies Come Last

21 ..... A Community of One: The Problem of Neglect

22 ..... Popularity and Its Discontents

23 ..... A Well-Maintained Project Is a Secure Project

24 ..... "It Sucks...": The Unique Challenges of Open-Source Security

## Section III

### 27 **Bug Bounties and the Ongoing Remaking of Security Work**

27 ..... Bounty Everything: The Rise of Bug Bounty Programs

27 ..... A Thriving Market for Flaws: An Overview of the Bounty Ecosystem

29 ..... The Benefits of Bounties: Improved Security and Flexible Work

30 ..... Precarious Work and Precarious Technology: Creating and Propagating Risk

## Section IV

### 32 **Open-Source Security Bounties: A Path Forward**

- 32 ..... Shrinking the Window of Vulnerability: Improving Security through Open-Source Bounties
  - 33 ..... The Risks of Open-Source Bounty Programs: The Case for Caution
  - 36 ..... Best Practices and a Path Forward: A Future for Open-Source Bounties
  - 36 ..... Recommendation #1: Invest in Holistic Approaches to Maintenance
  - 37 ..... Recommendation #2: Bounty Last, not First
  - 37 ..... Recommendation #3: Leverage Bounty Programs to Improve Identification
  - 38 ..... Recommendation 4: Open-Source Bounty Programs Should Adopt Ethical Practices
  - 39 ..... Recommendation #5: Bounty Funding Should Be Community-Driven and Aid Structural Support
- 

## Appendix A

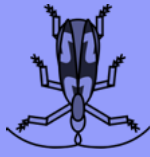
### 42 **Measuring Open-Source Maintenance**

- 42 ..... Metrics
- 42 ..... Statistics Glossary
- 43 ..... Statistical Insights

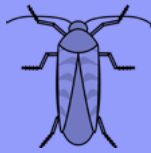
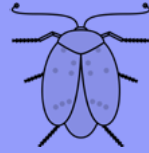
## Appendix B

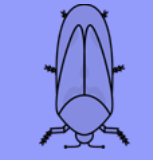
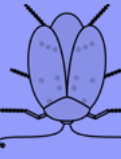
### 45 **Testing Maintenance and Security**

- 45 ..... Data Collection and Feature Selection
- 45 ..... Project Classification
- 46 ..... Modeling with Logistic Regression
- 46 ..... Feature Impact Analysis
- 46 ..... Key Insights
- 47 ..... Conclusion



**“When all the stars align just right then,**





I do what I can to help."



# Log4j: Open-Source Security Under a Microscope

Log4j had a “ticking bomb” sitting inside its code, and for years, nobody seemed to notice.<sup>1</sup> By the late-fall of 2021, the popular open-source logging tool was widely used: Apple, Google, Microsoft, Amazon, and countless others had integrated it into their applications.<sup>2</sup> Yet, unbeknownst to the developers that had included the software into their builds, or their billions of users that now used the resulting applications that included Log4j, a serious flaw had been introduced years earlier—in 2013—through an otherwise unremarkable update.<sup>3</sup> The vulnerability sat there for the better part of eight years, without anyone uncovering it.<sup>4</sup>

On November 21, 2021, a researcher emailed the small group of volunteers that maintain Log4j with the bad news: there was a significant bug in their code.<sup>5</sup> Like many open-source projects, despite its importance and ubiquity in commercial products, a fairly small collection of volunteers developed and maintained Log4j.<sup>6</sup> They offered it up as-is for the rest of the world to use as they saw fit with few restrictions.<sup>7</sup> The newly identified flaw was serious: it would allow a malicious attacker to take advantage of vulnerable systems and launch arbitrary code.<sup>8</sup> Gary Gregory, a member of the team that worked for years to maintain Log4j recalled his reaction to seeing the initial report: “[T]his one was, ‘oh crap.’ In this case some of us were surprised, not that there was a security issue, but just how bad it was.”<sup>9</sup> Jen Easterly, head of the U.S. Cybersecurity and Infrastructure Security Agency (CISA), described it as “one of the most serious flaws” she had encountered.<sup>10</sup>

As the frantic work to fix the flaw kicked off, more bad news followed: the vulnerability was being exploited in the wild.<sup>11</sup> The previously unknown bug was no longer a secret: billions of machines were potentially at risk.<sup>12</sup> The Log4j team rushed to notify the public and release an update that would fix the bug. The fallout was potentially catastrophic. Joe Sullivan, chief security officer for Cloudflare, observed that he would “be hard-pressed to think of company that is not at risk.”<sup>13</sup>

Now came the hard part. Developing a patch on a rushed timeline was indeed difficult, but deploying

<sup>1</sup> The description of the Log4j vulnerability as a “ticking bomb” is drawn from, Daniel Stenberg, Qtd. in William Turton, Jack Gillum, and Jordan Roberston, “Inside the Race to Fix a Potentially Disastrous Software Flaw.,” *Bloomberg*. Dec. 13, 2021.

<sup>2</sup> Google estimated that as of December 2021, tens of thousands of software packages and projects included Log4j. See James Wetter and Nicky Ringland, “Understanding the Impact of the Apache Log4j Vulnerability,” *Google Security Blog*. Dec. 17, 2021. Available online: <https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html>. The U.S. Cyber Safety Review Board, observed that Log4j was integrated into “millions of systems.” U.S. Cyber Safety Review Board, “Review of the December 2021 Log4J Event.” July 11, 2022, pg. *ii*.

<sup>3</sup> A detailed timeline of the genesis of the vulnerability, eventual disclosure, and response, is included in: Cyber Safety Review Board, “Review of the December 2021 Log4J Event.” Christian Grobmeier, vice president at Apache Software Foundation, which oversaw the development and maintenance of Log4j, noted during the ongoing crisis the ubiquity of Log4j, remarking that “basically half of the world, maybe even more” uses it. Qtd in Turton, Gillum, and Roberston, “Inside the Race to Fix a Potentially Disastrous Software Flaw..”

<sup>4</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event.”

<sup>5</sup> *Ibid*, 1-2.

<sup>6</sup> Log4j was maintained by a group of volunteers working under the Apache Software Foundation. See Turton, Gillum, and Roberston, “Inside the Race to Fix a Potentially Disastrous Software Flaw.”; and Patrick Howell O’Neill, “The Internet Runs on Free Open-Source Software. Who Pays to Fix It?,” *MIT Technology Review*, Dec. 17, 2021.



the fix—getting the myriad applications that had been built on top of Log4j to adopt it presented its own sharp challenges. There was no clear way to identify which applications actually used Log4j, with no master list or easy way to track them.<sup>14</sup> Most users had no idea if it was used in their applications.<sup>15</sup> A coordinated response was quickly spliced together: governments and tech companies released a flurry of advisories and warnings, and thou-

<sup>7</sup> Ibid.

<sup>8</sup> Later, the National Institute of Standards and Technology (NIST) would designate the flaw as a “critical vulnerability” and assign it the highest score possible according to the Common Vulnerability Scoring System (CVSS). Ibid, 1-2.

<sup>9</sup> Qtd. in Turton, Gillum, and Roberston, “Inside the Race to Fix a Potentially Disastrous Software Flaw.”

<sup>10</sup> Qtd. in O’Neill, “The Internet Runs on Free Open-Source Software.”

<sup>11</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event,” 2-4.

<sup>12</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event,” 2; O’Neill, “The Internet Runs on Free Open-Source Software.”

<sup>13</sup> Qtd. in Frank Bajakd, “The Internet’s on Fire’ as Tech Races to Fix Software Flaw,” *Associated Press*, Dec. 10, 2021.

<sup>14</sup> Joseph Marks, “One Month In, There Aren’t Any Huge, Known Log4j Hacks,” “The Cybersecurity 202” [Newsletter], *Washington Post*, Jan. 11, 2022.

<sup>15</sup> Ibid.

<sup>16</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event,” 3-9, 35-37.

<sup>17</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event,”; Marks, “One Month In, There Aren’t Any Huge, Known Log4j Hacks.”

<sup>18</sup> Cyber Safety Review Board, “Review of the December 2021 Log4J Event,” v.

sands of security professionals worked to mitigate the potential fallout, sharing tips, workarounds, and information through formal and informal channels.<sup>16</sup> Exploits and attacks followed the public release of the vulnerability, but the diligent and coordinated action of volunteers, tech companies, and governments appeared to mitigate worst case scenarios.<sup>17</sup>

The story of Log4j is something of a parable for open-source projects and open-source security. In many ways, Log4j points out the promise that open-source software provides: a small, volunteer-run project created a useful tool that found its way into billions of devices, providing untold value with minimal development costs. In terms of its reach, it is a stunning and unquestioned success. But the events of 2021 recast this success in a harsh light. Many commentators—in industry, government, the open-source community, and elsewhere—seized on the bug as an example of the shortcomings and failings of open-source security: a critical flaw in one of the building blocks of our shared digital infrastructure sat undetected for years and created a systemic and potentially catastrophic risk. The U.S. Cyber Safety Review Board’s detailed analysis of the incident placed the blame, in part, in the incentive structure and organization of open-source projects. As they observed:

[The Log4j] event also called attention to security risks unique to the thinly resourced, volunteer-based open source community. This community is not adequately resourced to ensure that code is developed pursuant to industry-recognized secure coding practices and audited by experts.<sup>18</sup>

The board drew attention to the seeming mismatch between the development of code on a voluntary basis and sound security practices. Security work, they reasoned, can often fall by the wayside without proper incentives and support.

This analysis was echoed by others. The U.S. Federal Trade Commission (FTC) argued that this vulnerability was a structural issue endemic to the open-source ecosystem:

[Log4j] is one of thousands of unheralded but critically important open-source services that are used across a near-innumerable variety of internet companies. These projects are often created and maintained by volunteers, who don't always have adequate resources and personnel for incident response and proactive maintenance even as their projects are critical to the internet economy.<sup>19</sup>

Patrick Howell O'Neill, writing in the *MIT Technology Review*, summarized the mismatch in incentives tartly, noting that while Log4j was founded as a volunteer project and run essentially for free, "million- and billion-dollar companies rely on it and profit on it every single day."<sup>20</sup> When things went awry, it fell to a group of volunteers working in their spare time to develop a workable patch. Chris Wysopal, CTO at Veracode, was blunt: the lack of funding of open-source projects was nothing short of "a systemic risk to the United States, to critical infrastructure, to banking, to finance."<sup>21</sup>

The Log4j bug, for many, underlines the fatal shortcoming at the center of open-source projects: voluntary projects with a lack of supporting resources fail to properly account for or prioritize security. One of the key benefits of open-source technologies, easy reuse packages, is undermined by potentially lax security. As packages are repurposed, insecure code spreads far and wide. Here, misaligned incentives allow for the propagation of a shaky component to be picked up and adopted, potentially causing catastrophic harm along the way.

In the months that followed, there were renewed calls for investment and prioritizing open-source security.<sup>22</sup> These were, certainly, not new concerns but they were crystalized and given a pointed edge.<sup>23</sup> But, the path forward is uncertain: how best to support open-source security remains both a pressing concern and an open question.

19 U.S. Federal Trade Commission (FTC), "FTC Warns companied to Remediate Log4j Security Vulnerability," *FTC Technology Blog*. Jan. 2, 2022.

20 O'Neill, "The Internet Runs on Free Open-Source Software."

21 Qtd. in O'Neill, "The Internet Runs on Free Open-Source Software."

22 For example, see Executive Office of the President, "Readout of White House Meeting on Software Security," Jan. 13, 2022. Available Online: <https://www.whitehouse.gov/briefing-room/statements-releases/2022/01/13/readout-of-white-house-meeting-on-software-security>; Responding to and Learning from the Log4Shell Vulnerability, U.S. Senate Committee on Homeland Security and Government Affairs [Hearing], Feb. 8, 2022; Brian Behlendorf, "Log4Shell Retrospective," Open Source Security Foundation [Blog]. Dec. 15, 2022. Available Online: <https://openssf.org/blog/2022/12/15/avoiding-the-next-log4shell-learning-from-the-log4j-event-one-year-later>; U.S. Office of the National Cyber Director, "Fact Sheet: Office of the National Cyber Director Requests Public Comment on Open-Source Software Security and Memory Safe Programming Languages," Aug. 10, 2023. Available Online: <https://www.whitehouse.gov/oncd/briefing-room/2023/08/10/fact-sheet-office-of-the-national-cyber-director-requests-public-comment-on-open-source-software-security-and-memory-safe-programming-languages>.

23 Corin Faife, "White House Hosts Tech Summit to Discuss Open-Source Security after Log4j," Jan. 13, 2022. Available Online: <https://www.theverge.com/2022/1/13/22881813/white-house-tech-summit-apple-google-meta-amazon-open-source-security>. See also Trey Herr, Prepared Remarks, United States Senate Committee on Homeland Security and Government Affairs. Feb. 8, 2022. Available Online: <https://www.hsgac.senate.gov/wp-content/uploads/imo/media/doc/Testimony-Herr-2022-02-08.pdf>.

## Bounties and Open-Source Security: Charting a Path Forward

The following pages consider the benefits and drawbacks of one possible intervention: the adoption of bug bounties for open-source projects.<sup>24</sup> While bounties have been selectively deployed for certain open-source projects, a comprehensive analysis of their suitability has not yet been undertaken. This report seeks to fill that gap. It draws together research on bug bounty programs, open-source communities, and open-source security, and it considers how bounties might improve security while avoiding myriad potential risks for security researchers, open-source projects, and the public.

The report uncovers a number of important findings. It indicates that bounties can usefully enhance the security of open-source software in specific circumstances: for mature open-source projects bounties can provide an extra layer of security. There are clear benefits: bounties can improve the number and quality of bug reports, they can help attract talented researchers, they can help projects retain expert talent and reduce community churn, and they can provide accountability mechanisms and tools that are otherwise lacking. These are potentially important and significant benefits. But, there are limitations and drawbacks as well. Bounty programs can undermine security in several ways. Counterintuitively, investing in them can undermine security by drawing unhelpful attention to understaffed or undermaintained projects, undermining reciprocity, a key value that animates open-source communities, creating new financial burdens that are unsustainable, and by drawing effort and resources away from the root causes of insecurity. These conclusions point toward a path forward, a way of designing and implementing open-source bounty programs that capture benefits for security researchers, open-source projects, and the public. This path is knotted and full of pitfalls, but with due caution there are opportunities for meaningful improvement.

## Structure of the Report

The report is divided into four sections. The first, “**Methods: Mapping Open-Source Challenges, Documenting Bounties**” (→ p. 11) provides an overview of the report’s methodology. The report employs mixed methods and draws on prior research to shed light on the challenges of open-source projects, the outline of bounty programs, and the potential fruitful collision

between the two. This section provides an overview of how complementary novel qualitative and quantitative data were scoped, collected, and analyzed; it also notes how earlier datasets were revisited and leveraged. Section two, **Open-Source Security: Structural Challenges** (→ p. 16), moves to consider the security challenges open-source projects face. Here, the report notes that the ability to identify and quickly

<sup>24</sup> Key examples of attempts to wed open-source projects and bug bounties include, the collaborative Internet Bug Bounty Program, Google’s Open-Source Vulnerability Reward Program, and the Sovereign Tech Fund’s Bug Resilience Project.

remediate bugs within a project is closely tied to the more general capacity to maintain the project. This section spotlights the diversity within the open-source ecosystem—projects of different size, shape, and organization dot the landscape. This section also pauses to underline the variability of performance across projects—while much attention is rightly drawn to security challenges, there are noteworthy successes worth highlighting. The report’s third section, **Bug Bounties and the Ongoing Remaking of Security Work** (→ p. 27), provides a primer on bounties, offering the unfamiliar with a capsule overview of where they came from and how they work. Drawing heavily on an earlier report by Ryan Ellis and Yuan Stevens, *Bounty Everything: Hackers and the Making of a Global Bug Marketplace*, this section identifies the benefits and harms associated with the rise of bounty platforms.<sup>25</sup> Finally, **Open-Source Security Bounties: A Path Forward** (→ p. 32), concludes with a detailed consideration of the risks and benefits associated with deploying bounties for open-source projects. It charts a path forward, identifying how and when bounties might be usefully expanded for select open-source projects. The report concludes with an overview of best practices and actionable recommendations. **Two appendices** (→ p. 42) provide a more in-depth discussion of the quantitative data analysis related to assessing project maintenance and the relationship between maintenance and security.

25 Ryan Ellis and Yuan Stevens, *Bounty Everything: Hackers and the Making of a Global Bug Marketplace*, Data & Society Research Institute, 2022. Available Online: <https://datasociety.net/library/bounty-everything-hackers-and-the-making-of-the-global-bug-marketplace>.

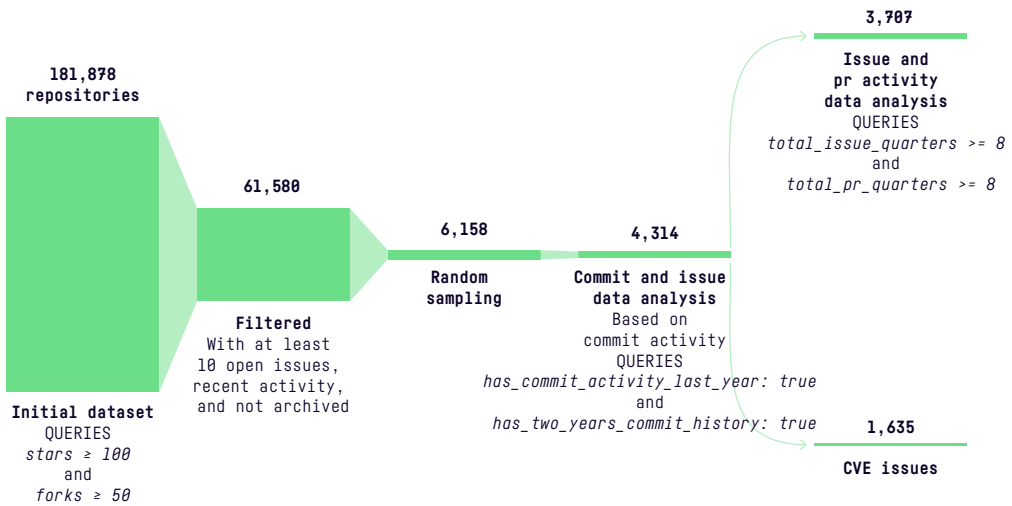
# Methods: Mapping Open-Source Challenges, Documenting Bounties

This report draws together data on the open-source ecosystem and bug bounties. It links newly collected qualitative and quantitative data with existing datasets in order to expand and build upon earlier analysis. The research is primarily based on two datasets: quantitative open-source project and bounty data; and qualitative interview data with key participants in open-source projects and bounty programs. The joining of qualitative and quantitative data provides a rich look that captures both the expansive and diverse nature of the open-source ecosystem, with fine-grained, personal observations. These datasets are complementary: they provide both a degree breadth and high-resolution focus that can speak usefully to the challenges of open-source security. Given the size of the open-source ecosystem, various filters were used to produce a sample that was representative and practical. A detailed discussion of the methodological approach follows; for those interested in skipping ahead to the analysis and results, see Section II: Open-Source Security: Structural Challenges, below.

## **GitHub Data: Surveying Maintenance and Security**

Data on open-source projects was collected from GitHub through targeted selection and subsequent rounds of filtering. This data was collected, stored, and analyzed in order to provide a window into the dynamics of open-source projects and allow for analysis across a number of domains related to general maintenance and security (for example, examining and interrogating the wide spread in performance related to the identification, review, and closing of issues related to a project). Initial selection sought to identify projects hosted on GitHub that were at least minimally significant and active in order to assess the challenges of maintenance and security (and the relationship, if any, between the two). Utilizing the query `stars ≥ 100` and `forks ≥ 50`, we identified 181,878 initial popular repositories [see Figure 1]. In order to narrow the dataset down to the most relevant repositories, we applied additional filters to focus on those that are currently active. The criteria included repositories with at least 10 open issues, a recent commit within the last year, and a status indicating that the repository was not archived. This filtering process reduced the dataset to approximately 61,000 popular and active repositories.

Given the large number of repositories identified, we performed a random sampling to manage the dataset size while preserving statistical significance. A 10% random sample of the filtered repositories



generated a subset of 6,158 repositories. This approach enabled a manageable yet representative sample, ensuring that the conclusions drawn would be applicable.

↑ **Figure 1** Process of targeted selection and subsequent rounds of filtering of GitHub repositories

Next, we collected detailed data on the identified repositories' issues and commit histories. This phase of the analysis focused on collecting data and developing metrics related to maintenance activities, including issue resolution times, commit frequencies, and contributor involvement. The rationale behind this step was to obtain a deeper understanding of each repository's development dynamics and to identify patterns that might indicate either sustained maintenance or potential decline. The comprehensive data collected during this phase provided the foundation for subsequent analyses of maintenance history and trends.

In looking at trend data, we ensured that partial or incomplete data (for example, projects with only one or two months of data) would not skew the analysis. To this end, we applied additional criteria to ensure that the selected repositories demonstrated development activity over an extended period and had a history of issue and pull request resolutions. First, we filtered for repositories with commit activity within the last year and a commit history spanning at least two years. Next, we filtered the dataset by focusing on repositories with a history of issue resolution, specifically those with at least eight quarters of data. This filtering step was crucial for analyzing long-term maintenance trends. Finally, to refine the dataset further, we applied a filter for repositories with a history of pull request resolution, also requiring at least eight quarters of data. These steps reduced the dataset to 3,707 repositories, ensuring that the ones selected were consistently active, well-maintained, and exhibited a suitable record of development and resolution practices.

These 3,707 repositories were analyzed and ranked across classification metrics in order to identify trends and distributions related to maintenance activity. Maintenance metrics included, calculating median issue resolution time, median pull request resolution time, and the relevant slopes associated with these metrics (to indicate improvement or decay in performance over time).

To enhance our understanding of the security practices, we conducted a focused analysis on vulnerabilities reported within sampled repositories. After filtering the dataset to 4,314 repositories based on recent commit activity, we shifted our attention to gathering detailed issue data and extracting Common Vulnerabilities and Exposures (CVE) information. This step involved processing JSON files that contained issue and pull request data for each repository. We cross-referenced these with CVE data downloaded from the official CVE GitHub repository to identify vulnerabilities associated with the repositories in our dataset. The process began by extracting CVE numbers from issue titles using a regular expression, followed by constructing file paths to locate the corresponding CVE JSON files. These files were then parsed to retrieve key information such as publication and update dates. For each issue linked to a CVE, we calculated various metrics, including the delay between the CVE publication and issue creation, as well as the time taken to resolve the CVE from both its publication and date of update. This detailed analysis allowed us to quantify the responsiveness of the repositories to security vulnerabilities, a useful proxy for the security posture of these open-source projects.

Following the extraction of CVE data, we identified approximately 1,600 repositories with associated CVE information. For these repositories, we conducted a secondary analysis to gather specific metrics related to their handling of security vulnerabilities. The secondary analysis involved calculating key statistics for each repository, including the median, mean, maximum, and minimum CVE resolution times from the date of publication, as well as the total count of CVEs per repository. Repositories without CVEs were filtered out to ensure the analysis focused on those actively dealing with vulnerabilities.

## HackerOne Data: The Internet Bug Bounty Program

HackerOne, the bug bounty platform, hosts the Internet Bug Bounty (IBB), a program designed for select open-source projects.<sup>26</sup> Data related to the Internet Bug Bounty program offered an opportunity to analyze, with some

<sup>26</sup> Currently, 21 projects are included in the IBB program's scope, including curl, Ruby, Rust, libssh, Django and others. See HackerOne, "Internet Bug Bounty." Available Online: <https://hackerone.com/ibb?type=team>.

limitations, how bounty programs have been adopted to open-source projects. We gathered all the available public reports from the HackerOne IBB. This dataset includes detailed reports of vulnerabilities and security issues submitted by researchers participating in the program. The collection of these reports is useful for analyzing the

impact and trends in security contributions across various open-source projects within the scope of the IBB. This data, in particular, is useful in illuminating how security researchers interact with different open-source repositories included within the program's purview.

## Interview Data: Understanding Open-Source Maintainers & Bug Bounty Participants

Quantitative data was blended with 62 interviews conducted with key participants in open-source projects and bug bounties. The sample included 22 interviews conducted specifically for this project that focused on open-source projects and open-source security, and 42 previously conducted interviews focusing on bug bounties. Interview data was critical: it illuminated and, in some cases, complicated the trends observed within the quantitative data. Semi-structured conversations with key figures in both open-source projects and bounty programs offered useful insights that would have not otherwise been visible.

In the summer of 2024, the project team conducted interviews with 22 maintainers and key participants in open-source projects and open-source security. The interviewees were identified in three ways. First, open-source projects included in the GitHub dataset, described above, were analyzed in order to identify the top five percent and bottom five percent of all projects according to median issue resolution time (that is, how long it took a project to resolve a submitted issue). This measure was selected as a rough but useful proxy for maintenance capacity or ability. Drawing interviewees from both ends of the spectrum sought to ensure that a range of perspectives, both in terms of success and frustrations or difficulties, would be captured in the interview data. Contact information for key participants and maintainers was extracted from the GitHub data sample and interview requests were sent to identified potential subjects.

Additionally, a more targeted outreach effort centering on individuals directly working on open-source security was undertaken. Here, personnel associated with the repositories included in the current Internet Bug Bounty program were identified through the HackerOne dataset. Manual identification of additional possible interviewees was undertaken by reviewing GitHub data, including contacts and personnel indicated in associated security policies and security teams. Snowball sampling led to the identification of additional potential subjects.

Interviews were open-ended and conducted via videoconferencing software.<sup>27</sup> Professional transcripts were prepared and shared with all interviewees. All participants were allowed to review and edit their transcript as they saw fit; they also were given the option to use their real name or a pseudonym for the purposes of this project. Interview memos were produced for each interview; and transcripts were analyzed.

<sup>27</sup> Two interviews were conducted according to a different protocol at the request of the interviewees. These interviews were conducted via email and were, as a result, somewhat more formal and structured.



This open-source security interview data was supplemented by earlier rounds of interviews conducted by Ryan Ellis and Yuan Stevens in preparation for their earlier report, *Bounty Everything*. This included 42 interviews with key participants in bounty programs conducted between 2019 and 2021. These interview transcripts were reviewed and reconsidered in light of the particular questions raised in this report. These interviews offered useful information concerning the mechanics, benefits, and risks of bounty programs in general. The bounty interviews were conducted in the same manner as the open-source ones: all interviews were transcribed, reviewed by participants, and then analyzed. Two interviewees, Katie Moussouris (Luta Security CEO) and Jack Cable (CISA), experts in cybersecurity with significant experience in both bug bounty programs and open-source security, were included in both rounds.

# Open-Source Security: Structural Challenges

It is hard to overstate the importance of open-source technology. Nearly every application, mobile device, and digital network that we use relies on open-source components.<sup>28</sup> Open-source technologies are foundational: they are the building blocks for much of the technology and services—commercial, non-profit, and government—that we encounter in our daily lives.

Open-source software stands in opposition to closed or proprietary software.<sup>29</sup> It is defined by a commitment to make source code available for review, modification, and sharing. Critically, a typical open-source license requires that all subsequent modifications be made available under the same terms. In other words, additional iterations must likewise be made available for review, modification, and sharing. Open source, as an innovation, is a clever inversion of typical property regimes and imaginings. While intellectual property is usually structured around exclusivity and control, open source is defined by an imperative to share—to distribute and make available work for others to make use of and build as they see fit.<sup>30</sup>

Voluntary work sits at the center of the open-source ecosystem. While some open-source participants are funded through their employer, grants, or other means, volunteering is the norm. As such, participants elect to work on projects and tasks that interest them.<sup>31</sup>

Volunteers are drawn to open-source projects for a host of reasons: curiosity, an opportunity to work with a likeminded community, a drive to learn new skills, a desire to empower others, and, perhaps most of all, a do-it-yourself spirit that inspires them to create something new in the world. Chris, a long-time contributor to MDN Web Docs (formerly, Mozilla Developer Network), recalled with still-evident wonder the moment when he realized that “all these little bits of software that we use...are made by a whole bunch of volunteers.”<sup>32</sup> The work of many, as he put it, has “advanced us more than we could know.”<sup>33</sup>

Reciprocity is key value. While projects are free to be reused and much—though not all—development is undertaken on a voluntary basis, reciprocal contributions are prized and sought after. Sarah, a member of Django's security team, recalled that she was initially drawn to

<sup>28</sup> The importance and ubiquity of open-source technology is broadly recognized, see Executive Office of the President, “Securing the Open-Source Software Ecosystem.” Jan. 2024, 4. A recent study concluded that open-source components are found in 96% of codebases. Manuel Hoffman, Frank Nagle, and Yanuo Zhou, “The Value of Open Source Software,” Harvard Business School Working Paper Series, 2024.

<sup>29</sup> No single definition of “open source” is available but, generally, the GNU General Public License and its requirements are a useful bedrock. Thomas Haigh and Paul E. Ceruzzi, *A New History of Modern Computing* (Cambridge, MA: MIT Press, 2021). Important debates between free software, open source, and other competing histories, visions, and projects exist but are beyond the scope of this report. For in-depth and useful discussions of the history, organization, and importance of free and open source software, see Steven Weber, *The Success of Open Source* (Cambridge, MA: Harvard UP, 2004); Christopher M. Kelty, *Two Bits: The Cultural Significance of Free Software* (Durham, NC: Duke UP, 2008); E. Gabriella Coleman, *Coding Freedom: The Ethics and Aesthetics of Hacking* (Princeton, NJ: Princeton UP, 2013); and Christopher Tozzi, *For Fun and Profit: A history of the Free and Open Source Revolution* (Cambridge, MA: MIT Press, 2017).

<sup>30</sup> See Coleman, *Coding Freedom*.

<sup>31</sup> Weber describes this as an ideal type, true often enough and strived for but with notable exceptions (as discussed below). Weber, *The Success of Open Source*, 62.

<sup>32</sup> Chris M., Interview, 2024.

<sup>33</sup> *Ibid.*

participate in open source projects in part as a way to “give back a little to some of the packages that I really admire and rely on.”<sup>34</sup> In conversations with participants and maintainers, they spoke plainly about their desire and hope that users would turn and contribute back to the projects that they use.

In practice, the open-source ecosystem is defined by a broad expanse of projects with wide variation in purpose, size, and organization. It includes large, complicated projects that span decades of work with established codes of conduct and contributor guidelines, a defined group of core maintainers, specialized security teams, thousands of contributors, and millions of users; and it includes small projects that are created and run by a handful of individuals with few (if any) meaningful contributors from others. Nadia Eghbal’s survey of the challenges of creating sustainable open source projects, *The Making and Maintenance of Open Source Software*, provides a rough typology of different open source projects. It identifies *federations*, defined by a high number of contributors (usually organized into subgroups) and a large pool of users; *clubs*, with a roughly equal match of contributors and users; *toys*, small projects with one maintainer and few users; and, finally, *stadiums*, defined by a small number of contributors and many passive users.<sup>35</sup> Dennis, a core contributor to Managram, an open-source microkernel-based operating system, summarized the breadth of the ecosystem well, reflecting on how large and popular projects, such as Debian, develop into effectively a “mini-government” with a dedicated administrative staff, steering committees, and all the rules and process that follows, while other projects hinge almost entirely on the work of two or three maintainers.<sup>36</sup> These different sorts of projects all have a role to play in the open-source ecosystem, though Dennis had to note that “[n]o want starts wanting to be involved in a mini-government.” If you want to be involved in a government...go into politics.”<sup>37</sup>

34 Sarah B., Interview, 2024.

35 Nadia Eghbal, *The Making of Maintenance of Open Source Software* (San Francisco: Stripe, 2020), 56-64.

36 Dennis B., Interview, 2024.

37 Ibid.

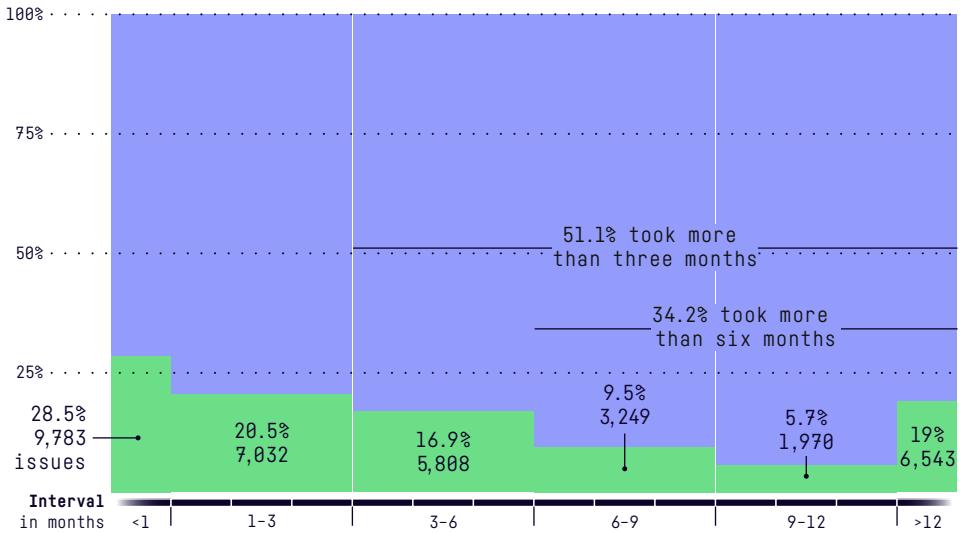
38 Bugs, of course, are not only or simply a security problem. Many flaws create annoying difficulties for users without undermining security. For the purposes of this report, the more general label, “bug,” will be used interchangeably with the more specific and narrower category or subset of bugs that are security flaws or vulnerability.

39 Frederick P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition (Reading, MA: Addison-Wesley, 1995), 20.

## Securing the Commons: Collaborative Debugging in Open-Source Projects

For both open source and proprietary projects, finding and fixing bugs is a constant challenge. Unpatched bugs can create novel pathways for attackers and undermine security.<sup>38</sup> Decades ago, Fred Brooks, in his classic reflection on software design and management, *The Mythical Man-Month*, estimated that roughly 50% of the time devoted to developing a commercial software program is occupied with finding and fixing bugs.<sup>39</sup> Yet, despite best efforts, they do, of course, slip through the cracks and find their way into software applications that have escaped the backroom of product testing and made it to market. As Brooks ruefully noted, “[i]n the merciless light of real use, every flaw will

Percentage



show.”<sup>40</sup> Users have a way of spotting all the flaws that the developers somehow missed.

Years later, Eric Raymond recast Brooks’ observation in the context of open-source software, and the lament became a celebration.<sup>41</sup> For Raymond and others, open source collapses the distance between user and developer. By exposing projects to varied users, new flaws can be found and fixed by relying on the effort, experience, and wisdom of the crowd. As Raymond famously put it, “[g]iven enough eyeballs, all bugs are shallow.”<sup>42</sup> The embrace of a wide-pool of users and developers, for Raymond, can lead to the more efficient spotting and fixing of flaws.<sup>43</sup> For Brooks, users finding and reporting flaws, was a testament to the inevitable incompleteness and imperfection that clung to software development; for Raymond, collaborative debugging was one of the strengths of open source development.

In practice, however, finding and fixing bugs in open-source projects remains a spot of stubborn friction. Bugs have a long shelf life, often remaining unidentified or unfixed for months or *years* after first being disclosed and published. A review of issue resolution time and CVE data for our GitHub sample is revealing. The majority of issues (51.1%) with an associated CVE number took more than three months to fix, with more than a third (34.2%) taking longer than six months to resolve [see Figure 2]. A fairly narrow set of issues, 28.5%, were fixed within the first month after initial publication of the CVE. Clearly, as with general issues, most security issues linger and are slow to be resolved.

↑ Figure 2 Resolution Time for Issues with Associated CVE

*Interval runs from initial CVE publication date to the date when associated issue is resolved by a project. Data drawn from 1,824 unique repositories and includes 34,385 resolved issues and 3,446 open issues. See Section I: Methods: Mapping Open-Source Challenges, Documenting Bounties for a detailed discussion of data collection and sampling.*

<sup>40</sup> Brooks, *The Mythical Man-Month*, 69.

<sup>41</sup> Eric S. Raymond, *The Cathedral and the Bazaar* (2000). Available Online: <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html>.

<sup>42</sup> Raymond famously dubbed these observations as “Linus’s Law” in reference to Linux development. Raymond, *The Cathedral and the Bazaar*.

<sup>43</sup> *Ibid.*

## Maintaining Security: Time, Neglect, and the Problem of Popularity

Open-source security challenges largely track the general challenges of maintaining open-source projects (as discussed in detail below). The same difficulties that make it hard to keep up with the day-to-day maintenance of an open-source project also spill over and undermine security. A close look at a cross-section of projects is clear: a well-maintained project is often a secure project. Likewise, projects that struggle with everyday maintenance typically have trouble identifying and fixing bugs in a timely fashion.

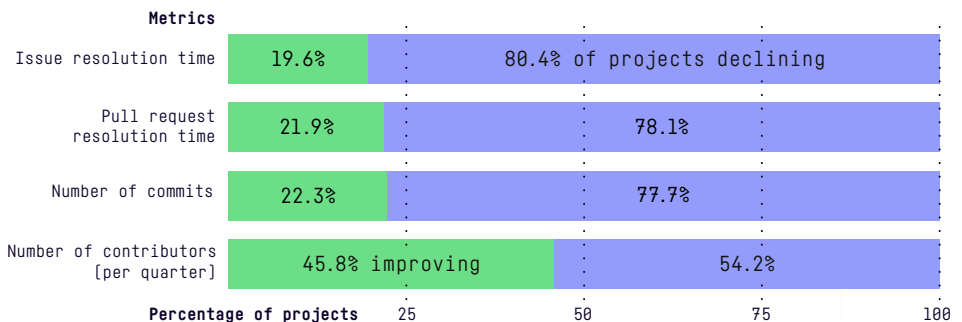
Maintaining open-source projects is a clear and ongoing challenge. A review of the assembled GitHub data underlines the significant variation that exists within open-source projects—some are well-maintained, while others are in a state of comparative disrepair (for a detailed analysis of the data, see **Appendix A: “Measuring Open-Source Maintenance”** → p. 42). While the median time to resolve issues for projects across the sampled data was a little over 20 days, some projects can take several years to resolve issues. Similarly, data regarding pull request resolution times follows a similar pattern: significant variation with wide extremes across projects. An analysis of a broad set of metrics related to maintenance, including the rate of commit activity, issue resolution time, pull request resolution time, and others show significant variance across projects.

The majority of open-source projects are in a state of decline, following a similar trajectory: a decline in maintenance and activity over time. A review of the collected project data reveals a consistent picture (see **Figure 3**). As they progress, most open-source projects begin to take longer and longer to respond to submitted issues.

Likewise, over time, most of them start to show longer gaps between when a pull request is submitted and when it is resolved. As for activity, most projects see a decline in the number of active contributors as the project ages: fewer and fewer new people enroll to contribute. And, unsurprisingly, the number of commits also dwindles.

► **Figure 3** Analysis of Open-Source Project Performance Over Time

*Data collected from 3,707 open-source repositories. Improving and declining projects are identified through analysis of the slopes associated with various metrics.*



The difficulties associated with maintaining open-source projects are no mystery, with maintainers and core contributors repeatedly pointing to the same family of issues that bedevil effective stewardship: a lack of time and resources, too few participants and worthwhile contributions, and an overwhelming flood of contributions and attention—that bedevil effective stewardship. These issues eat away at the ability of maintainers to keep their projects updated and responsive to their users; they also, unsurprisingly, undermine security.

### Life Gets in the Way: When Hobbies Come Last

In conversations with open-source maintainers, complaints regarding the lack of time available to devote to their projects is a near-constant refrain. For many, maintaining an open-source repository is—or at least started out as—a labor of love. But the pressures of life often do not leave enough space for this voluntary work. For many, the commitments of a full-time job, family, or other interests squeeze the project. Johann, a research scientist at the Pasteur Institute, ran through all the ways that life had started to crowd out participating in voluntary projects. Work and family were his priority: he had left little time for his hobby, and this impacted his ability to maintain voluntary projects. As he recounted with a certain amount of self-deprecation:

[S]ometimes I miss messages. I forget to answer or I tag them somewhere and I forget to add them to my to-do list. So I just forget about them. And there's messages without answers for a month. Or I don't integrate patches...I don't follow up with contributions because I [have]very [little] time.<sup>44</sup>

He spoke for many harried open-source participants when he noted that “very often this hobby, open-source projects, are the first thing to [go].”<sup>45</sup> Mark, the creator and maintainer of a popular gaming add-on, echoed Johann's point, noting that as a hobby, the project almost always, “gets put down at the end” of his list of responsibilities.<sup>46</sup> Dennis, the Managram maintainer introduced above, was succinct: “Unfortunately, real life is busy.”<sup>47</sup>

Some, like Uday, the creator and maintainer of the package RNSwipeButton, find themselves in a bind.<sup>48</sup> Four years ago, he started this project because it was something that he wanted—rather than waiting for someone else to write the code he needed, he did it himself. Over time, it became useful and widely used, adopted and repurposed by others. But now his interest is starting to flag. Uday wants to move onto something new, something

<sup>44</sup> Johann, Interview, 2024.

<sup>45</sup> Johann, Interview, 2024.

<sup>46</sup> Mark W., Interview, 2024.

<sup>47</sup> Dennis B., Interview, 2024.

<sup>48</sup> Uday K., Interview, 2024.

different. But he feels a responsibility to those that use and have built on his work. They are counting on him to continue to maintain and keep the project updated; if he doesn't, he feels that he will jeopardize their work. Dennis, introduced above, summarized this common trajectory: “[it] always starts out as either I need something, it isn't available, so I'll write it myself; or, 'hey, this looks fun and I want to help the project out because I use it.’”<sup>49</sup> But, over time, he reflected, this somewhat spur of the moment lurch into open-source morphs into a sense of responsibility. It is a sticky problem: maintainers like Uday are torn between keeping up support for the project and community that they helped foster and a desire to move on.

The cross-pressures of life, the lack of time to get back to that nagging hobby that needs tending, creates difficulties for open-source projects. When maintenance understandably comes last, projects become slow to respond to pull requests and the pile of issues waiting for attention. With respect to security, the costs are easy to spot: new bugs are slow to be identified and slow to be fixed.

### A Community of One: The Problem of Neglect

For many projects, these challenges are compounded by neglect. In these instances, the hoped-for flowering of a community of contributors has not come to pass. Rather, within what Eghbal describes as *Toy* and *Stadium* projects, a single maintainer or small group of maintainers shoulder the full burden of running the project with little productive support from a larger community of users.<sup>50</sup> Even for projects that have caught on and developed a large user base, there often is still a lack of *useful* contributions. Dennis, the Managram maintainer, was quick to note that many important and popular projects are actually contingent on the work of one or two maintainers: “It is almost scary that we as a community...are relying on a few people to maintain such core pieces of infrastructure.”<sup>51</sup> Users may well like and use the project, but, in the eyes of the maintainers, they are not always investing their time and effort into growing or supporting the project.

Maintainers commented on what they took to be a lack of reciprocity. Users often are eager to make use of an open-source project, perhaps even flag an issue or two, but they are reluctant or unwilling to step in and make ongoing durable contributions. Jan, a contributor to BigBlueButton, an open-source virtual classroom software program, remarked with some exasperation that:

one thing I really don't like that I have experienced in several [open source] projects is people creating feature requests with a kind of tone, like demanding stuff to be developed, but

<sup>49</sup> Dennis B., Interview, 2024.

<sup>50</sup> Eghbal, *The Making and Maintenance of Open Source Software*.

<sup>51</sup> Dennis B., Interview, 2024.

...  
[they] refuse to work on it by themselves...And I don't know how, where they get it from, but they think they have the right to ask for features and they have to be done just because they're asking for it.<sup>52</sup>

For Jan and others, it seems that it is always easier for users to write in with a request for some new added feature or point out a possible problem, than it is for them to roll up their sleeves and try to develop a feature or fix. Chris, the MDN maintainer, is always trying to spot potentially useful contributors and grow the larger community, looking to see who might be invested in the project as opposed to others that are just passing through and “submitting a single fire-and-forget PR.”<sup>53</sup>

The problem of neglect is a burden for open-source projects. A lack of active contributing users and a lack of reciprocity among users leaves projects in a difficult state: contributions are concentrated on a single or small circle of overburdened maintainers.

### Popularity and Its Discontents

Ironically, in some instances the only thing worse than too few contributors is too many.<sup>54</sup> Not all contributions are useful or welcome: sometimes they are more hassle than help. Poorly drawn pull requests, meandering or unclear issues, harassing comments, and outright spam pull maintainer attention, already a scarce resource, away from core project work with little added benefit.

Mark, the core maintainer of a small project that provided an add-on for role playing games, spoke for many when he noted that the poor quality of some issues and pull requests created a drag for the project.<sup>55</sup> These requests are time-consuming and sometimes tangential to the project. Uday underlined this point, complaining that often people submit what are effectively a feature request as an issue, an annoyance that draws him away from actual issues that need his attention.<sup>56</sup>

For Alex, a long-time contributor to Rust, the accessibility of open-source projects is at once a blessing and a curse. For a popular project like Rust, he noted, the stream of issues and pull requests can be a real “firehouse,” an endless thread of things that need attending to.<sup>57</sup> Alex reflected on working on a popular project, noting that users are constantly interjecting:

...  
[they will] be like, ‘it’s broken. Please fix this. I need this bug, I need this feature,  
...  
...

<sup>52</sup> Jan K., Interview, 2024.

<sup>53</sup> Chris M., Interview, 2024.

<sup>54</sup> Eghbal discusses at length the challenge of extractive contributions and the problem of attention. There reframing of the problem of open-source maintenance as one of too much community involvement, rather than too little, is invaluable. See Eghbal, *the Making and Maintenance of Open Source Software*.

<sup>55</sup> Mark W., Interview, 2024.

<sup>56</sup> Uday K., Interview, 2024.

<sup>57</sup> Alex C., Interview, 2024.



...

blah, blah, blah, blah, blah.’ And so dealing with the constant weight of that can become very burdensome, especially as it streams in over the span of a decade... Some folks are super rude. They’re like, ‘why have you not done this? This is ridiculous. How can anyone not do this?’ So seeing that over time can be very crushing...<sup>58</sup>

To Alex, these sort of demanding and rude comments are “one of the major downsides of open source.”<sup>59</sup> In an open community, where all contributions are welcome, some contributions are, at best, counterproductive, and at worst, entitled and abusive. He was clear: these sorts of interactions are the one of the primary reasons he “eventually burned out on Rust.”<sup>60</sup>

Chris, the maintainer introduced above that works on MDN Web Docs agreed with many others in noting that the high-volume of incoming submissions was time consuming and draining. Remarking with sympathy that, “[t]hen there’s loads of people that turn up...that are interested in helping but they haven’t got a clue how to help.”<sup>61</sup> But, he is reluctant to foreclose avenues for broad participation or contributions. He wants the project to remain open and accessible: as he notes, you never know, when a new first-time contributor might actually grow into a productive and long-term member of the project. Alex, despite his misgivings, agrees on this point: keeping open source open, having a low bar for entry, despite the drawbacks, is important; it is how new, fresh, blood joins the project. This churn of new entrants finding their way to contribute, is important and matches what many participants see as a key benefit of open-source projects: an ecosystem that is open to all and that can empower anyone.

Finding ways to manage attention—to limit those tasks that sap energy and pull maintainers away from positive contributions—is critical. Many projects have taken steps to try to improve the quality, and not just the quantity, of contributions. Guidelines, some degree of formal or informal vetting, and other practices can all help, but these steps always involve some degree of trade-off: barriers are, in the end, risk alienating new or returning contributors. Yet, for many projects it is worth the risk.

### A Well-Maintained Project Is a Secure Project

Maintenance challenges spillover and undermine security. Quantitative analysis of the GitHub data is instructive: projects that are adept at essential maintenance activities, including most notably efficient management of pull requests and a high degree of ongoing commit activity (engagement with the project) are also adept at identifying and fixing security issues in the repository (for a detailed overview of the quantitative analysis, see **Appendix A: “Testing Maintenance and Security”** → p. 45).

<sup>58</sup> Alex C., Interview, 2024.

<sup>59</sup> Alex C., Interview, 2024.

<sup>60</sup> Alex C., Interview, 2024.

<sup>61</sup> Chris M., Interview, 2024.

These practices are firmly knotted: maintenance and security walk hand-in-hand. The work of maintaining a project—responding to issues, approving pull requests in a timely manner, seeking out and cultivating active and productive contributions—is also the work of creating a secure project. Without proper maintenance, security inevitably suffers. Indeed, without adequate time to work on a project, neglect from users, or an abundance of counterproductive attention, reported issues and pull requests languish while waiting for a response. The project becomes undermaintained, still alive but limping. Here, too, bugs go unidentified or unaddressed.

## “It Sucks...”: The Unique Challenges of Open-Source Security

The general challenges of maintaining an open-source project are compounded by the additional unique challenges associated with security work. Voluntary labor allows interested individuals to pick and choose—self-select—the aspects of a project that they want to focus on. For many, security is simply not attractive. Aaron, a core contributor that does security work for both Rack and Ruby on Rails, bluntly explained why contributors avoided security work: “Because it sucks! It’s not fun.”<sup>62</sup> He recounted how, he had reluctantly stumbled into security work:

I didn’t really like [security work]. I wasn’t particularly interested or wanted to do it. But I don’t know...I was like, ‘Well, somebody’s got to do it and I can do it.’ So I did. And that’s basically how I’ve been involved in any security team.

Volunteers are driven by what they find interesting and fulfilling. Security, often, does not fit the bill. Working on security issues often cuts out some of the core individual benefits that participants derive from working on open-source projects: collaboration and feedback. Aaron expanded on the challenge:

[Security is] not a particularly fun task to have. So you have like I’ll try to describe the different aspects that make it unfun. First, you have to take every security report that comes to you, you need to take it seriously and investigate it thoroughly. And unfortunately, you have to do that basically in secret, because if it’s a real security vulnerability, you want to make sure that you get it fixed or have a fix for it before it’s more widely more widely known. And this is not fun because it means like you can’t get

...

...

you can't really get feedback from the community or other folks. So you're fairly isolated working on it.

The nature of security issues, requires a degree of isolation—these problems cannot always be fixed in the open. In some cases, projects silo these issues into a different triage and tracking system, and only particular authorized contributors are able to work on security issues. In this way, collaboration is made more difficult or significantly circumscribed.

Additionally, security work is both high-pressure and filled with potential time-wasting detours. As Aaron continued:

Second... so you do have to take these [issues] seriously, but you also get a lot of junk reports at the same time. So you're trying to filter the wheat from the chaff. And a lot of times you just get these crap reports, it just sucks.

Volunteers are driven by what they find interesting and fulfilling. Given this description, it is easy to see why many avoid security work.

Some projects are able to solve these problems by supporting paid staff to take on the important but unloved job of managing security and by developing security teams to enable collaboration. Django stands out as a useful example. While the bulk of their security team are volunteers, a small number of contractors help manage incoming security issues and develop fixes. This ensures that security remains an active priority, while allowing the volunteers to focus on issues that they find engaging and interesting (and sloughing off the bits they would rather not deal with). Shai, a member of the security team at Django, described his role:

I try to help resolve whatever issues that I can, when I have time for it and when usually also when I have interest, some of the issues that come up, I have nothing to contribute to... When all the stars align just right then, I do what I can to help.

Shai noted that many security issues that are reported to the project are, in fact, not really security issues at all: they are misunderstandings or non-issues. Wading through the pile of incoming reports is time consuming and tedious. For Shai, the work of contractors is absolutely essential: they have the time and the incentive to work through these reports and sort the serious from the trivial. Shai is clear about the importance of the

contractors: “These days, when I know that there are some people paid to do it, I trust them to be more patient than I am.”<sup>63</sup>

The structure of Django’s security team ensures that, for volunteers like Shai, this often unloved corner of open-source work remains interesting and engaging. Working with the security team—a group of about 10—enables collaboration. Security reports are isolated from the regular issues tracking system and delivered to this small working group, who are able to discuss and review the submissions in private. Even better, with the help of the invaluable paid contractors, volunteers can peel off the aspect of the work they do not enjoy and focus on what they find engaging. The work is not isolating for Shai; it remains fulfilling.<sup>64</sup>

But for thinly resourced projects, developing a security team with paid support may not be possible. In a purely voluntary model, self-selection leads some participants to avoid taking on the burden of security work.

<sup>63</sup> Shai B., Interview, 2024.

<sup>64</sup> Shai B., Interview, 2024.

# Bug Bounties and the Ongoing Remaking of Security Work

In some ways, little has changed since Fred Brooks' insight, noted above, decades ago: software testing is expensive, time-consuming, and imperfect. Finding and fixing bugs remains an ongoing challenge. Over the past two decades, bug bounty programs—programs that pay independent security researchers, “hackers,” that find and report novel bugs—have been adopted in an effort to improve this process. Bounty programs seek to embrace the wisdom of the crowd, rewarding hackers for their work while improving security at a reasonable cost for vendors.

As these programs are increasingly adopted by companies and governments, open-source projects have also begun to experiment with bounties as well. At first blush, they appear to offer an ideal solution for improving open-source security and confronting the challenges sketched above: they hold out the promise of improving engagement with otherwise neglected open-source projects, and they offer a way to help better manage submissions for popular projects. Yet, as an analysis of the bounty market reveals, these programs are not without drawbacks: in some instances, they can create real risks for hackers, organizations, and the public at large.

## Bounty Everything: The Rise of Bug Bounty Programs

Before turning to consider the benefits and risks of integrating bounties with open-source repositories, a brief overview of bug bounty programs—their history, organization, and associated advantages and hazards—follows. This overview and analysis are significantly indebted to Ryan Ellis and Yuan Stevens' earlier report, *Bounty Everything: Hackers and the Making of the Global Bug Marketplace*.<sup>65</sup>

### A Thriving Market for Flaws: An Overview of the Bounty Ecosystem

Bug bounties are a thriving business. Hundreds of organizations now offer rewards for novel bugs. Each year, thousands of hackers participate in these programs, spending countless hours hunting for previously unknown and undisclosed flaws. Companies pay millions, or in some cases, tens of millions of dollars each year to hackers through bounty programs.

Bounty programs are not particularly new. Netscape started the first widely-recognized one nearly 25 years ago.<sup>66</sup> After suffering a run of bad press regarding the security of their popular web browser, Netscape pioneered a new approach—paying hackers.<sup>67</sup> This first bounty program

<sup>65</sup> See Ellis and Stevens, *Bounty Everything* for a detailed overview.

<sup>66</sup> Ellis and Stevens, *Bounty Everything*, 28-33.

<sup>67</sup> *Ibid.*

was something of a stunt; it was an attempt to encourage security researchers to stop reporting new flaws publicly, where they were making headline news and eroding trust in Netscape.<sup>68</sup> In time, the model caught on. Mozilla, Netscape's successor, and tech companies, including Google, Facebook, and Microsoft would all eventually adopt a bounty model.<sup>69</sup>

The advent of bounty platforms—BugCrowd in 2011 and HackerOne in 2012—rapidly expanded the market.<sup>70</sup> Platforms recruited new companies and organizations and encouraged them to offer bounty programs; importantly, at the same time, they marketed these programs to hackers as a way to make extra income or as a full-time career.<sup>71</sup>

Exact details on the size and scope of the current market are hard to come by, but the piecemeal available numbers are nonetheless staggering. HackerOne, one of the largest platforms, hosts dozens of bounty programs for different companies and organizations and handles thousands of new reports each month.<sup>72</sup> To date, over 2 million hackers have registered to participate in bug bounty programs hosted on the platform.<sup>73</sup> Since HackerOne launched over a decade ago, it has paid over \$300 million in bounties across more than 400,000 valid submissions.<sup>74</sup> It is not just bounty platforms that have found success, Google, Facebook, and others offer their own, standalone or in-house, bounty programs. Since it first launched its program in 2011, Facebook (now, Meta) has paid hackers over \$16 million in bounties.<sup>75</sup> Google has more than tripled that figure over the life of their program, paying out over \$58 million to over 3,000 different hackers.<sup>76</sup>

Aided by the platforms and the growth in programs, bounty work has, for some, become a full-time job. BugCrowd reports that over 60% of the hackers that participate in their platform view bounties as a full-time occupation.<sup>77</sup> It is a global workforce. Hackers in India, the United States, and Nepal represent the top three countries by participation on BugCrowd's platform.<sup>78</sup>

The mechanics of bounty programs are fairly straightforward. At their most simple, they pay hackers for finding and reporting novel bugs. Programs define what assets and categories of bugs are in scope and set payout ranges for qualifying bugs. Submitted bug reports are reviewed and, if deemed to qualify, paid. Non-qualifying bugs, reports that are determined to be either out-of-scope, duplicate, invalid, or simply informational, are not paid.

Bounty programs are organized in different ways. Some are open to any and all hackers, while others

<sup>68</sup> Ibid.

<sup>69</sup> Ellis and Stevens, *Bounty Everything*, 38-43.

<sup>70</sup> Ibid.

<sup>71</sup> See Ellis and Stevens, *Bounty Everything*.

<sup>72</sup> HackerOne, "Why HackerOne?" Available Online: <https://www.hackerone.com/why-hackerone>.

<sup>73</sup> However, how many of these accounts are active is unclear. Ibid.

<sup>74</sup> HackerOne, "Why HackerOne?"; HackerOne, "Hackers Surpass \$300 Million in All-time Earnings on the HackerOne Platform," Oct. 26, 2023. Available Online: <https://www.hackerone.com/press-release/hackers-surpass-300-million-all-time-earnings-hackerone-platform#>.

<sup>75</sup> Neta Oren, "Looking Back at Our Bug Bounty Program in 2022," Dec. 15, 2022. Available Online: <https://about.fb.com/news/2022/12/metabug-bounty-program-2022/>.

<sup>76</sup> Google, "Our Greatest Achievements (So Far)," Aug. 8, 2022. Available Online: <https://bughunters.google.com/about/key-stats>.

<sup>77</sup> BugCrowd, *Inside the Mind of a Hacker 2023*. Available Online: <https://www.bugcrowd.com/wp-content/uploads/2023/12/inside-the-mind-of-hacker.pdf>. 8.

<sup>78</sup> BugCrowd, *Inside the Mind of a Hacker 2023*, 5. The international nature of the labor pool is consistent across platforms and programs. For example, India, Nepal, and Tunisia are the top three countries by country of origin for Facebook's program. Oren, "Looking Back at Our Bug Bounty Program in 2022."

are invitation-only. Additionally, programs operate either standalone or via an established bounty platform. For in-house operation, the vendor or organization pays for bugs in their own software applications and handles all elements of the program, from defining terms of service, triage of incoming reports, and payment. Bounty platforms host bounties for other organizations for a service fee or commission (the services they offer vary, but can include not only managing payment, but also overseeing triage, defining in-scope targets, and other program elements). Platforms encourage hackers to seek out bounty programs that they host through a number of means, including offering exclusive access to live events and private programs for active and successful participants.<sup>79</sup>

## The Benefits of Bounties: Improved Security and Flexible Work

Bounty programs offer significant benefits for vendors, hackers, and, by extension the public. Bounties are, at their core, a way of incentivizing hackers to review and test targeted assets. For vendors, these programs are tantalizing: they promise expert analysis at a fraction of the price of a contracted penetration test or internal audit. Pen test contracts are paid regardless of the results—a report that finds a dozen new flaws costs the same as a report that finds two. Similarly, internal security work that uses full-time employees carries all of the costs associated with this model of employment, namely defined benefits, regardless of the results of the outcome of the work. With bounty programs, hackers are only paid when they find a qualifying bug. The time they spend hunting down duplicate bugs, following dead ends, filing out-of-scope reports, is all uncompensated.

In talking with hackers, many are enthusiastic about the benefits of bounty programs.<sup>80</sup> They see them as a way to make money doing something they love. For those working in countries with lower annual salaries, bounties can offer a significant boon. For some, bounties are seen as a steppingstone, a way of getting a start in the competitive world of computer security. For others, the flexibility of bounty programs—working when you want, as much or as little as you like—provides an attractive work-life balance.<sup>81</sup>

Reducing the costs of creating secure software is not just in the interests of vendors and hackers; it is in the public interest. The costs of insecure software are not shouldered by software vendors. On the contrary, the current state of software liability ensures that the costs of insecurity are largely externalized and displaced onto the public.<sup>82</sup> Interventions, like bounty programs, that can improve security without significantly increasing costs are plainly in the public interest. Yet, bounty programs are not a panacea: they carry significant often unacknowledged risks.

<sup>79</sup> Ellis and Stevens, *Bounty Everything*, 67-72.

<sup>80</sup> See Ellis and Stevens, *Bounty Everything*.

<sup>81</sup> *Ibid.*

<sup>82</sup> See Executive Office of the President, *National Cybersecurity Strategy*, March 2003. Available Online: <https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf>. 20-21.

## Precarious Work and Precarious Technology: Creating and Propagating Risk

The earlier report, *Bounty Everything*, documented how bug bounty programs can, in some instances, create risks for participating hackers, vendors that seek to use bounties as a way of improving their security, and the public at large.<sup>83</sup> Before considering how or if bounties might be suited to the challenges of open-source security, it is worth pausing to consider the thorny issues that surround bounties in general.

For hackers, the rise of bug bounty programs can look like the advent of a new golden age. After decades of hostile legal threats or indifference from software vendors, bounties appear to offer a warm embrace: freedom to explore and tinker with the promise of a paycheck. Yet, as bounties have become a key source of income for many, and a full-time job for some, the reality is more complex. Bounty programs can reproduce the worst aspects of the gig economy: while vendors get access to high-quality work at a comparatively low price, workers take on significant risk.<sup>84</sup> Katie Moussouris, founder and CEO of Luta Security, and an expert in all things related to bug bounty programs, was direct: “It’s speculative work...It’s the worst gig economy job. An Uber or Lyft driver will get paid if they accept a ride and take you to the airport,” but for hackers working in bounty programs, payment is never certain.<sup>85</sup> Uncompensated labor is standard. Since hackers are only paid when they find a qualifying bug, the hours and late nights spent hunting for a novel flaw without success are unpaid work.

In this market, bounty programs and platforms wield significant power. They determine what counts as a valid submission, set price scales, and have what amounts to unilateral authority to determine when or if a bounty is paid. When they disagree with a triage decision—when a bounty program determines that a bug is merely informational or out-of-scope, and thus ineligible for payment—hackers have few avenues to push back.<sup>86</sup>

For software firms and vendors, the promise of bounty programs can quickly curdle. Creating financial incentives that encourage the public to report flaws can lead to a flood of ultimately unhelpful reports. Moussouris, who has played a formative role in establishing bug bounties for large companies and governments, notes that the widespread commercialization of bounties has led to an influx of low-quality reports and spam. She reflected on this trend:

<sup>83</sup> Ellis and Stevens, *Bounty Everything*.

<sup>84</sup> Ibid.

<sup>85</sup> Katie M., Interview, 2024.

<sup>86</sup> Ellis and Stevens, *Bounty Everything*.



[the popularization of bounty programs] began a process that I call 'beg bounty' where [participants] submit a lot of low-quality reports...And they would kind of spray and pray and hope for a cash reward because [it is] very low cost for them...And these beg bounty hunters are using this tactic to generate income and it's working.<sup>87</sup>

If not prepared, bounty programs can struggle to tirage and sort out what is valuable from all of the noise. Firms that seek out bounties as a way to improve their security can find themselves wading through a morass of low-quality reports—and blowing their planned bounty budget on a high-volume of minor issues in the process.

For organizations, improving security is ideally about more than simply fixing a single bug: it is about creating sustainable development practices that minimize the instance and impact of flaws. Responding to incoming report can be an important part of this development process, but extracting significant lessons from bug reports—integrating the findings into root cause analysis that can not only fix the issue at hand but help prevent the introduction of new bugs down the road—is not trivial. Some firms get stuck simply playing whack-a-mole, deploying point fixes that address the issue at hand but fail to get at the underlying causes of vulnerability.

Not only do they carry risks for hackers and firms, but they also carry risks for the larger public. Bounty programs can also, counter-intuitively, undermine security. In a world of scarce resources, the hours and dollars devoted to tracking down and responding to each and every new bug report might be better devoted to working to first develop and deploy secure software. Bounty programs can help mature organizations that have already invested in and instituted secure software development practices—they provide an extra added layer of security and protection. But, for organizations that have not yet made these investments, bounty programs actually preserve a world of faulty software and bugs: they provide an illusion of security while perpetuating a cycle of software development that privileges speed, market share, and cost over security.<sup>88</sup>

<sup>87</sup> Katie M., Interview, 2024.

<sup>88</sup> Ellis and Stevens, *Bounty Everything*.

# Open-Source Security Bounties: A Path Forward

Open-source security faces the common challenges that plague maintenance—a lack of maintainer time, too little attention, too much attention—and other challenges related to the perceived desirability of security work among open-source volunteers. Bounties can help address some of these issues in certain cases. Yet, risks abound: bounties can exacerbate the maintenance and security challenges that some projects currently face. If not deployed judiciously, bug bounty programs may do more harm than good. Clarifying these opportunities and risks can help chart a path forward that avoids pitfalls and improves security in ways that are sustainable.

## Shrinking the Window of Vulnerability: Improving Security through Open-Source Bounties

There are significant opportunities to enhance open-source security by improving the ability of projects to find, fix, and ultimately extract important insights from bugs. A number of maintainers that have experimented with integrating bounties into their projects find the experience to be very positive. Many have worked with the Internet Bug Bounty supported by HackerOne. In this program, HackerOne does not triage incoming reports; the projects themselves remain responsible for managing reports. The financial model differs from other bounty programs: the IBB is funded through donations from other companies—the open-source projects do not pay the bounty—and a portion of the money paid per bounty is also donated back to the target projects.

Bounties can draw attention to underserved open-source projects and help them keep their participants engaged. Daniel, the founder of curl, the popular and widely used command-line tool, turned to bounties in an effort to make the project as secure as possible. A conversation with a contributor led him to adopt a bounty model:

He mentioned at some point that he was not going to look at curl anymore... because he was going to get food on his table. He was going to go off and hunt for bugs in some projects that were actually giving him bounties.<sup>89</sup>

He turned to bounties as a way of keeping people like this engaged in the project. He also hoped that, perhaps, a bounty program would entice highly skilled experts that were otherwise not

<sup>89</sup> Daniel S., Interview, 2024.

engaged with curl to review it.<sup>90</sup> In his view, the bounty program has worked wonders, having “increased the frequency of [high quality] security reports.”<sup>91</sup> Daniel compared the findings generated through bounty program with costly security audits that curl had used in the past and it was not a close call: bounties turned up more CVEs at a far lower cost.<sup>92</sup>

Daniel’s positive experience with the IBB was echoed by others. Sarah at Django agreed, bounties help keep people engaged with the project and draw new contributors in.<sup>93</sup> Alex, a contributor on Rust, also maintained that bounties can be a positive way to draw participants into the project.<sup>94</sup> Shai, a member of the security team at Django, liked that bounties provide a way to acknowledge and reward participants. For projects that suffer from a lack of attention, they offer a possible balm: a way to draw new blood into the project. For projects worried about contributors leaving them, the added incentive of a bounty can entice those contributors to stick around.

Bounties can also help manage harmful attention that projects may receive—a key challenge that can undermine maintenance and security. Aaron, a core contributor to Ruby on Rails and other projects introduced above, spoke about the increased accountability that bounties enabled. The use of the HackerOne platform was a useful way to manage the abuse and harassment that sometimes followed having a bounty program. Aaron observed that, although he did not want to generalize, “people in the security research community are just assholes, pretty much.”<sup>95</sup> With money on the line, a degree of antagonism can appear, as some hackers are looking to get paid quickly. Partnering with HackerOne allowed him to assign negative reviews to hackers—an assessment that would damage their ranking on the platform with potentially impactful consequences. Here, the bounty platform introduced a degree of accountability that Aaron welcomed.

## The Risks of Open-Source Bounty Programs: The Case for Caution

Yet, the introduction of bounty programs into open-source projects is not without risk: for some projects, bounties can make things worse. Woven inside the testimonials of maintainers that have successfully integrated bounty programs into open-source projects are notes of caution. Bounty

<sup>90</sup> Ibid.

<sup>91</sup> Ibid.

<sup>92</sup> Ibid.

<sup>93</sup> Sarah B., Interview, 2024.

<sup>94</sup> Alex C., Interview, 2024.

<sup>95</sup> Aaron P., Interview, 2024.

programs can exacerbate the challenges that projects already face—increasing commitments for already time-strapped maintainers, drawing unhelpful and extractive attention to the project, and undermining the fragile reciprocity that underpins much of the open-source ecosystem.

Bounty programs can eat up what little precious time maintainers have available to work on their projects by drawing unhelpful attention to the project. Bounties are not a substitute for secure development.

Daniel, the founder of curl is enthusiastic in his support for open-source bounties, but only under limited circumstances. For new projects that were just finding their feet, bounties could well be a disaster. Starting a bounty too soon could be, in his words, “quite scary.”<sup>96</sup> He finds that they work best when they are the last piece of the security puzzle, not the first. Before launching a bounty program, it is important to first have in place strong security practices. These practices are not, in his account, mysterious:

[to achieve] good quality and good security it’s mostly just a matter of following all those engineering practices that we all know we should follow. We all know exactly how to do things securely but mostly when things go wrong is when we don’t do that right. We don’t follow them, we don’t do tests, we don’t do reviews, and we skip parts of it...<sup>97</sup>

By not having proper security practices in place, open-source projects are inviting disaster. The project is likely to be overrun with low-quality reports that it cannot possibly keep up with. Katie Moussouris underlined this theme: “Bug bounties ideally catch the vulnerabilities that your security development processes miss.”<sup>98</sup> But, if these processes are not yet in place or mature, a bounty program is going to make things exponentially worse—spam, minor issues, and other relevant and irrelevant reports will swamp the project.

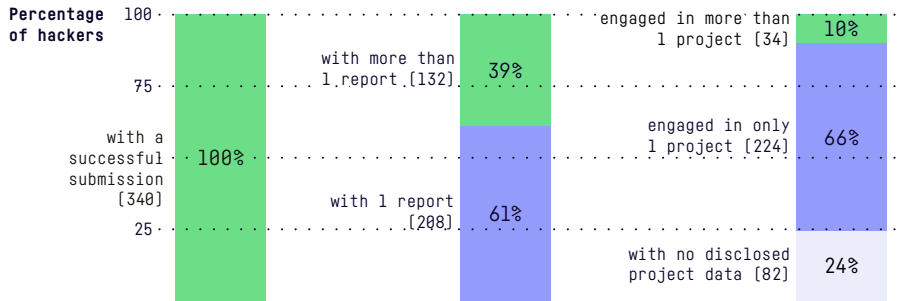
Bounty programs may add counterproductive or harmful attention to the project. Dealing with the volume of reports that follow a bounty program is a challenge, even for well-established projects that have security teams, clear disclosure policies and processes, and years of experience. Shai, a member of the Django security team, recounted the seemingly endless stream of emails reporting the same non-security issues over and over.<sup>99</sup> Daniel agreed that dealing with “crap” reports came with the territory. For established projects such as Django and curl, a deep bench of experience and clear policies make it possible for maintainers to wade through the inevitable junk that pours in thanks to the bounty program. Other projects might not be so well positioned. What little time maintainers have available to devote to their project, is going to be further eroded by tending to the work of triaging bug reports. This work—tirage—is not something that can be easily offloaded to a bounty platform or outsourced. Familiarity with and understanding of the projects—and their quirks and particularities—are needed. Project maintainers need to do this work. When time is already a scarce resource, adding this new task, managing bug reports, on top of their already busy and crowded schedule can undermine maintenance and, in the process, security.

<sup>96</sup> Daniel S., Interview, 2024.

<sup>97</sup> Daniel S., Interview, 2024.

<sup>98</sup> Katie M., Interview, 2024.

<sup>99</sup> Shai B., Interview, 2024.



➤ **Figure 4** HackerOne's Internet Bug Bounty Program

*Data drawn from 980 publicly available reports (touching 24 repositories) associated with the IBB on HackerOne's platform.*

Bounty programs incentivize participation, but they are unlikely to help projects with small contributor bases grow in meaningful ways. Unleashing economic rewards draws attention to a project, but bounties are unlikely to be a shortcut to growing a sustainable and supportive community. Aaron, the contributor to Rack and Ruby on Rails introduced above, worried that while bounties could help

“more established projects,” smaller ones would sink under the weight of unproductive engagement. As he noted, “most of the folks that are coming to the project... are just fishing for bounties essentially.”<sup>100</sup> Others observed that those that enter the project through bounty programs rarely, if ever, matriculate into other roles—they are just there for a bounty.

Indeed, a look at the public IBB data hosted on HackerOne indicates that most hackers in open-source bounty programs are infrequent participants—they typically only submit one successful bounty report. A clear majority—61%—of those that have received a bounty via the IBB program have only one successful submission (see Figure 4). These are not hackers that are necessarily going to have a long-term investment in the ongoing health and maintenance of the project.

Incentivizing this sort of casual, for-profit engagement threatens to undermine reciprocity. Bounties can amplify the frustrating cycle of harmful attention that undermines what many participants in open source prize: collaboration and community. Rather than investing time and energy into participating in the project, inevitably, some participants that encounter the project thanks to a bounty program will, in effect, be “spraying and praying,” that is: sending out batches of low-quality reports with the hopes of maybe getting paid. These contributions can take away more than they add. Additionally, the presence of bounties may drive a wedge through a project. Once certain participants start getting paid for their contributions, others that have been toiling and donating their labor on a voluntary basis may become resentful.

There was agreement on one critical point regarding the viability of open-source bounty programs: funding is key.

A number of maintainers were only able to support bounty

<sup>100</sup> Aaron P., Interview, 2024.

programs thanks to the IBB's unique funding model, agreeing that the program would not be possible without external financial backing. Aaron noted that IBB support made it possible to fund the increasing number of reports that were coming in: "I don't know how many of these ReDoS attacks we paid out, but it's a lot. And [I'm] glad it's not my money we're spending."<sup>101</sup> Daniel agreed with this sentiment. Initially, curl experimented with a different bounty model before being brought into the IBB. But the IBB's model of financial support was critical. Now, in his words, the financial impact of a high number of submissions is not a worry: "it doesn't really matter if we get a lot of security problems. The amount of money we pay is not a problem for us."<sup>102</sup> Without funding support through programs such as the IBB, bounties are largely unsustainable for open-source projects.

## Best Practices and a Path Forward: A Future for Open-Source Bounties

Open-source security is a significant challenge: given the ubiquity of open-source components in commercial software applications and the importance of open-source software as standalone tools, it is difficult to talk about software security without addressing open-source security. Indeed, the two are now inseparable. Bug bounty programs can help improve open-source security in specific instances, but there are good reasons to proceed with caution. Bounty programs can unintentionally undermine security. If not deployed judiciously, they can amplify the challenges that open-source projects already face.

The following five recommendations are drawn from the proceeding analysis. They provide a set of coordinates that can help chart a path forward. Bounty programs are, of course, not a silver bullet. They are but one of many possible security interventions. As with many complex problems, a cross-section of solutions and approaches is needed to tackle the challenges of open-source security. These recommendations are designed to help foundations, policymakers, open-source projects, funders, and others in sorting out when and how to deploy open-source bounty programs, and when to turn to other tools and approaches.

### **Recommendation #1: Invest in Holistic Approaches to Maintenance**

Open-source security is closely tied to larger question of maintenance. Poorly maintained projects are, inevitably, insecure. Investing in and improving maintenance has spillover effects that also improve security. To be clear, most projects *are* struggling.

Their performance over time is declining across nearly all key measures—it is taking them longer to resolve

<sup>101</sup> Aaron P., Interview, 2024.

<sup>102</sup> Daniel S., Interview, 2024.

issues and longer to respond to pull requests, they are shedding contributors, and the number of commits is trending downward. The trajectory for most open-source projects is not promising.

For these projects, improving baseline maintenance capacity leads to security gains above what a bounty program can offer. For projects that currently struggle to respond to submitted issues, resolve pull requests, or attract active contributors, a bounty program is likely do more harm than good. Bounties provide new economic incentives that draw unhelpful attention to the project, while saddling maintainers with additional work that they likely do not have the time to manage. For foundations, corporations, governments, and others looking to invest in open-source security, continuing to seek out opportunities to buttress maintenance is worthwhile. These interventions can and should be considered as not just investments in the open-source ecosystem, but investments in security.

### **Recommendation #2: Bounty Last, not First**

Bug bounty programs work best when they are the layer on top of already-existing security practices. Bounties are not a substitute for secure development practices; and they should not be deployed before these practices are in place. Adopting best practices, such as the processes associated with security development lifecycle (SDL) or secure software development framework (SSDF), can ensure baseline security measures are in place.<sup>103</sup> For mature projects, bounty projects can provide an added layer of defense and new insights that can be integrated back into the development process in a cycle of ongoing improvement.

Adopting a bug bounty program before achieving maturity, however, undermines security. Starting a bounty program too soon can lead to a near-endless stream of reports regarding minor issues. These issues can overwhelm project participants. Immature projects not only struggle to review and respond to submitted bugs; they falter in their ability to extract meaningful lessons and insights from new reports. Superficial errors may be fixed, but the root causes will remain unaddressed.

<sup>103</sup> For an overview of the secure development lifecycle approach, see: Microsoft Security Engineering, "Security Development Lifecycle (SDL) Practices." Available Online: <https://www.microsoft.com/en-us/securityengineering/sdl/practices>. For an overview of the secure software development framework, see: Murugiah Souppaya, Kare Scarfone, and Donna Dodson, *Secure Software Development Framework (SSDF) Version 1.1.: Recommendation for Mitigating the Risk of Software Vulnerabilities*, NIST Special Publication 800-218. Feb. 2022. Available Online: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf>.

### **Recommendation #3: Leverage Bounty Programs to Improve Identification**

Not all mature projects derive the same benefits from a bug bounty program. Identifying which projects may be the most attractive candidates for bounty programs is critical. A look at the sampled GitHub and CVE data indicates that there are ripe opportunities to improve security by using bounty programs to help speed up the initial identification and reporting of vulnerabilities to impacted repositories.

Looking at how different projects respond to known vulnerabilities is a useful, albeit rough, indicator of security performance. Programs that are quick to fix known flaws perform, overall, better than their peers. Fixing a known flaw in a project's code follows a timeline that involves three key steps: (1) the initial disclosure and publication of the bug (CVE publication date, or *pd*); (2) the identification of the bug in a particular project's code (identification date, or *id*); and (3) the deployment of an update that resolves the bug by the impacted project (resolve date, or *rd*). This entire timeline, *Time to Fix*, can be described as the difference between the resolve date (*rd*) and CVE publication date (*pd*).

The timeline can be split into two segments: (1) identifying the bug within a project; and (2) creating and deploying the update. This first segment—identifying the bug—is the period between *pd* and *id* (*Time to Identify* =  $id - pd$ ). The second segment—creating and deploying the update—is the period between *id* and *rd* (*Time to Resolve* =  $rd - id$ ). The entire process, *Time to Fix*, encompasses the full timeline (*Time to Fix* =  $rd - pd$ ).

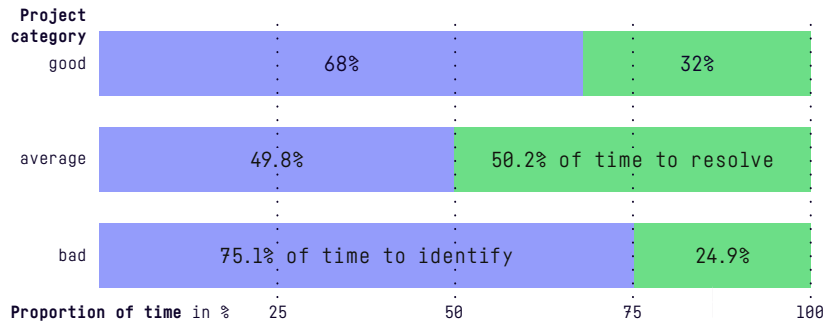
Examining the composition of the *Time to Fix* timeline reveals pain points and opportunities to improve performance. This analysis uses equal quantiles to categorize projects into “good,” “average,” and “bad” categories. Good projects are those in the fastest third (bottom 33%) in terms of median CVE resolution time, average projects fall within the middle third (33% to 67%), and bad projects are in the slowest third (top 33%). Examining how different projects within these categories are dividing their time—How much time are they spending identifying vulnerabilities in their repository? How much time are they spending developing resolutions to reported flaws?—is instructive. Projects that fall into the “average” category spend an equal amount of time on identification and resolution (see Figure 5), while those in both the “good” and “bad” categories spend a disproportionate amount of time identifying the presence of published vulnerabilities in their projects.

The analysis reveals that while average performers maintain a balance between identification and resolution, both strong and below-average projects are notably less efficient, spending significantly more time on the initial identification phase. The takeaway from this analysis is plain: there is significant room to improve the security performance of projects by investing in tools and processes that can help more quickly identify bugs in open-source projects. Bounty programs should target mature projects that are nonetheless spending a disproportionate amount of time on identification. Speeding up initial identification can reduce exposure and improve security.

#### **Recommendation 4: Open-Source Bounty Programs Should Adopt Ethical Practices**

Bounty programs should recognize the unique characteristics of the open-source ecosystem and take care to enhance rather than erode reciprocity.





➤ **Figure 5** Time to Fix:  
Comparing Time to Identify  
and Time to Resolve

In all instances, they should be designed to minimize the risks for participating hackers—these steps should include ensuring legal protections, adopting transparent processes around payment and review, offering dispute mediation processes, and other measures.<sup>104</sup> Bounty programs introduce additional risks in an open-source context: they can disrupt and undermine reciprocity. Open-source projects largely rely on voluntary labor. These contributions are often offered in the spirit reciprocity: they are free for other to make use of, but the hope is that some will offer their own contributions that will then be freely available (typical open-source licenses make this commitment explicit). Bounties selectively provide payment for some identified subset of project activities, while leaving others uncompensated. This can undermine reciprocity and create ill-will within the project between those that are being paid and those that are not.

Open-source projects have navigated similar challenges with success. As sponsorships, grants, and corporate support interact with open-source projects, it is common for paid and unpaid workers to collaborate. To ensure that bounties do not undermine reciprocity, it is critical that bounty programs are designed and embraced by the existing project community. Programs that are organic to the project—not bolted on without consultation with current participants—can and should address questions of equity and reciprocity. These conversations ought to precede the adoption of any bounty program. Failing to do so risks alienating the current project community in ways that are both undesirable and counterproductive to security.

### **Recommendation #5: Bounty Funding Should Be Community-Driven and Aid Structural Support**

Open-source bug bounty programs require resources that are beyond the capacity of most open-source projects. Funding for these programs should come from the larger community of users that benefit and make use of open-source technologies. Despite efforts of

<sup>104</sup> For a detailed discussion of how to mitigate the general risks that bounty programs can displace onto researchers, see Ellis and Stevens, *Bounty Everything*.

corporations, governments, and other organizations to create open-source program offices and invest in open-source programs, contributors still perceive a lack of reciprocity from large users, particularly profitable corporations. Bounty programs can be one avenue—among many—through which users can give back to and support open-source programs.

In order to avoid the unintended consequences and the distorting power of economic incentives, bounties should be paired with general investments in the maintenance of the open-source project. Bounties create new work for projects: at minimum, they must triage incoming bug reports and develop new fixes. Without supporting or offsetting this new work, the benefits of the bounty programs can be lost. Adding new responsibilities without new capacity is unlikely to improve the maintenance or security of an open-source project. The IBB is one useful model of how to make sure that bounty programs also provide aid to maintainers and the larger project. In the IBB, companies and organizations contribute to the revolving bounty fund; payments are distributed *both* to the hacker that identifies the previously unknown issue and to the project (the payment to the project is a percentage of the bounty). The funding is offered up with no limitation or direction—projects can use it as they see fit. This model seeks to minimize or offset the new work involved with managing the overhead associated with a bounty program through paired general support.



# Measuring Open-Source Maintenance

---

To understand the general maintenance challenges of projects, we analyzed key maintenance metrics, including median issue resolution time and median pull request (PR) resolution time across sampled GitHub projects. The data shows significant variability in how quickly projects resolve issues and PRs.

We focus on understanding the variability in maintenance performance across different open-source projects by analyzing key maintenance features. These features include metrics related to commit frequency, issue resolution, and pull request handling, which are critical indicators of project maintenance.

## Metrics

---

**Quarterly Commit Slope** Measures the rate of change in commit activity over time. A positive slope (↗) indicates increasing activity, while a negative slope (↘) indicates decreasing activity.

**Issue Time Slope** Captures whether projects are resolving issues more quickly or slowly over time. A negative value (▶) indicates improvement, while a positive value (■) indicates delays.

**PR Time Slope** Similar to Issue Time Slope, but for pull requests. It reflects whether PRs are being handled more efficiently over time.

**Quarterly Median Active Contributors** Shows the median number of active contributors participating in the project each quarter. This is an indicator of the project's community health and engagement.

**Median Issue Resolution Time** The average time it takes to resolve

issues in a project. Lower values indicate quicker response times.

**Median PR Resolution Time** The average time it takes to resolve pull requests. Faster PR resolution times suggest a more responsive project.

## Statistics Glossary

---

**Mean** The average value for each metric.

**Median** The median is the middle value in a sorted dataset. It represents the point at which half the data points are below, and half are above, making it less sensitive to outliers compared to the mean.

**Standard Deviation (std)** Illustrates how much variation exists from the mean.

**Min** The lowest value observed, indicating the best performance in this metric.

**25% (1<sup>st</sup> Quartile)** The value below which 25% of the data fall.

**50% (Median)** The middle value of the dataset. Projects at this point are performing averagely.

**75% (3<sup>rd</sup> Quartile)** The value below which 75% of the data fall.

**Max** The highest value observed, representing the poorest performance in this metric.

**Variance** The degree of spread or variability in the data. A higher variance means the data is more spread out, while a lower variance means values are more clustered around the mean.

Statistics	Quarterly commit slope	Issue time slope	PR time slope	Quarterly median active contributors	Median issue resolution time [days]	Median PR resolution time [days]
Mean	↘ -2.67	■ 14.72	■ 10.43	5.64	▬ 20.16	▬ 9.19
Median	↘ -1.12	■ 3.78	■ 0.79	3.00	▬ 9.00	▬ 1.00
Std	↗ 20.80	■ 46.86	■ 44.29	11.33	▬ 38.36	▬ 34.16
Min	↘ -335.50	▶ -647.15	▶ -807.00	1.00	▬ 0.00	▬ 0.00
25%	↘ -3.66	■ 0.36	● 0.00	2.00	▬ 4.00	▬ 0.00
50%	↘ -1.12	■ 3.78	■ 0.79	3.00	▬ 9.00	▬ 1.00
75%	↘ -0.10	■ 13.50	■ 6.96	5.00	▬ 21.00	▬ 4.50
Max	↗ 415.00	■ 879.00	■ 972.00	271.00	▬ 729.00	▬ 619.50
Variance	432.63	▲ 2,195.52	1,961.72	128.35	1,471.28	1,167.19

↑ Figure 6 Measuring Maintenance Activity - Data and analysis drawn from 3,707 active repositories.

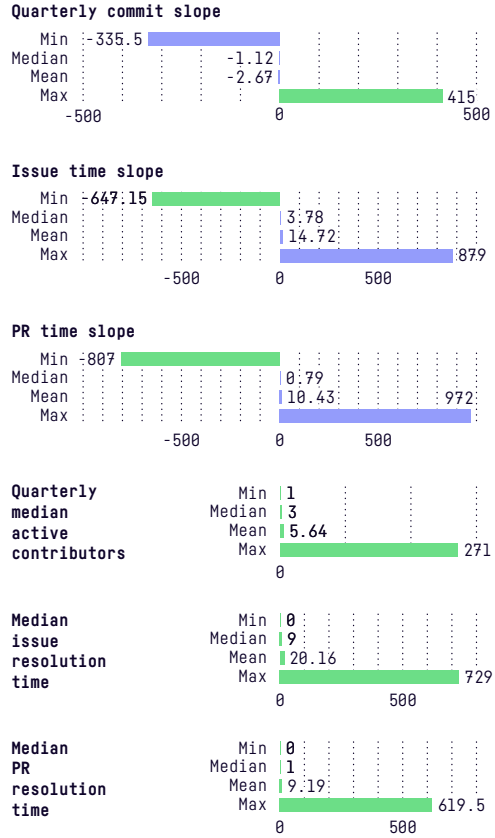
### Statistical Insights

- Quarterly Commit Slope** On average, commit activity is slightly decreasing with a mean of -2.67 (↘). The *high variance of 432.63* indicates significant variability between projects. While some projects are accelerating their commit frequency, others are experiencing substantial slowdowns, with a minimum slope of -335.5 (↘) and a maximum of +415.0 (↗).
- Issue Time Slope** The mean value of +14.72 (■) shows that, on average, projects are seeing an increase in the time taken to resolve issues. However, the large variance of 2,195.52 (▲) suggests there is considerable divergence: some projects are improving, while others face severe delays in resolving issues. The extreme range, from -647.15 (▶) to +879.0 (■), reflects this challenge.
- PR Time Slope** Similar to issue resolution, pull requests are taking longer to resolve, with a mean slope of +10.43 (■). The *high variance of 1,961.72* again highlights the wide performance gap across projects. The slope ranges from -807.0 (▶) to +972.0 (■), showing that some projects have greatly improved PR resolution, while others struggle significantly.
- Quarterly Median Active Contributors** The average number of active contributors per project is 5.64, with a *variance of 128.35*, which indicates that while most projects have small teams, a few larger projects have significantly higher numbers of active contributors. This discrepancy is seen in the range from 1.0 to 271.0 contributors per project.
- Median Issue Resolution Time** The median issue resolution time is 20.16 days, but the variance of 1,471.28 reveals considerable disparity. Some projects are resolving issues very quickly (minimum 0.0 days), while others are taking much longer, with a maximum of 729 days.

The variability in issue resolution times indicates that while some projects manage issue resolution efficiently, others struggle with long delays.

- Median PR Resolution Time** On average, pull requests are resolved faster than issues, with a median resolution time of 9.19 days. However, the *variance of 1,167.19* demonstrates significant inconsistency across projects, with some resolving PRs almost immediately (minimum 0.0 days), while others take as long as 619.5 days. This reflects variability in how efficiently different projects manage their pull requests.

Taken together, these statistics emphasize the variability in the maintenance and performance of open-source projects. While some projects exhibit efficient resolution times and consistent contributor activity, others face substantial delays and maintenance challenges, contributing to the overall complexity in project management.



↑ **Figure 7** Visualizing Maintenance Metrics and Trends

Data and analysis drawn from 3,707 active repositories.

# Testing Maintenance and Security

---

Collected GitHub data and CVE information can be used to test the relationship between maintenance practices and security performance within open-source projects. Specifically, we aim to explore whether projects that are well-maintained—those with regular commits, timely pull request (PR) management, and active issue resolution—are more secure. Security performance is assessed by how quickly a project identifies and resolves known vulnerabilities, as measured by the median CVE resolution time.

To address this, we used logistic regression to predict a project's security classification based on its maintenance features. This approach allows us to quantitatively assess the relationship between maintenance-related metrics (such as PR handling and commit activity) and security outcomes.

## Data Collection and Feature Selection

---

To perform this examination, we gathered two main types of data from sampled open-source repositories:

1. **Maintenance Data** This includes features that provide insight into how a project is maintained, including:
  - **PR Time Slope** a measurement of how PR resolution times change over time;
  - **Quarterly Median Commits** a count of the median number of commits made to the repository per quarter, indicating the project's development activity;

- **Extended Quarterly Median Commits** a longer-term view of commit activity;
- **Open Issues Count** a measure of the total number of open issues, providing a sense of the project's workload or backlog;
- **Stargazers Count** a proxy for community interest in the project;
- **Quarterly Median Active Contributors** a measure of how many active contributors are involved in the project.

2. **Security Data** This includes the *median CVE resolution time*, which serves as a useful stand-in for the project's security performance. Generally, projects that resolve CVEs quickly are considered to have better security practices.

## Project Classification

---

To establish a clear relationship between maintenance and security, we next categorized projects based on their median CVE resolution time using equal quantiles:

**Good Projects** projects that fall into the bottom 33% of median CVE resolution times (i.e., those that resolve security vulnerabilities the fastest);

**Average Projects** projects that fall into the middle 33%;

**Bad Projects** projects that fall into the top 33% of median CVE resolution times (i.e., those that resolve vulnerabilities the slowest).

This classification allowed us to compare the maintenance practices of projects across different levels of security performance.

## Modeling with Logistic Regression

---

To predict the security classification (good, average, or bad), we employed a logistic regression model. Logistic regression is well-suited for this analysis because it provides probabilities for classification, allowing us to assess how different maintenance features contribute to a project's likelihood of falling into a particular security category.

The process involved several steps:

1. **Feature Selection** We selected key maintenance-related features (e.g., PR Time Slope, Quarterly Median Commits, and others) to be used as predictors in the model.
2. **Cross-Validation** We used 5-fold stratified cross-validation to evaluate the performance of the model. This ensures that the model is tested on different subsets of the data, reducing the risk of bias and providing a more robust estimate of its performance.
3. **Metrics** We measured the model's performance using three key metrics:
  - **Cross-Validation Accuracy** The proportion of correct predictions made by the model across different folds of the data;
  - **Coverage** The proportion of the dataset that was used for each feature combination, ensuring that the model is evaluated across a large portion of the dataset;

- **Accuracy-to-Coverage Ratio**  
This metric balances the model's accuracy with the amount of data it was able to cover. A higher ratio indicates a better trade-off between accuracy and coverage.

## Feature Impact Analysis

---

To determine which maintenance features had the greatest impact on security performance, we evaluated different combinations of features using logistic regression. We focused on both single-feature and two-feature combinations to identify which metrics were most predictive of strong security outcomes.

The top ten feature combinations based on cross-validation accuracy are listed in **Figure 8**.

## Key Insights

---

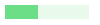


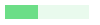


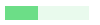


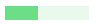


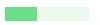


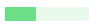


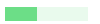


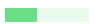


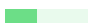


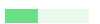


The analysis demonstrates that certain maintenance practices are strongly correlated with better security performance:

*PR Time Slope and Quarterly Median Commits* consistently appeared among the top-performing feature combinations. These metrics reflect both efficient PR management and regular development activity, which are crucial for maintaining project health and security.

*Extended Quarterly Median Commits* provides a longer-term view of commit activity and also appeared frequently in the top combinations, indicating the importance of sustained development over time.

*Open Issues Count and Median PR Resolution Time* also contribute to security performance, especially when combined with regular commit activity.



Feature combination	CV accuracy [%]	Coverage [%]	Accuracy-to-Coverage ratio
PR time slope + quarterly median commits	 39.63	 99.94	 0.40
PR time slope + extended quarterly median commits	 39.64	 99.94	 0.40
Quarterly median commits + median pr resolution time	 39.14	 100.00	 0.39
Extended quarterly median commits + median pr resolution time	 39.31	 100.00	 0.39
Extended quarterly median commits	 38.3	 100.00	 0.38
Extended quarterly median commits + stargazers count	 36.97	 100.00	 0.37
Extended quarterly median commits + open issues count	 37.61	 100.00	 0.38
Quarterly median commits	 38.18	 100.00	 0.38
Quarterly median commits + open issues count	 37.73	 100.00	 0.38
Pr time slope + quarterly median active contributors	 39.55	 99.94	 0.40

↑ **Figure 8** Testing the Connection Between Security and Maintenance

*Data and analysis drawn from 3,707 active repositories. Table summarizes the effectiveness of various maintenance metrics to predict security outcomes.*

*This data in this figure were corrected by the authors after printing and so appear correctly only in the digital version. The updated numbers do not in any way change the conclusions or findings.*

## Conclusion

The results of the logistic regression analysis indicate a clear correlation between strong maintenance practices and security performance. Projects that are regularly maintained, with high commit activity and efficient handling of pull requests and issues, tend to have shorter CVE resolution times, making them more secure. The *PR Time Slope* and *Quarterly Median Commits* features, in particular, emerged as critical indicators of a project's security posture.

This analysis supports the conclusion that *well-maintained projects are more secure*, as robust maintenance practices help ensure that vulnerabilities are identified and addressed in a timely manner.

### **Acknowledgments**

The authors wish to thank and acknowledge the contributions of Yuan Stevens.

The earlier round of interviews with bug bounty participants were conducted jointly by Ryan Ellis and Yuan Stevens for their coauthored Data & Society report, *Bounty Everything: Hackers and the Making of the Global Bug Marketplace* (2022). That report provides the foundation for much of this work. Yuan's contributions are greatly appreciated.

This report would not be possible without the participation of the dozens of interviewees that kindly shared their experience. Their insights and generosity are acknowledged and deeply appreciated.

The support of the Sovereign Tech Fund was critical to the conception and execution of this report. Paul Sharratt and Tara Tarakiyee provided useful feedback, suggestions, and contributions that helped shape and improve the final document.

Additionally, the authors wish to acknowledge additional funding support from the National Science Foundation. This material is based upon work supported by the National Science Foundation under Grant Nos. 1915815, 1935520, and 2203175. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

### **Authors**

Ryan Ellis  
Jaikrishna Bollampalli

### **Sovereign Tech Fund**

SPRIND GmbH  
Federal Agency  
For Disruptive Innovation

Lagerhofstraße 4  
04103 Leipzig  
Germany

### **Editing**

Supertext Deutschland GmbH

### **Layout and Design**

Village One eG × Nina Bender

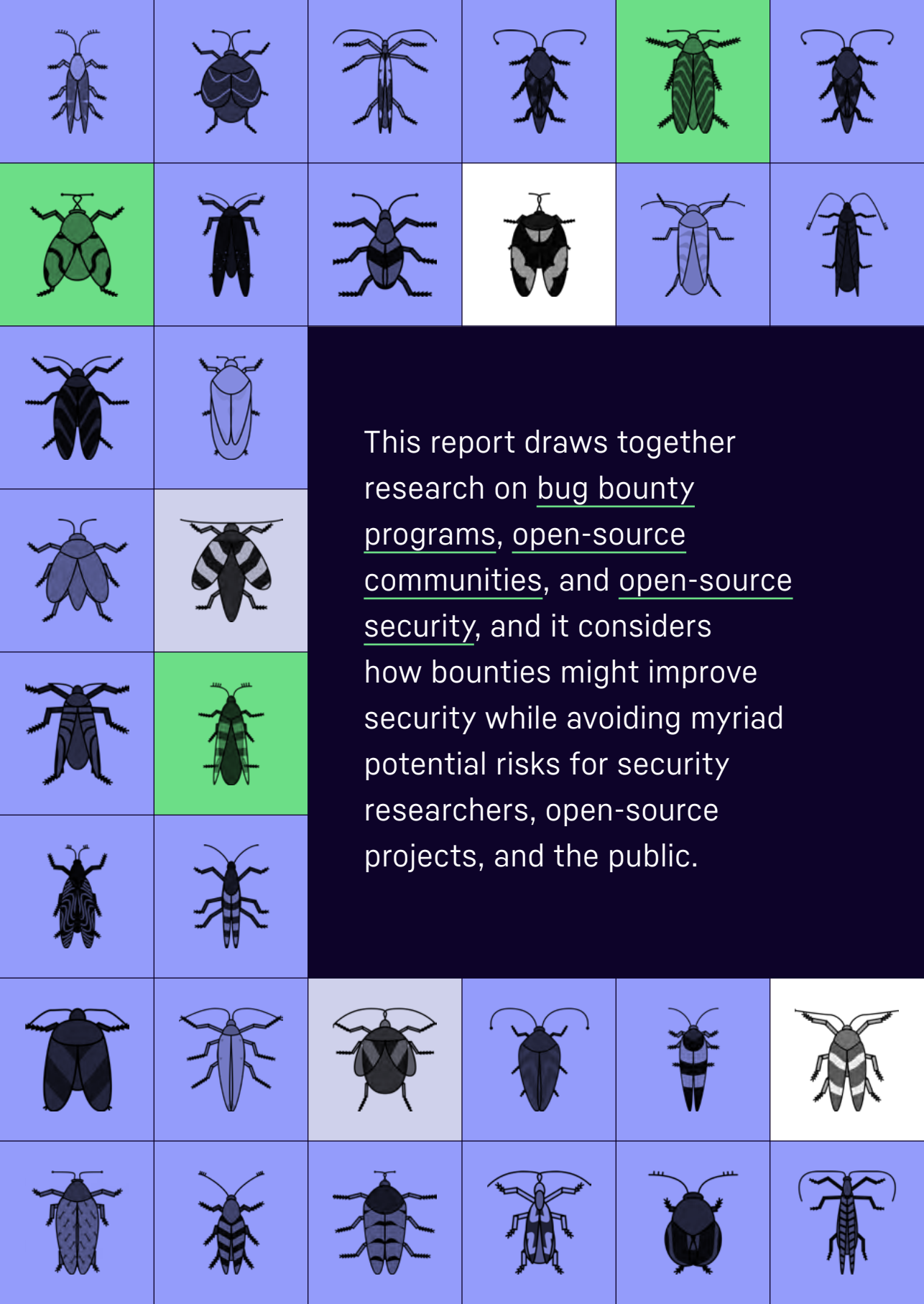
### **Printing**

Printed in Germany at Pinguin Druck, Berlin

### **Fonts**

Fabrikat Normal & Fabrikat Mono  
by HvD Fonts





This report draws together research on bug bounty programs, open-source communities, and open-source security, and it considers how bounties might improve security while avoiding myriad potential risks for security researchers, open-source projects, and the public.