The Benefits of Vulnerability Discovery and Bug Bounty Programs: Case Studies of Chromium and Firefox

Soodeh Atefi
University of Houston
USA

Amutheezan Sivagnanam
Pennsylvania State University
USA

Afiya Ayman
Pennsylvania State University
USA

Jens Grossklags
Technical University of Munich
Germany

Aron Laszka
Pennsylvania State University
USA

ABSTRACT

Recently, bug-bounty programs have gained popularity and become a significant part of the security culture of many organizations. Bug-bounty programs enable organizations to enhance their security posture by harnessing the diverse expertise of crowds of external security experts (i.e., bug hunters). Nonetheless, quantifying the benefits of bug-bounty programs remains elusive, which presents a significant challenge for managing them. Previous studies focused on measuring their benefits in terms of the number of vulnerabilities reported or based on the properties of the reported vulnerabilities, such as severity or exploitability. However, beyond these inherent properties, the value of a report also depends on the probability that the vulnerability would be discovered by a threat actor before an internal expert could discover and patch it. In this paper, we present a data-driven study of the Chromium and Firefox vulnerability-reward programs. First, we estimate the difficulty of discovering a vulnerability using the probability of rediscovery as a novel metric. Our findings show that vulnerability discovery and patching provide clear benefits by making it difficult for threat actors to find vulnerabilities; however, we also identify opportunities for improvement, such as incentivizing bug hunters to focus more on development releases. Second, we compare the types of vulnerabilities that are discovered internally vs. externally and those that are exploited by threat actors. We observe significant differences between vulnerabilities found by external bug hunters, internal security teams, and external threat actors, which indicates that bug-bounty programs provide an important benefit by complementing the expertise of internal teams, but also that external hunters should be incentivized more to focus on the types of vulnerabilities that are likely to be exploited by threat actors.

CCS CONCEPTS

• **Security and privacy** → *Economics of security and privacy*; Software and application security; • **Information systems** → Browsers; World Wide Web.

KEYWORDS

security, vulnerability discovery, bug bounty, vulnerability reward program, Chrome, Mozilla, web browser, technology policy

1 INTRODUCTION

Despite significant progress in software-engineering practices, the security of most software products and services remains imperfect in practice. Traditionally, testing the security of software products and services was the responsibility of internal security teams and external penetration-testing teams. However, these efforts are necessarily limited in their size and in the range of expertise applied. This limitation puts defenders at a disadvantage compared to attackers since publicly-available products and services may be targeted by myriads of attackers, who possess diverse expertise (e.g., different attackers may be familiar with different techniques).

Spearheaded by Netscape as a forerunner in 1995 [13], *bug-bounty programs*—which are also known as *vulnerability reward programs*—have emerged as a key element of many organizations' security culture [18, 25, 35]. Bug-bounty programs are a form of crowdsourced vulnerability discovery, which enables harnessing the diverse expertise of a large group of external bug hunters [13]. A program gives hackers the permission to test the security of a software product or service and to report vulnerabilities to the organization sponsoring the program [20]. By rewarding valid reports with bounties, the program incentivizes hackers to spend effort on searching for vulnerabilities and reporting them [1, 36]. In addition to enabling the sponsoring organization to fix security vulnerabilities before they could be exploited, a bug-bounty program also publicly signals the organization's commitment to continuously improving security.

However, *quantifying the benefits of a bug-bounty program remains elusive*, which presents a significant challenge for managing them. A number of prior research efforts have investigated bug-bounty programs (e.g., Finifter et al. [13], Zhao et al. [35], Laszka et al. [19, 20], Maillart et al. [22], Luna et al. [21], Elazari [9], Malladi and Subramanian [23], and Walshe and Simpson [33]). However, a common limitation of previous studies is that they typically measure the value provided by a bug-bounty program in terms of the number of vulnerabilities reported or, in some cases, based on the inherent properties of the reported vulnerabilities, such as severity or exploitability. As we discuss below, the number of reported vulnerabilities and their inherent properties alone cannot quantify security benefits since they ignore the *likelihood of discovery*.

While some vulnerability reports provide immense value to organizations by enabling them to patch vulnerabilities before threat actors would exploit them, other reports might provide relatively low value. First, *some vulnerabilities might be discovered anyway by internal security experts* before any threat actors could exploit them. Reports of such vulnerabilities provide low value since organizations could patch these vulnerabilities before exploitation without spending funds to reward external bug hunters. Second,

some vulnerabilities might never be discovered by threat actors. Patching such vulnerabilities is futile; in fact, it could even be considered harmful since patches can reveal the existence of vulnerabilities to threat actors [29]. Finally, even if some vulnerabilities are eventually discovered by threat actors, discovery might take so long that the software components become obsolete before the vulnerabilities could be exploited [6, 30]. In contrast, other software projects may have relatively stable code bases over time, which also dominates the number of discovered vulnerabilities [28]. In light of these considerations, the value of a vulnerability report hinges not only on the inherent properties of the vulnerability, such as severity, but also on the *probability that the reported vulnerability would be exploited by threat actors before another benign actor would report it.*

Research Questions. **Benefits of vulnerability discovery (RQ1):** To study the issues mentioned above, we consider the probability of rediscovery, that is, the probability that a previously discovered vulnerabilities is independently rediscovered by another bug hunter. The probability of rediscovery should be a key consideration for bug-bounty and vulnerability management since known vulnerabilities have a negative impact only if they are (re-)discovered by threat actors before they are patched (and before the patches are applied by users). In fact, some prior works suggested that vulnerabilities should not be patched proactively because patching only brings them to the threat actors' attention. According to this view, proactive vulnerability patching and bug bounties would provide little value [29]. However, this proposition holds only if the probability of rediscovering a vulnerability is negligible. Schneier [31] conjectures, in contrast, that when one "person finds a vulnerability, it is likely that another person soon will, or recently has, found the same vulnerability." Indeed, based on studying Microsoft security bulletins, Ozment finds that vulnerability rediscovery is non-negligible; but this result is based on a small sample (14 rediscovered vulnerabilities, constituting 7.69% of all vulnerabilities listed in the bulletins) [27]. In contrast, we characterize rediscovery probabilities based on thousands of vulnerability reports and thereby respond to Geer's call to conduct longitudinal research in this context [15].

- RQ1.1: Are vulnerabilities rediscovered? Are vulnerabilities more difficult to find, in terms of rediscovery probability, in stable releases than in development ones?
- RQ1.2: Are vulnerability discoveries and rediscoveries clustered in time, or is rediscovery a "memory-less" process?

Benefits of bug bounties (RQ2): If external bug hunters work similarly to internal security teams and discover similar vulnerabilities, then bug-bounty programs provide relatively low security benefits, and internalizing vulnerability-discovery efforts might be more efficient than sponsoring bug-bounty programs. However, theoretical work by Brady et al. suggests that there are efficiency benefits to testing software products in parallel by different teams that likely use different test cases and draw on different types of expertise [5]. Supporting this view, Votipka et al. report key differences between internal security testers and external bug hunters based on a survey of 25 participants, focusing on how each group finds vulnerabilities, how they develop their skills, and the challenges that they face [32]. In contrast, we focus on the actual vulnerabilities reported by these groups to facilitate the quantification of security benefits from the perspective of a sponsoring organization.

- RQ2: Do external bug hunters report different types of vulnerabilities than internal discoveries?

Management of vulnerability discovery and bug bounties (RQ3): The objective of both external and internal vulnerability discovery is to find and patch vulnerabilities that would be found by threat actors (since patching vulnerabilities that threat actors would not find provides a much lower security benefit).[1] Hence, the benefits of running bug-bounty programs hinge on whether bug hunters find the same set of vulnerabilities that the threat actors would find. If there is a significant discrepancy, bug-bounty managers must try to steer bug hunters towards discovering the right types of vulnerabilities, e.g., using incentives.

- RQ3.1: Do bug hunters report similar types of vulnerabilities than those that are being exploited by threat actors?
- RQ3.2: Which types of vulnerabilities are the most difficult to discover?

To answer these questions, we collect vulnerability data from the issue trackers of two major web browsers, Chromium (i.e., the open-source project that provides the vast majority of code for Google Chrome) and Firefox. We combine these with other datasets and apply a thorough data cleaning process to reliably determine which reports are internal and which are external, which releases and components are impacted by each issue, which reports are duplicates (i.e., rediscoveries), which vulnerabilities were exploited, etc. Our cleaned datasets and our software implementation of the data collection and cleaning processes are publicly available [4].

Organization. The remainder of this paper is organized as follows. Section 2 provides an overview of our data collection and cleaning processes. Section 3 presents an in-depth analysis of the benefits of vulnerability discovery and bug-bounty programs. Section 4 discusses related work on vulnerability discovery and bug bounties. Finally, Section 5 presents concluding remarks.

2 DATA COLLECTION AND CLEANING

We collect reports of security issues (i.e., vulnerabilities) from the issue trackers of Chromium and Firefox. An *original report* of a vulnerability is a report that does not have *duplicate* in its *Status* field, which has typically—but not always—the earliest report date. A *duplicate report*, identified by *duplicate* in its *Status* field, is a report of an issue that had already been reported. If the duplicate and original reports were submitted by different bug hunters, then we consider the duplicate to be an independent *rediscovery*.

We describe the technical details of the data collection and cleaning processes in Appendices D and E.

---

[1]Note that some bug hunters could be malicious; in this paper, we define bug hunter as someone who reports a vulnerability, thereby helping the program. At the same time, they might also try to exploit the vulnerability, which could be reported as the vulnerability being exploited in the wild. Since we focus on the benefits of vulnerability discovery, we study both activities strictly from the programs' perspective.

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

2.1 Data Collection

We collect the following five attributes for each vulnerability: impacted release channels (stable and/or development), security severity (critical, high, moderate, or low), weakness type represented as broad type of Common Weakness Enumeration (CWE), affected components, and programming languages (i.e., languages of files that were modified to patch the issue). Note that for ease of exposition, we use the same names for severity levels and impacted releases for Chromium and Firefox; however, there is no one-to-one equivalence since there may be differences between the definitions of the two VRPs. For a duplicate report, we use the attributes of the original report as the attributes of the duplicate. If an original report is missing some attributes, we use the attributes of its duplicates.

Chromium: We collect the details of all vulnerability reports from September 2, 2008 – September 13, 2022 from the Chromium issue tracker[2] using Monorail API version 2[3]. For each report, the Chromium issue tracker lists affected components, impacted release channels, comments that include conversations among internal employees and external parties, as well as a history of changes (i.e., amendments) made to the report.

Firefox: We collect data from two main resources, the Bugzilla Firefox bug tracker[4] and the Mozilla website (Known Vulnerabilities[5] and Mozilla Foundation Security Advisories (MFSA)[6]). We collect security reports from January 24, 2012 – September 15, 2022. The MFSA lists vulnerabilities for Mozilla products. We scrape advisories for Firefox to be able to identify reports that pertain to stable releases. We also collect the *Reporter* field, which some pages in MFSA have, to identify external vs. internal reporters.

2.2 Data Cleaning

Rediscovery and Duplicate Reports. In both issue trackers, there is a *Status* field that indicates whether a report is a duplicate of a previously reported vulnerability or an original report. In the Chromium issue tracker, for rediscoveries the *Status* field is marked as *Duplicate*. For each duplicate report, we follow the *MergeInto* field to retrieve the referenced original report. If that is also a duplicate, we again follow the *MergeInto* field of the referenced report until we find the original. In the Firefox issue tracker, we can similarly determine whether a report is a duplicate based on the *Status* field, and we can find the original report by following the references (recursively, if needed). Some vulnerabilities are reported multiple times by the same hunter; we remove these redundant reports and keep only the earliest report of each vulnerability for each hunter. Some vulnerabilities do not have accessible pages in Bugzilla. Since we cannot identify the earliest report for these vulnerabilities, we excluded them from our rediscovery analysis. Some Firefox reports are incomplete with respect to replication and patching. For some of these, Mozilla opened a new report of the vulnerability, which was then completed with respect to this information, and the first report was marked as a duplicate. We also exclude these vulnerabilities from our analysis since they are not actual rediscoveries.

External vs. Internal Reports: Chromium. The Chromium issue tracker contains reports of vulnerabilities either reported internally by Google or reported externally by bug hunters. For each report, we use the reporter's email address to classify the report as either an *internal* or an *external report*. Note that we cannot always determine the report's origin based solely on the email address. For each such email address, we manually check the associated activities, such as vulnerabilities reported and comments posted to determine the reporter's origin. We are able to identify the email address of the actual external reporter for 98% of the valid external reports.

External vs. Internal Reports: Firefox. Vulnerabilities in Firefox VRP are reported either internally by Firefox team members or by external reporters. We use four steps to separate internal and external reports. First, we use the *Whiteboard* and *bug-flag* fields in the report information page. If a report has *reporter-external* in *Whiteboard* or *sec-bounty* in *bug-flag*, we generally consider the report to be external; otherwise, we consider it to be internal. However, there are reports, which do not have the above keywords, but were reported by external bug hunters. To identify those reports, in the next step, we leverage a snow-balling technique (on the report comments) to identify around 650 reports that appear to be from internal team members of Mozilla on the first glance, but their actual reporters are external. In the third step, we consider reporters that appear to have both internal and external reports (around 50). We manually disambiguate these cases by reading comments and checking their public websites. In the last step, we leverage the *Reporter* field in the MFSA by matching the reporters' profile names (from Bugzilla) with the names mentioned by the MFSA. By applying the above steps, we are able to distinguish internal and external reports in 97% of the cases.

Stable vs. Development Release Channels. Stable releases are widely used by end users, while development releases are typically used only by testers and bug hunters. We use the term *release channel* to refer to these different release versions. Note that we distinguish between reports that affect *stable* releases and reports that affect only *development* releases. In Chromium, there are reports that affect both stable and development releases, which we exclude from our analysis of stable vs. development. For Firefox, we consider Bugzilla reports that are listed in the MFSA to be reports that affect stable releases.

2.3 Other Data Sources

Most vulnerabilities that have been fixed have links to the Google or Mozilla source-code repositories in their comments, which we use to identify the files that were changed to fix the vulnerability. For each vulnerability with a repository link, we collect the programming languages of the files that were changed. We also leverage CVEDetails[7] and MITRE CWE[8] to collect information regarding CVE IDs and weakness types (CWEs), when available.

To identify exploited vulnerabilities, we first collect an initial set of exploited vulnerabilities from the Known Exploited Vulnerabilities Catalog of CISA[9]. Then, we extend this set iteratively using a

Soodeh Atefi, Amutheezan Sivagnanam, Afiya Ayman, Jens Grossklags, and Aron Laszka

Table 1: Summary of Datasets

Security Severity	Chromium	Firefox
Critical	309	1,420
High	8,616	2,332
Moderate	5,598	1,156
Low	2,720	605

Impacted Releases	Chromium	Firefox
Stable	8,152	3,002
Development	5,325	3,064

Reports	Chromium	Firefox
Duplicates	3,905	1,262
Originals	21,453	4,804

Reports' Origins	Chromium	Firefox
Externals	12,221	1,837
Internals	13,137	4,229

Table 2: Comparison of Stable and Development Releases

Impacted Releases	Chromium	Firefox
Number of Unique External Reporters		
Stable	1,297	413
Development	198	285
Ratio of Rediscovered Vulnerabilities		
Stable	8.63%	9.27%
Development	4.26%	12.37%
Mean Patching Time in Days		
Stable	80.73	73.62
Development	12.35	103.36

snowballing method by identifying terms in comments related to exploitation (e.g., *exploited in the wild*) and gathering vulnerabilities whose comments include these terms.

2.4 Summary of Datasets

For Chromium, we collected in total 25,358 valid reports of security issues. Of these, 12,221 were externally reported, and 13,137 were internally reported. Among reports with information about impacted releases (13,477 reports in total), 8,152 reports pertain to stable releases, and 5,325 pertain to development one. Finally, 21,453 are original reports, and 3,905 are duplicate reports. For Firefox, we collected in total 6,066 valid reports of security issues, of which 4,804 are original reports, and 1,262 are duplicates. There are 3,002 reports of vulnerabilities which pertain to stable releases, and 3,064 reports that pertain to development releases. 1,837 reports were reported externally, and 4,229 were reported internally. Table 1 shows summary statistics of the two datasets. Table 2 shows the number of unique external bug hunters (note that 86 external reports were submitted anonymously for Firefox). For year-over-year temporal analysis, we provide annual data in Appendix A. We observe that most key metrics of interest are stable (e.g., fraction of duplicate reports is around 18.2% for Chromium with 4.9% standard deviation annually), which suggest that the reward programs' evolution over the past years does not significantly impact our findings.

3 RESULTS

3.1 Benefits of Vulnerability Discovery and Bug Bounty Programs

3.1.1 Probability of Rediscovery (RQ 1.1). We begin our analysis by investigating whether vulnerabilities are more difficult to find in stable releases than in development ones. To quantify this, we first study the probability that a vulnerability is rediscovered. Table 2 shows for each release channel the ratio of vulnerabilities that are rediscovered at least once. In Firefox, *vulnerabilities that impact development releases are rediscovered more often than those that impact stable releases;* in Chromium, it is *vice versa*.
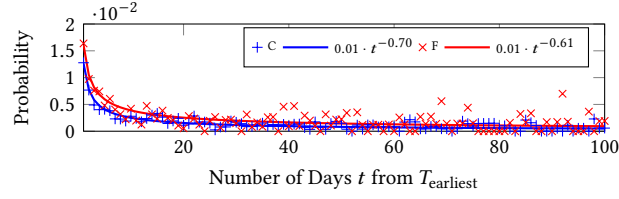
Before drawing any conclusions about the difficulty of finding vulnerabilities, we must also consider the number of unique external reporters working on stable and development releases (see Table 2). We find that in both Chromium and Firefox, *there are considerably more bug hunters who report vulnerabilities in stable releases than in development ones.* Combined with the rediscovery probabilities, this suggests that it is more difficult to find vulnerabilities in stable releases: although stable releases seem to attract significantly more attention, differences in rediscovery probabilities are less pronounced.

However, there is one more factor that can contextualize the difference in rediscovery probabilities: the amount of time required to patch a vulnerability. If it takes longer to patch a vulnerability, bug hunters have more time to rediscover it, which should lead to a higher rediscovery probability, ceteris paribus. Table 2 shows the average time between the first report of a vulnerability and the time when it was patched. We compute the time to patch Δ_{fix} as $\Delta_{\text{fix}} = T_{\text{fix}} - T_{\text{earliest}}$, where T_{fix} is the date and time when the vulnerability was fixed and T_{earliest} is when the vulnerability was first reported in the issue tracker. For Chromium, we observe that vulnerabilities in stable releases are patched much slower than in development releases, giving bug hunters significantly more time to rediscover them. For Firefox, the evidence is more nuanced. Here, we also observe a lower rediscovery probability for stable releases even though there is a larger workforce; however, hunters have to work with a slightly shorter average patching time window.

FINDING 1. *The rediscovery probabilities, number of bug hunters, and mean patching times in conjunction suggest that vulnerabilities are easier to find in development releases; vulnerabilities that are easy to find are likely to be discovered and patched during development, demonstrating the benefits of vulnerability discovery.*

3.1.2 Rediscovery Probability over Time (RQ 1.2). Since the probability of rediscovery alone cannot quantify the benefit of a vulnerability report, we contextualize the rediscovery probabilities of *stable*, *development*, and both releases together with the impact of patching. For a duplicate report, we define the *time until rediscovery* as $\Delta_{\text{rediscover}_d} = T_{\text{open}} - T_{\text{earliest}}$ (i.e., difference between the time of submitting the duplicate report and the submission time of the earliest report of the vulnerability). We estimate the probability $\Pr\left[Re(t) \mid t < \Delta_{\text{fix}}\right]$ that a vulnerability is rediscovered on the t-th day after it is first reported (this event is denoted $Re(t)$) given that the vulnerability has not been patched by day t as follows:

$$\left|\left\{o_d \mid d \in D, o_d \in O_{\text{fix}}, \Delta_{\text{rediscover}_d} = t\right\}\right| / |O_{\text{fix}}| \qquad (1)$$

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

(a) Probability that a vulnerability is rediscovered on the t-th day after it is first reported ($\Pr\left[Re(t)\,|\,t < \Delta_{\mathbf{fix}}\right]$).
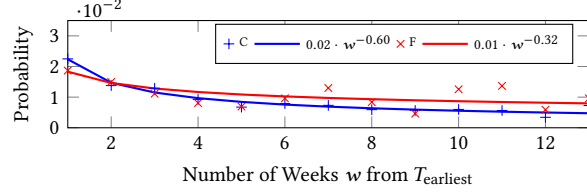
(b) Probability that a vulnerability in a stable release is rediscovered in the w-th week after it is first reported ($\Pr\left[Re(w)\,|\,w < \Delta_{\mathbf{fix}}\right]$).
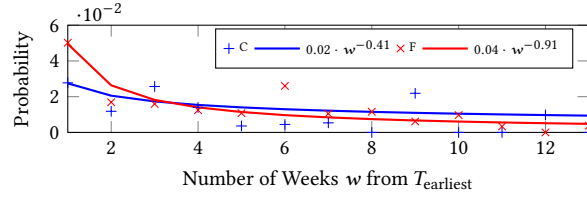
(c) Probability that a vulnerability in a development release is rediscovered in the w-th week after it is first reported ($\Pr\left[Re(w)\,|\,w < \Delta_{\mathbf{fix}}\right]$).

Figure 1: Probability that a vulnerability is rediscovered a certain time after its first report, given that it has not been patched by that time. F and C denote Firefox and Chromium.
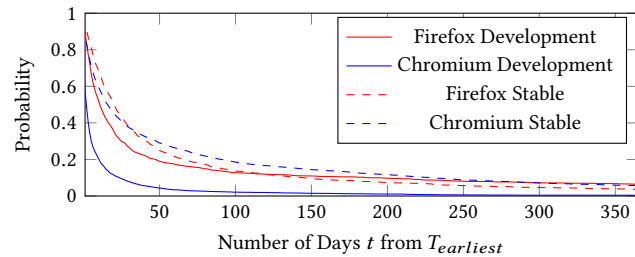

Figure 2: Probability that a vulnerability is not fixed in t days after it is first reported ($\Pr\left[\Delta_{fix} > t\right]$) in *development* (*solid lines*) and *stable* releases (*dashed lines*).

The nominator is the number of original reports (o_d) that have not been fixed by day t ($o_d \in O_{\text{fix}}$) and are rediscovered on the t-th day after they are first reported ($\Delta_{\text{rediscover}_d} = t$). The denominator is the number of original reports that have not been fixed by day t.

Similarly, we also estimate probability $\Pr\left[Re(w)\,|\,w < \Delta_{\text{fix}}\right]$ for the w-th week as follows:

$$\left|\left\{o_d \,|\, d \in D, o_d \in O_{\text{fix}}, 7w - 6 \leq \Delta_{\text{rediscover}_d} \leq 7w\right\}\right|/|O_{\text{fix}}| \quad (2)$$

where $O_{\text{fix}} \leftarrow \left\{o \in O \,|\, \Delta_{\text{fix}_o} > 7w\right\}$, i.e., not fixed by week w.
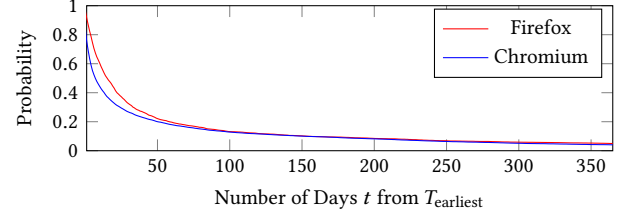

Figure 3: Probability that a vulnerability is not fixed in the t days after it is first reported ($\Pr\left[\Delta_{fix} > t\right]$).
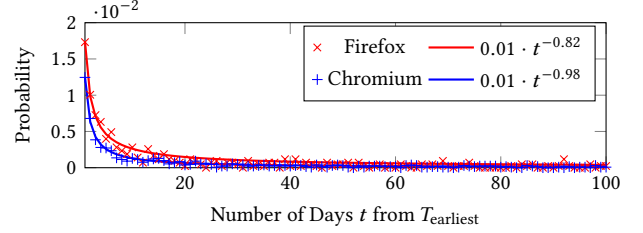

Figure 4: Probability that a vulnerability is rediscovered on the t-th day after it is first reported ($\Pr[Re(t)]$).

Fig. 1a shows that rediscovery probabilities decrease over time in both Chromium and Firefox. We fit curves to identify and visualize important trends; building principled models that fit these trends is a major task, which we leave for future work. When fitting curves, we weigh each probability value by its confidence, which we measure as the number of vulnerabilities based on which the value is estimated. We also computed the probability that a vulnerability is not patched in t days after it is first reported (see Fig. 3). This shows that 20% of vulnerabilities are patched within 5 days of first being reported and that most vulnerabilities are patched quickly.

Interestingly, even if we remove the condition ($t < \Delta_{\text{fix}}$) and consider the probability of rediscovery without the impact of patching, there is still a rapid decline in the first few days in both curves, i.e., Firefox and Chromium (see Fig. 4). On the other hand, both curves have long tails later, which suggests a somewhat memory-less process of discovery (i.e., some vulnerabilities that are not discovered soon may remain hidden for long).

Figs. 1b and 1c show the probability that a vulnerability is rediscovered in the w-th week after it is first reported (condition $w < \Delta_{\text{fix}}$) for vulnerabilities in stable and development releases, respectively. We observe that rediscovery probabilities are lower in stable releases than in development releases. In particular, this suggests that vulnerabilities in stable releases are more difficult to find and are non-trivial (see Fig. 1b). Further, the long tail suggests that vulnerabilities that have not yet been found may remain hidden for a long time, and discovery is mostly a memory-less process.

However, there is a small peak in both curves in the first few days, which contradicts the memory-less property of rediscovery in stable releases, and suggests a clustering of rediscoveries. One hypothesis is that vendors pay more to external bug hunters for the discovery of vulnerabilities in stable releases relative to development releases. As a result, bug hunters may stockpile vulnerabilities in development releases to receive higher rewards by reporting vulnerabilities in stable releases, which increases the likelihood
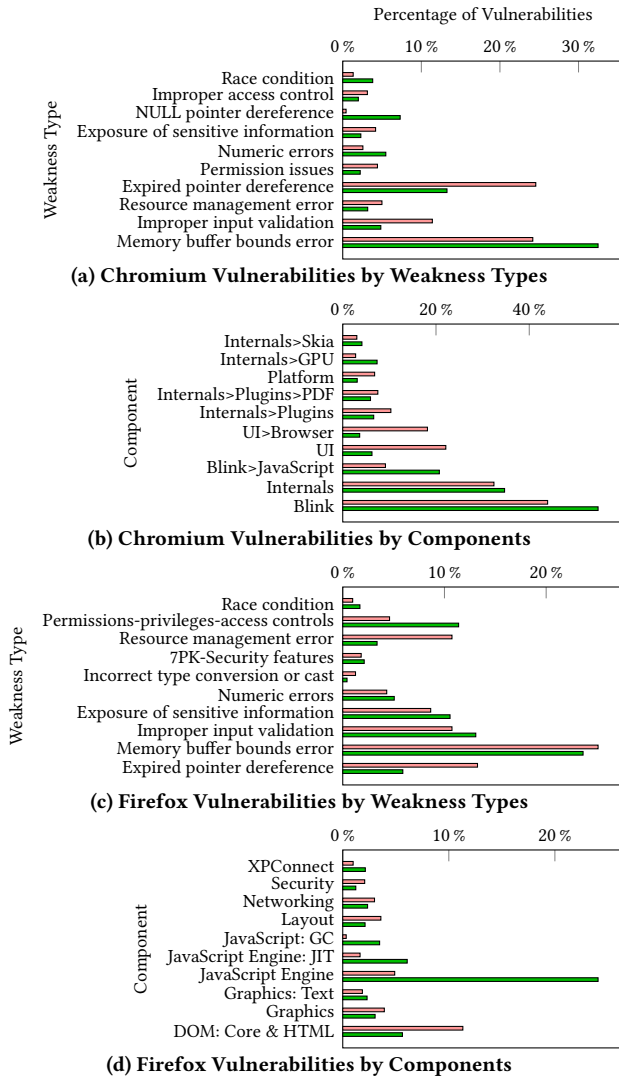
(a) Chromium Vulnerabilities by Weakness Types

(b) Chromium Vulnerabilities by Components

(c) Firefox Vulnerabilities by Weakness Types

(d) Firefox Vulnerabilities by Components

Figure 5: Comparison of internal (■) and external (■) security reports in Chromium and Firefox.

of duplicate reports. We tested this hypothesis by checking the reward policies of these two vendors. However, we did not find evidence for a higher reward policy for stable releases in either Chromium or Firefox. Instead, the reward policies are mostly based on vulnerability severity. Another hypothesis for the small peak in both curves (specifically for Chromium) is that vulnerabilities are more likely to be discovered shortly after they are introduced; exploring this hypothesis would require further technical analysis.

Fig. 1c shows rapid decline in the first few days. This suggests that most vulnerabilities in development releases are rediscovered in the first few days after they are introduced.

FINDING 2. *Vulnerability discoveries are clustered in time, which suggests that there is a limited pool of easy-and-quick-to-discover vulnerabilities. Other vulnerabilities may remain hidden for long.*

3.1.3 Internal Discoveries and External Bug Hunters (RQ2). Next, we study the differences between reports of different origins (i.e., external versus internal) for Chromium and Firefox. We provide a detailed comparison considering release channels in Appendix B.

Fig. 5 shows the distributions of internal vs. external reports with respect to weakness types and impacted components. As for weakness types (see Fig. 5a), the most common type among internal Chromium reports is *Memory buffer bound error* (32.5%), while the most common type among external reports is *Expired pointer dereference*. In contrast, *Memory buffer bound error* is the most common type among both internal and external Firefox reports (Fig. 5c). As for impacted components (see Fig. 5b and Fig. 5d), the *Blink* component was most common among both internal and external Chromium reports; while in Firefox, *DOM: Core & HTML* is the most common impacted component among external reports and *JavaScript Engine: JIT* is the most common among internal ones.

We also compared internal and external reports in terms of impacted release channels, severity, and programming languages. In Chromium, *stable releases* are impacted by a higher percentage of reports than other releases, for both internal (50.1%) and external (78.9%) reports. In Firefox, we observe that 57.8% of external reports pertain to *stable releases*, while 54.1% of internal reports relate to *development releases*. As for severity, a high percentage of both internal and external reports have a *high severity*. Further, external reports are more common than internal reports among vulnerabilities with *critical severity*. We also find that vulnerabilities in *C++* code are most frequently reported both internally and externally and in both Chromium and Firefox.

Based on Pearson's chi-squared test, external and internal reports follow significantly different distributions in terms of impacted release channels, severity level, weakness type, affected components, and programming languages for both Chromium and Firefox.

FINDING 3. *External bug hunters and internal security teams report different types of vulnerabilities, which indicates that bug-bounty programs do complement the expertise of internal teams.*

3.2 Management of Bug Bounty Programs

3.2.1 Vulnerabilities Reported and Exploited (RQ 3.1). Finally, we study how many vulnerabilities have been discovered and exploited by malicious actors and what the differences are between these exploited vulnerabilities and other vulnerabilities. Among the 25,358 (Chromium) and 6,066 (Firefox) valid vulnerability reports, we can identify 37 and 18 vulnerabilities that have been exploited in the wild for Chromium and Firefox, respectively. We compare these exploited vulnerabilities to those that are discovered by benevolent external reporters. We also compare these vulnerabilities with all other vulnerabilities (i.e., vulnerabilities that have not been exploited) based on release channels, severity, weakness type, components, and programming languages (see Appendix C). We perform chi-squared tests for all these comparisons as well. However, we acknowledge that the number of exploited vulnerabilities is limited; therefore, the results of our analysis might not be generalizable.

Comparison with Other Externally Reported Issues. Since our focus is on bug-bounty programs, we study the differences and similarities between vulnerabilities that are exploited by threat actors and vulnerabilities that are reported by external bug hunters (Fig. 6).

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

Percentage of Vulnerabilities

(a) Chromium Vulnerabilities by Weakness Types

(b) Chromium Vulnerabilities by Components

(c) Firefox Vulnerabilities by Weakness Types

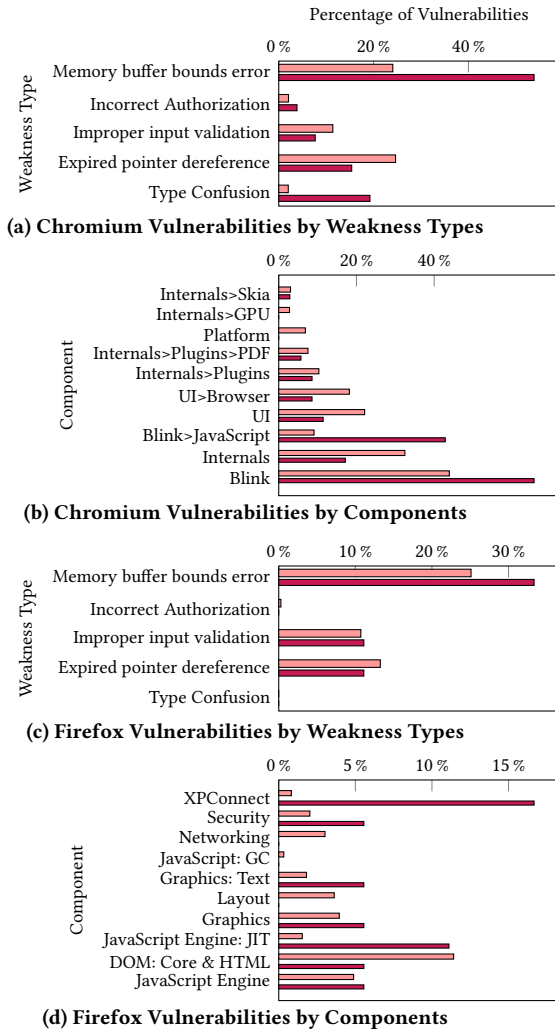**(d) Firefox Vulnerabilities by Components**

Figure 6: Comparison of exploited vulnerabilities (■) and external security reports (■) in Chromium and Firefox based on weakness types and impacted components.

As for release channels, all exploited Chromium vulnerabilities impact *stable releases*, while only 78.9% of external reports pertain to stable releases. In Firefox, 88.9% of exploited vulnerabilities impact *stable releases*, while only 57.5% of external reports pertain to stable releases. With respect to severity, 71.4% of exploited Chromium vulnerabilities are of *high severity*, whereas only 45.3% of external report have high severity. In Firefox, 62.5% of exploited vulnerabilities have *critical severity*, while only 25.8% of external reports have *critical* severity. Among both exploited vulnerabilities and external reports, vulnerabilities in *C++* code were the most common.

As for weakness types, *Memory buffer bound error* is the most commonly exploited type of vulnerability in both Chromium and Firefox (Figs. 6a and 6c). Fortunately, this weakness type is also very common among external reports in both Chromium and Firefox: it is the most common type in Firefox, and the second most common type in Chromium (just slightly behind the most common type,

Expired pointer dereference). With respect to impacted components, the *Blink* component of Chromium is the most common among both exploited vulnerabilities and external reports (see Fig. 6b). In Firefox, the *XPConnect* component is the most commonly impacted by exploited vulnerabilities; however, this component is relatively rare among external reports.

Our exploratory statistical tests show that for Chromium, exploited vulnerabilities and external reports follow significantly different distributions in terms of impacted release channels and security-severity levels; however, this does not hold for affected programming languages. Similarly, in Firefox, exploited vulnerabilities and external reports follow significantly different distributions in terms of impacted release channels and security severity.

FINDING 4. *There are significant differences between the types of vulnerabilities that are reported by bug hunters and those that are exploited by threat actors in terms of impacted release channels, and security-severity levels, which suggests that bug bounties could be more effective if they incentivized bug hunters to shift their focus.*

3.2.2 Difficulty of Discovery (RQ 3.2). We estimate the probability of rediscovery as a function of the inherent properties of a vulnerability (i.e., security severity, weakness type, impacted components, and programming languages) to study whether different types of vulnerabilities are more or less difficult to rediscover (see Fig. 7). As for security severity, vulnerabilities with *critical* and *high* severity in Firefox and vulnerabilities with *critical* severity in Chromium are rediscovered more than vulnerabilities with other severity levels. This can be partially explained by reward policies, which scale with the severity of the vulnerabilities. In Chromium, vulnerabilities with *low* severity are rediscovered more than vulnerabilities with *high* and *moderate* severity levels. In Firefox, vulnerabilities with *low* severity are rediscovered more than vulnerabilities with *moderate* severity level. This may imply that vulnerabilities with *low* severity are not only low-impact, but they are also shallow and easier to find. With respect to programming languages, vulnerabilities related to *CSS* files in Chromium and *Java* files in Firefox have higher probabilities of rediscovery compared to vulnerabilities related to files in other languages.

As for weakness types in Chromium, vulnerabilities of the type *Permission issues* are rediscovered more than vulnerabilities of other types (Fig. 7a). In Firefox, vulnerabilities of type *Incorrect type conversion or cast* are more likely to be rediscovered than vulnerabilities of other types (Fig. 7c). We also observe that vulnerabilities that impact the *UI>Browser* component in Chromium and the *Networking* component in Firefox are more likely to be rediscovered than vulnerabilities that impact other components (Figs. 7b and 7d). We also performed statistical tests between different types of vulnerabilities. The results show that there are significant differences between the rediscovery probabilities of different types of vulnerabilities.

FINDING 5. *There are significant differences between the rediscovery probabilities of different types of vulnerabilities. Since vulnerabilities that are more severe than others receive higher rewards, and they are also rediscovered more often than other vulnerabilities, vendors could include other properties of vulnerabilities in their reward policy to incentivize external bug hunters.*
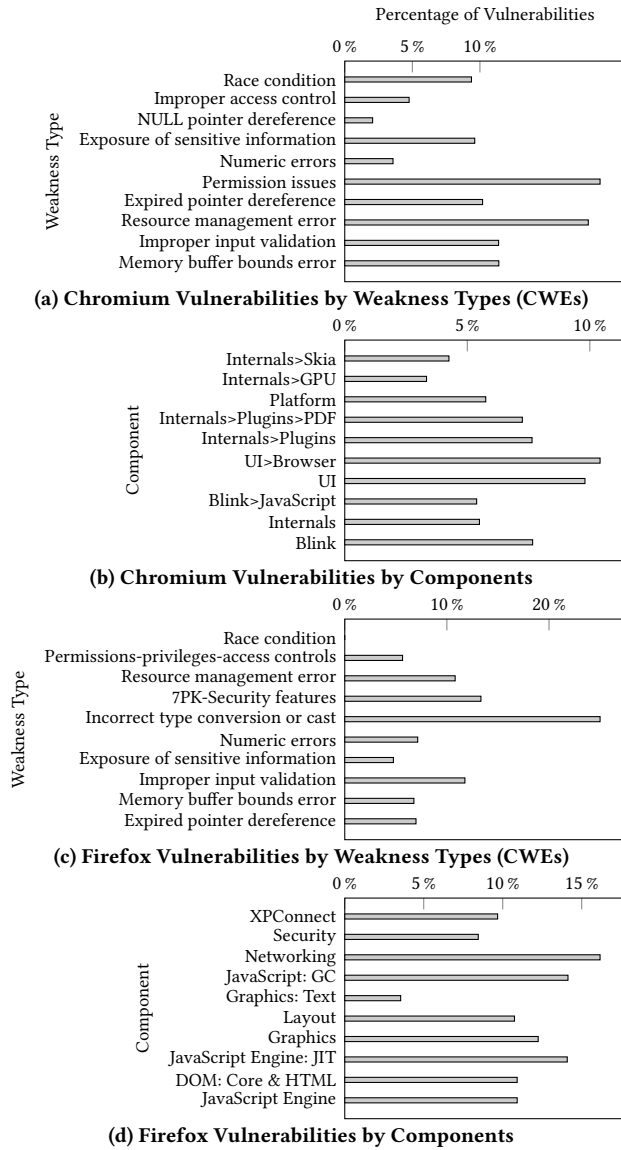
Percentage of Vulnerabilities

(a) Chromium Vulnerabilities by Weakness Types (CWEs)

(b) Chromium Vulnerabilities by Components

(c) Firefox Vulnerabilities by Weakness Types (CWEs)

(d) Firefox Vulnerabilities by Components

Figure 7: Fraction of vulnerabilities that are rediscovered at least once in Chromium and Firefox.

4 RELATED WORK

From a technical perspective, vulnerability discovery can be approached with a variety of static, dynamic, and concolic analysis methods as well as fuzzing [7, 10]. Taking an analytical and empirical perspective, Massacci and Nguyen [24] evaluated different mathematical vulnerability discovery models, which can be beneficial for vendors and users in terms of predicting vulnerability trends, adapting patching and update schedules, and allocating security investments. Prior works also investigated the empirical facets of vulnerability discovery in the context of bug bounty programs (e.g., [1, 8, 9, 21, 22, 34, 35]) and security bulletins (e.g., [11, 12]); however, research on rediscovery of vulnerabilities is sparse. Ozment [27]

provided data on rediscovery frequency based on Microsoft's vulnerability bulletins and concluded that rediscovery is not negligible and should be explicitly considered in discovery models. Finifter et al. [13] studied VRPs; in one part of their study, they estimated average rediscovery rates of 4.6% and similar rates for Chromium and Firefox, respectively. Both studies had to rely on very small datasets, but can serve as key motivators for our work. Herr et al. [16] estimated that vulnerability rediscovery occurs more often than previously reported (1% to 9%) in the literature (e.g., [26]) and discuss patterns of rediscovery over time. Our work relies on a considerably more sizable dataset, which allows us to consider inherent patterns of rediscovery such as impacted release channels or weakness types. As such, our work goes well beyond the mere estimation of rediscovery rates.

Complementary to our investigation of vulnerability discovery, Iannone et al. [17] study how, when, and under which circumstances vulnerabilities are introduced into software by developers and how they are removed. While Iannone et al. studied the life-cycle of vulnerabilities by analyzing source code, Alomar et al. [3] conducted 53 interviews with security practitioners in technical and managerial roles to study vulnerability discovery and management processes in the wild. In contrast, Akgul et al. [1], Votipka et al. [32], and Fulton et al. [14] conducted surveys and interviews with bug hunters. Alexopoulos et al. [2] also studied bug hunters, but instead of conducting interviews, they collected information about a large number of bug hunters from public sources.

5 CONCLUSION

Our analysis illustrates that it is more difficult to rediscover vulnerabilities in *stable* releases than in *development* releases, considering all aspects of the process, including the number of bug hunters and the time-to-patch. Further, vulnerability discoveries and rediscoveries tend to be clustered in time after the first discovery, but seem to exhibit a long tail afterwards. In addition, the rediscovery probabilities of different types of vulnerabilities vary considerably. Likewise, our analysis shows that external bug hunters and internal staff and tools report different types of vulnerabilities, indicating that bug-bounty programs leverage the diverse expertise of external hackers. Furthermore, we discuss initial evidence regarding the difference between vulnerabilities that are exploited by threat actors and those found by external bug hunters.

Suggestions for Improving Bug Bounties. Bug-bounty programs may benefit from incentivizing external hunters to focus more on development releases since the temporal clustering in stable releases suggest that some vulnerabilities that are relatively easy to find are not discovered during development. Similarly, programs may benefit from incentivizing hunters to focus more on the types of vulnerabilities that are likely to be exploited by threat actors. Our analysis offers another important facet for the management of bug-bounty programs. Conducting the work to identify a vulnerability and filing a comprehensive report is a time-consuming matter. However, duplicate reports are typically not rewarded. As such, our work may provide guidance regarding how to channel the attention of bug hunters to avoid collisions or which patch development or triage efforts to prioritize to avoid hacker frustration.

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

ACKNOWLEDGMENTS

REFERENCES

[1] Omer Akgul, Taha Eghtesad, Amit Elazari, Omprakash Gnawali, Jens Grossklags, Michelle Mazurek, Daniel Votipka, and Aron Laszka. 2023. Bug hunters' perspectives on the challenges and benefits of the bug bounty ecosystem. In *32nd USENIX Security Symposium (USENIX Security)*. https://doi.org/10.48550/arXiv.2301.04781

[2] Nikolaos Alexopoulos, Andrew Meneely, Dorian Arnouts, and Max Mühlhäuser. 2021. Who are vulnerability reporters? A large-scale empirical study on FLOSS. In *15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Article 25, 12 pages. https://doi.org/10.1145/3475716.3475783

[3] Noura Alomar, Primal Wijesekera, Edward Qiu, and Serge Egelman. 2020. "You've got your nice list of bugs, now what?" Vulnerability discovery and management processes in the wild. In *16th USENIX Conference on Usable Privacy and Security (SOUPS)*. 319–339. https://www.usenix.org/conference/soups2020/presentation/alomar

[4] Soodeh Atefi, Amutheezan Sivagnanam, Afiya Ayman, Jens Grossklags, and Aron Laszka. 2023. Dataset: The benefits of vulnerability discovery and bug bounty programs. (February 2023). https://doi.org/10.6084/m9.figshare.22056617

[5] Robert M. Brady, Ross J. Anderson, and Robin C. Ball. 1999. *Murphy's law, the fitness of evolving species, and the limits of software reliability*. Technical Report UCAM-CL-TR-471. University of Cambridge, Computer Laboratory. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-471.pdf

[6] Sandy Clark, Michael Collis, Matt Blaze, and Jonathan M. Smith. 2014. Moving targets: Security and rapid-release in Firefox. In *21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1256–1266. http://dx.doi.org/10.1145/2660267.2660320

[7] Lei Cui, Jiancong Cui, Zhiyu Hao, Lun Li, Zhenquan Ding, and Yongji Liu. 2022. An empirical study of vulnerability discovery methods over the past ten years. *Computers & Security* 120, Article 102817 (2022), 13 pages. https://doi.org/10.1016/j.cose.2022.102817

[8] Aaron Yi Ding, Gianluca Limon De Jesus, and Marijn Janssen. 2019. Ethical hacking for boosting IoT vulnerability management: A first look into bug bounty programs and responsible disclosure. In *8th International Conference on Telecommunications and Remote Sensing (ICTRS)*. 49–55. https://doi.org/10.1145/3357767.3357774

[9] Amit Elazari. 2019. Private ordering shaping cybersecurity policy: The case of bug bounties. In *Rewired: Cybersecurity Governance*, Ryan Ellis and Vivek Mohan (Eds.). Wiley. https://ssrn.com/abstract=3161758

[10] Sarah Elder, Nusrat Zahan, Rui Shu, Monica Metro, Valeri Kozarev, Tim Menzies, and Laurie Williams. 2022. Do I really need all this work to find vulnerabilities? An empirical case study comparing vulnerability detection techniques on a Java application. *Empirical Software Engineering* 27, 6, Article 154 (2022). https://doi.org/10.1007/s10664-022-10179-6

[11] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. 2019. Hey Google, what exactly do your security patches tell us? A large-scale empirical study on Android patched vulnerabilities. *2019 Workshop on the Economics of Information Security (WEIS)*, 24 pages. https://doi.org/10.48550/arXiv.1905.09352

[12] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. 2020. An empirical study of Android security bulletins in different vendors. In *The Web Conference 2020*. 3063–3069. https://doi.org/10.1145/3366423.3380078

[13] Matthew Finifter, Devdatta Akhawe, and David Wagner. 2013. An empirical study of vulnerability rewards programs. In *22nd USENIX Security Symposium (USENIX Security)*. 273–288. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/finifter

[14] Kelsey R. Fulton, Samantha Katcher, Kevin Song, Marshini Chetty, Michelle L. Mazurek, Daniel Votipka, and Chloé Messdaghi. 2022. Vulnerability discovery for all: Experiences of marginalization in vulnerability discovery. In *2023 IEEE Symposium on Security and Privacy (S&P)*. 289–306. https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00017

[15] Dan Geer. 2015. For good measure: The undiscovered. *;login:* 40, 2 (2015), 50–52. https://www.usenix.org/publications/login/apr15/geer

[16] Trey Herr, Bruce Schneier, and Christopher Morris. 2017. *Taking stock: Estimating vulnerability rediscovery*. White Paper. Belfer Cyber Security Project. https://doi.org/10.2139/ssrn.2928758

[17] Emanuele Iannone, Roberta Guadagni, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2022. The secret life of software vulnerabilities: A large-scale empirical study. *IEEE Transactions on Software Engineering* 49, 1 (2022), 44–63. https://doi.org/10.1109/TSE.2022.3140868

[18] Andreas Kuehn and Milton Mueller. 2014. Analyzing bug bounty programs: An institutional perspective on the economics of software vulnerabilities. In *42nd Research Conference on Communication, Information and Internet Policy (TPRC)*. https://doi.org/10.2139/ssrn.2418812

[19] Aron Laszka, Mingyi Zhao, and Jens Grossklags. 2016. Banishing misaligned incentives for validating reports in bug-bounty platforms. In *21st European Symposium on Research in Computer Security (ESORICS)*. 161–178. https://doi.org/10.1007/978-3-319-45741-3_9

[20] Aron Laszka, Mingyi Zhao, Akash Malbari, and Jens Grossklags. 2018. The rules of engagement for bug bounty programs. In *22nd International Conference on Financial Cryptography and Data Security (FC)*. Springer, 138–159. https://doi.org/10.1007/978-3-662-58387-6_8

[21] Donatello Luna, Luca Allodi, and Marco Cremonini. 2019. Productivity and patterns of activity in bug bounty programs: Analysis of HackerOne and Google vulnerability research. In *14th International Conference on Availability, Reliability and Security (ARES)*. Article 67, 10 pages. https://doi.org/10.1145/3339252.3341495

[22] Thomas Maillart, Mingyi Zhao, Jens Grossklags, and John Chuang. 2017. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity* 3, 2 (2017), 81–90. https://doi.org/10.1093/cybsec/tyx008

[23] Suresh S. Malladi and Hemang C. Subramanian. 2019. Bug bounty programs for cybersecurity: Practices, issues, and recommendations. *IEEE Software* 37, 1 (2019), 31–39. https://doi.org/10.1109/MS.2018.2880508

[24] Fabio Massacci and Viet Hung Nguyen. 2014. An empirical methodology to evaluate vulnerability discovery models. *IEEE Transactions on Software Engineering* 40, 12 (2014), 1147–1162. https://doi.org/10.1109/TSE.2014.2354037

[25] David McKinney. 2007. Vulnerability bazaar. *IEEE Security & Privacy* 5, 6 (2007), 69–73. https://doi.org/10.1109/MSP.2007.180

[26] Katie Moussouris and Michael Siegel. 2015. The wolves of Vuln Street: The 1st dynamic systems model of the 0day market. In *Retrieved from RSA Conference USA*. https://cams.mit.edu/wp-content/uploads/2017/12/The-Wolves-of-Vuln-Street-The-1st-System-Dynamics-Model-of-the-0day-Market.pdf

[27] Andy Ozment. 2005. The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting. In *4th Workshop on the Economics of Information Security (WEIS)*. http://infosecon.net/workshop/pdf/10.pdf

[28] Andy Ozment and Stuart Schechter. 2006. Milk or wine: Does software security improve with age?. In *15th USENIX Security Symposium (USENIX Security)*. 93–104. https://www.usenix.org/conference/15th-usenix-security-symposium/milk-or-wine-does-software-security-improve-age

[29] Eric Rescorla. 2005. Is finding security holes a good idea? *IEEE Security & Privacy* 3, 1 (2005), 14–19. https://doi.org/10.1109/MSP.2005.17

[30] Shanto Roy, Nazia Sharmin, Jaime C. Acosta, Christopher Kiekintveld, and Aron Laszka. 2023. Survey and taxonomy of adversarial reconnaissance techniques. *ACM Computing Surveys* 55, 6, Article 112 (2023), 38 pages. https://doi.org/10.1145/3538704

[31] Bruce Schneier. 2014. Should U.S. hackers fix cybersecurity holes or exploit them? The Atlantic, Available online at https://www.theatlantic.com/technology/archive/2014/05/should-hackers-fix-cybersecurity-holes-or-exploit-them/371197/.

[32] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. 2018. Hackers vs. testers: A comparison of software vulnerability discovery processes. In *39th IEEE Symposium on Security and Privacy (S&P)*. 374–391. https://doi.org/10.1109/SP.2018.00003

[33] Thomas Walshe and Andrew Simpson. 2020. An empirical study of bug bounty programs. In *2nd IEEE International Workshop on Intelligent Bug Fixing (IBF)*. IEEE, 35–44. https://doi.org/10.1109/IBF50092.2020.9034828

[34] Thomas Walshe and Andrew C. Simpson. 2022. Coordinated vulnerability disclosure programme effectiveness: Issues and recommendations. *Computers & Security*, Article 102936 (2022), 14 pages. https://doi.org/10.1016/j.cose.2022.102936

[35] Mingyi Zhao, Jens Grossklags, and Peng Liu. 2015. An empirical study of web vulnerability discovery ecosystems. In *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1105–1117. https://doi.org/10.1145/2810103.2813704

[36] Mingyi Zhao, Aron Laszka, and Jens Grossklags. 2017. Devising effective policies for bug-bounty platforms and security vulnerability discovery. *Journal of Information Policy* 7 (2017), 372–418. https://doi.org/10.5325/jinfopoli.7.2017.0372

Soodeh Atefi, Amutheezan Sivagnanam, Afiya Ayman, Jens Grossklags, and Aron Laszka

A ADDITIONAL DATA

Table 3: Number of Reports Per Year Based on Severity

Year Opened	Chromium				Firefox			
	Critical	High	Moderate	Low	Critical	High	Moderate	Low
2008				3				
2009	5	52	33	46				
2010	20	285	98	140				
2011	54	528	134	133				
2012	17	729	278	159	391	123	94	29
2013	20	544	242	154	331	215	92	56
2014	12	661	496	157	217	226	103	55
2015	10	675	282	174	170	263	124	63
2016	17	574	560	208	146	287	125	102
2017	25	855	726	325	106	427	164	82
2018	24	942	772	365	30	236	113	71
2019	45	859	807	355	17	235	112	45
2020	21	732	481	216	9	175	107	40
2021	31	900	547	207	0	128	84	39
2022	8	280	142	78	3	15	32	17

Table 4: Number of Reports Per Year Based on Releases

Year Opened	Chromium		Firefox	
	# of Stable Reports	# of Development Reports	# of Stable Reports	# of Development Reports
2008	1			
2009	95	1		
2010	361	0		
2011	524	27		
2012	716	103	302	396
2013	551	96	248	514
2014	776	338	270	413
2015	651	401	334	362
2016	637	538	376	331
2017	941	665	515	322
2018	875	794	281	217
2019	860	1066	233	209
2020	685	512	212	147
2021	435	624	177	117
2022	44	160	45	29

Table 5: Number of Original Vs. Duplicate Reports Per Year

Year Opened	Chromium		Firefox	
	# of Duplicate Reports	# of Original Reports	# of Duplicate Reports	# of Original Reports
2008		51		
2009	17	212		
2010	121	770		
2011	231	991		
2012	330	1278	157	541
2013	218	1253	154	608
2014	275	1509	148	535
2015	320	1243	147	549
2016	310	1642	153	554
2017	458	2156	241	596
2018	423	2292	92	406
2019	407	2742	82	360
2020	313	2064	43	316
2021	373	2422	26	268
2022	109	828	3	71

Table 6: Number of Reports Per Year Based on The Origins

Year Opened	Chromium		Firefox	
	# of Internal Reports	# of External Reports	# of Internal Reports	# of External Reports
2008	23	28		
2009	146	83		
2010	455	436		
2011	632	590		
2012	867	741	502	196
2013	833	638	549	213
2014	1023	761	510	173
2015	762	801	460	236
2016	915	1037	461	246
2017	1590	1024	555	282
2018	1551	1164	329	169
2019	1908	1241	311	131
2020	1039	1338	275	84
2021	1130	1665	206	88
2022	263	674	57	17

Tables 3 to 6 show annual data.

Table 7 shows how average patching time varies with severity, weakness type, components, and programming languages.

Table 7: Mean Patching Time in Days

Chromium Security Severity	Days	Firefox Security Severity	Days
Critical	28.46	Critical	23.56
High	38.25	High	55.26
Moderate	47.14	Moderate	133.24
Low	114.09	Low	183.38
Chromium Weakness Types	**Days**	**Firefox Weakness Types**	**Days**
Race condition	53.27	Race condition	65.90
Expired pointer dereference	30.17	Expired pointer dereference	39.78
Memory buffer bounds error	49.99	Memory buffer bounds error	42.39
Improper input validation	74.31	Improper input validation	100.70
Exposure of sensitive information	79.57	Exposure of sensitive information	107.61
Numeric errors	49.44	Numeric errors	25.45
Permission issues	102.96	Incorrect type conversion or cast	24.75
Null pointer dereference	94.66	7PKSecurity features	143.80
Improper access control	60.67	Permissions-privileges-access controls	91.82
Resource management error	28.64	Resource management error	49.55
Chromium Component	**Days**	**Firefox Component**	**Days**
Internals>Skia	29.32	XPConnect	120.77
Internals>GPU	25.48	Security	164.56
Platform	90.85	Networking	72.86
Internals>Plugins>PDF	46.11	Layout	119.37
Internals>Plugins	56.29	JavaScript: GC	48.91
UI>Browser	88.26	JavaScript Engine: JIT	35.70
UI	82.32	JavaScript Engine	52.88
Blink>JavaScript	13.27	Graphics: Text	56.55
Internals	48.20	Graphics	80.97
Blink	51.58	DOM: Core & HTML	51.11
Chromium Language	**Days**	**Firefox Language**	**Days**
C++	39.18	C++	51.21
JS	34.28	JS	66.47
HTML	65.87	HTML	81.90
C	31.25	C	52.99
XML	114.00	XML	110.81
Python	73.71	Python	87.57
Java	76.11	Java	189.70
CSS	69.44	CSS	330.42

Table 8: Chi-Squared Test Results

Chromium Internal vs. External Reports	p-Value	Firefox Internal vs. External Reports	p-Value
Impacted Releases	< .001	Impacted Releases	< .001
Security-Severity	< .001	Security-Severity	< .001
Component	0.0	Component	< .001
Weakness Types	< .001	Weakness Types	< .001
Language	< .001	Language	< .001
Chromium Internal vs. External Reports (Only Stable)	**p-Value**	**Firefox Internal vs. External Reports (Only Stable)**	**p-Value**
Security-Severity	< .001	Security-Severity	< .001
Component	0.0	Component	< .001
Weakness Types	< .001	Weakness Types	< .001
Language	< .001	Language	< .001
Chromium Rediscoveries	**p-Value**	**Firefox Rediscoveries**	**p-Value**
Impacted Releases	< .001	Impacted Releases	0.006
Security-Severity	< .001	Security-Severity	< .001
Component	0.0	Component	0.0
Weakness Types	0.0	Weakness Types	< .001
Language	0.0	Language	0.0
Chromium Exploited vs. All Other Vulnerabilities	**p-Value**	**Firefox Exploited vs. All Other Vulnerabilities**	**p-Value**
Impacted Releases	< .001	Impacted Releases	0.001
Security-Severity	0.06	Security-Severity	0.006
Component	0.04	Component	< .001
Weakness Types	0.87	Weakness Types	0.99
Language	0.45	Language	0.97
Chromium Exploited vs. All External Vulnerabilities	**p-Value**	**Firefox Exploited vs. All External Vulnerabilities**	**p-Value**
Impacted Releases	0.01	Impacted Releases	0.01
Security-Severity	0.01	Security-Severity	0.003
Component		Component	
Weakness Types		Weakness Types	
Language	0.21	Language	

Table 8 shows the results of chi-squared tests between different types of vulnerabilities. For some variables, we could not apply tests due to the 0 values (empty cells).

B INTERNAL AND EXTERNAL REPORTS IMPACTING STABLE RELEASES

Fig. 8 shows the distribution of weakness types and impacted components for internal and external reports in stable releases. As for weakness types, we find that reports related to *Memory buffer bounds error* are the most common among both origins, in both Chromium and Firefox (Figs. 8a and 8c). In Chromium, the *Blink*
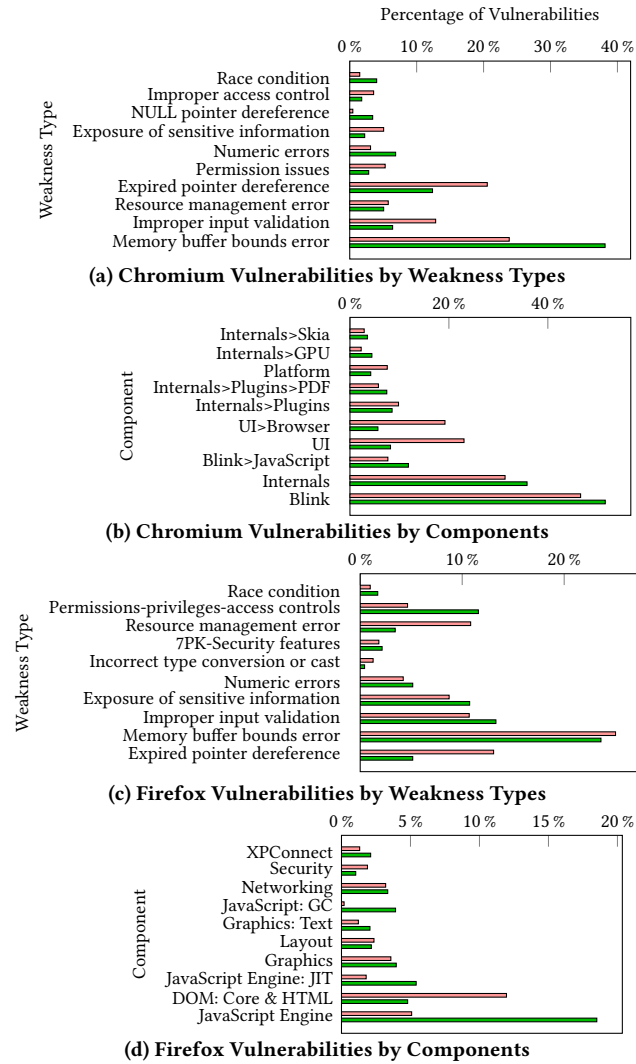
Percentage of Vulnerabilities

(a) Chromium Vulnerabilities by Weakness Types

(b) Chromium Vulnerabilities by Components

(c) Firefox Vulnerabilities by Weakness Types

(d) Firefox Vulnerabilities by Components

Figure 8: Comparison of internal (■) and external (■) security reports in *stable* releases of Chromium and Firefox.

Percentage of Vulnerabilities

(a) Chromium Vulnerabilities by Weakness Types

(b) Chromium Vulnerabilities by Components

(c) Firefox Vulnerabilities by Weakness Types

(d) Firefox Vulnerabilities by Components

Figure 9: Comparison of exploited vulnerabilities (■) and all other vulnerabilities (■) in Chromium and Firefox.

component is most commonly impacted by both internal and external reports (Fig. 8b). In Firefox, *Javascript Engine* is most common among internal reports, while *Dom: Core & HTML* is most common among external ones (Fig. 8d). We also compared internal and external reports in terms of severity and programming languages. Most internal and external reports have *high severity* in both Chromium and Firefox. External reports are more common than internal ones among vulnerabilities with *critical severity* in both software products. As for programming languages, we find that most reports are related to *C++*, regardless of origins, for both Chromium and Firefox. Pearson's chi-squared test shows that external and internal reports (that impact stable releases) follow significantly different distributions in terms of severity, weakness type, impacted components, and programming languages in both Chromium and Firefox.
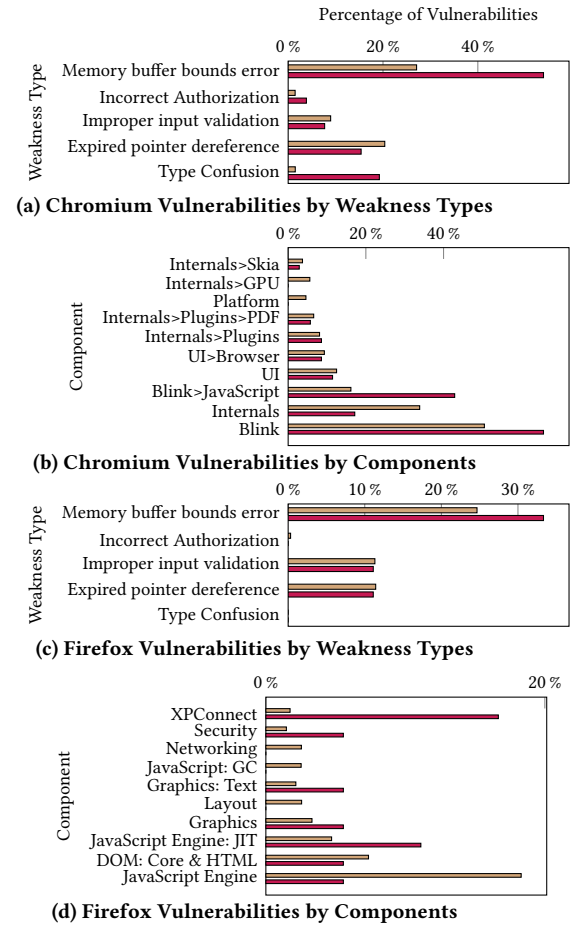
C   COMPARISON OF EXPLOITED VULNERABILITIES

We compare vulnerabilities that are exploited in the wild with all other vulnerabilities (i.e., vulnerabilities that have not been exploited). Fig. 9 shows the distributions of weakness types and components for exploited vulnerabilities and all other vulnerabilities for both Chromium and Firefox. Regarding release channel, we observe that 100% of exploited Chromium vulnerabilities and 88.88% of exploited Firefox vulnerabilities impacted *stable* release channel whereas 49.37% and 60.40% of all other vulnerabilities that have not been exploited pertain to *stable* releases in Firefox and Chromium respectively. As for security-severity 71.42% of exploited Chromium vulnerabilities were labeled as *high* severity whereas 49.92% of all other vulnerabilities have *high* severity. In Firefox 62.50% of exploited vulnerabilities have *critical* severity while 25.65% of all other vulnerabilities have *critical* severity assigned to them. Among programming languages, *C++* has the highest exploited reports among other languages in both datasets. This also holds for all other vulnerabilities that have not been exploited.

Comparison of exploited vulnerabilities and other vulnerabilities in terms of weakness type shows that vulnerabilities with *Memory buffer bound error* have a higher percentage in both exploited vulnerabilities and other vulnerabilities in both Chromium and Firefox datasets (see Fig. 9a and Fig. 9c). For component attribute in Chromium (see Fig. 9b), vulnerabilities with *Blink* component have a higher percentage in both exploited vulnerabilities and other vulnerabilities. In Firefox, vulnerabilities with *XPConnect* component have been exploited more than vulnerabilities with other components. For all other vulnerabilities, vulnerabilities with *JavaScript Engine* component has a higher percentage in comparison with others (see Fig. 9d). The results of Pearson's chi-squared test show that exploited vulnerabilities and all other reported vulnerabilities follow significantly different distributions in terms of impacted release channels and components in Chromium. Chi-squared tests for Firefox show that exploited vulnerabilities and all other reported vulnerabilities follow significantly different distributions in terms of impacted release channels, security severity, and impacted components (weakness types and languages accepted the null).

D DATA COLLECTION

In this section, we describe the key steps of our data collection process for both Chromium and Firefox.

D.1 Chromium Issue Tracker

We collected all data from September 2, 2008 to September 13, 2022 from the Chromium issue tracker[10] using Monorail API version 2[11]. We use three types of requests from the Monorail API for data collection: (i) *ListIssues*, which returns the list of reports that satisfy the query specified in the request; (ii) *GetIssue*, which returns the details of the report corresponding to the report identification number specified in the request; and (iii) *ListComments*, which returns the list of comments posted on the report specified in the request. For each vulnerability's report, the Chromium issue tracker stores a list of comments, which includes conversations among internal employees and external parties as well as a history of changes (i.e., amendments) made to the report.

Each report contains the following fields:

- *IdentificationNumber*: A unique number that identifies the report.
- *Status*: Current state of the report (*Unconfirmed*, *Untriaged*, *Invalid*, *Duplicate*, *Available*, *Assigned*, *Started*, *ExternalDependency*, *Fixed*, *Verified*, and *Archived*).
- *Component*: Component (or components) of the Chromium project that are affected by the report.
- *Owner*: Email address of the person who currently owns the report (e.g., reporter of the vulnerability or the person who fixes or closes the vulnerability).
- *AllLabels*: Labels associated with the report. These labels are used to categorize reports, e.g., to indicate security-severity levels (*critical*, *high*, *medium*, or *low*), impacted versions (*stable*, *beta*, or *head*), reward amount for bug bounty (e.g.,

Reward-500 indicates that $500 is awarded to the reporter of the vulnerability), or CVE ID.
- *Summary*: Short description of the report.
- *ReporterEmail*: Email address of the person who reported the report.
- *CCDetails*: Email addresses of all users who are part of the conversation thread of the report.
- *OpenedTimestamp*: Date and time when the report was initially reported.
- *ClosedTimestamp*: Date and time when the report was closed.
- *BugType*: Type of the report (e.g., Bug-Security).
- *MergedInto*: This is an optional field that applies only to duplicate reports. This field references the original report.

Each comment posted on a report consists of the following fields:

- *CommenterEmail*: Email address of the person who posted the comment.
- *Content*: Text of the comment, images of the report, videos of how to reproduce the report, etc.
- *SequenceNumber*: Order of the comment among all comments posted on the report.
- *CommentedTimestamp*: Date and time when the comment was posted on the report.
- *Amendments*: Updating or removing the values of some fields of the report (e.g., changing the owner, status, or priority).

Each amendment added to a comment consists of the following fields:

- *FieldName*: Name of the field that the amendment changes.
- *OldValue*: List of previous values of the field. This an optional field.
- *NewOrDeltaValue*: List of new and removed values of the field.
- *AmendmentTimestamp*: Date and time when the amendment was posted.

Chrome Releases. We also collected all the release notes from the Chrome Releases blog[12], which provides information regarding both the closed-source Chrome and the open-source Chromium projects. A release note is a blog post written by Google when they officially release a new version of Chrome or Chromium. Each Chromium release note contains a list of vulnerabilities that Google patches in the Chromium project when releasing the corresponding version. Each entry in this list of vulnerabilities contains the following fields:

- *IdentificationNumber*: Unique identification number of a report. We use this to join the Chromium issue tracker data with the Chrome Releases dataset.
- *ReporterName*: List of bug hunters who reported the particular vulnerability.
- *Association*: Organization (or organizations) where the reporters work.
- *ReleaseDate*: Release date of Chromium version that includes the fix for the vulnerability.

---

[10]https://bugs.chromium.org/p/chromium/issues/
[11]https://chromium.googlesource.com/infra/infra/+/master/appengine/monorail/api/README.md

[12]https://chromereleases.googleblog.com/

Google Git For Programming Languages Analysis. Another data resource that we use in our study is the Google Git repository[13]. From analyzing comments on vulnerabilities that we collected from the Chromium issue tracker, we find that most vulnerabilities that have been fixed have links to the Google Git repository, which we can use to identify the files that were changed to fix the vulnerability. For each vulnerability with a Google Git repository link, we collected the programming languages of the files that were changed.

D.2 Mozilla Firefox VRP

We collected data from two main sources, Bugzilla [14] (Firefox bug tracker) and Known Vulnerabilities from the Mozilla website [15], which is a list of security advisories based on product and advisories for older products that all are listed in the Mozilla Foundation Security Advisories (MFSA) [16]. We collected all security issues from January 24, 2012 to August 25, 2022.

To collect security vulnerabilities, we used security keywords added to the URL of Bugzilla search and scraped all URLs of reports that had at least one of the security keywords in the report *Keywords* field. Finally, all of the information related to a report was scraped. Each report contains several fields, of which the collected ones are listed below:

- *BugID*: A unique identifier of the report.
- *CVE*: CVE Id of the report (does not exist for all of the reports).
- *Opened*: Date and time when the report is opened.
- *Closed*: Date and time when the report is closed.
- *Summary*: A brief summary of the report.
- *Product*: Product type which the report is related to (we are interested in the Core and Firefox).
- *Component*: Component (or components) of the Firefox that are affected by the vulnerability.
- *Type*: This field represents type of the bug. It contains three types: defect, enhancement, and task. We are only interested in the defect type.
- *Status*: This represents current status of the report and what has happened to the report. It contains UNCONFIRMED, NEW, ASSIGNED, REOPENED for reports that are open. For reports that are closed, *Status* field contain, RESOLVED and VERIFIED which each have 7 resolutions (resolution represents the approach applied to reach to the current status): FIXED, INVALID, WONTFIX, MOVED, DUPLICATE, WORKSFORME, and INCOMPLETE.
- *Reporter* Username of a person who reported the issue.
- *Keywords* Criticality of the report (critical, high, medium, low) is mentioned in this field.
- *Duplicates* ID number of duplicates of the report.
- *Whiteboard* It contains tags, or information of a report's status. *reporter-external* tag is one of the tags which is used to identify external versus internal reporter.
- *Bug Flags* It contains *sec-bounty* value and it does not exist for all of the reports.

- *TrackingFlagsStatus* It contains vulnerability's statuses tracked by developers (does not exist for all of the reports).
- *Comments* All comments posted on the report.

Fixed Timestamp. Each report that has *VERIFIED* or *RESOLVED* followed by *FIXED* in its *Status* field, has an arrow followed by *RESOLVED* (→ RESOLVED) and an arrow followed by *FIXED* (→ FIXED) in one of its last comments (date of that comment which equals to the close time of the report). We use that date (close time) as the date the vulnerability is fixed. There are some reports that are closed because of incomplete fix status and reopened again. In those cases, we consider the last fixed time as the fix time of that vulnerability.

Mozilla Foundation Security Advisories (MFSA). MFSA reports vulnerabilities for Mozilla products. In this paper, we focus on Firefox. To identify reports pertaining to *stable* releases, we use MFSA. For FireFox and its older versions, we scraped advisories to be able to label reports that pertain to stable releases. We also collected the *Reporter* field, which some pages in MFSA have, to identify external versus internal reporters in our cleaning process.

Firefox Modified Source Files For Programming Languages Analysis. Most vulnerabilities that have been fixed have links to the Mozilla source-code repositories in their comments, which we use to identify the files that were changed to fix the vulnerability. For each vulnerability with a repository link, we collect the programming languages of the files that were changed.

Reopened. In Firefox, some reports had incomplete status due to the lack of information for replication and patching. For some of them, Mozilla reopened a new report of the vulnerability, which was then completed with respect to this information, and marked the first report as a duplicate. These reports can be identified by searching a right arrow to *REOPENED* (→ REOPENED) in the comments of that reports. Later in our analysis, we exclude these reports from rediscovery analysis since they are not actual rediscoveries.

D.3 CVEs and CWEs

CVEDetails. We leverage CVEDetails[17] and MITRE CWE[18] to collect information regarding CVE IDs and weakness types (CWEs), for both Chromium and Firefox when available. One of the fields associated with a report is *AllLabels* in Chromium. These labels may include a categorical parameter *Common Vulnerabilities and Exposures (CVE) Entry*, which contains an identification number called *CVE ID*. In Firefox, some reports contain *CVE ID* field which we collected them for analysis. These identifiers are used by cyber-security product and service vendors and researchers as one of the standard methods for identifying publicly known vulnerabilities and for cross-linking with other repositories that also use CVE IDs. Using these CVE IDs, we collected CVSS scores, impact metrics, and weakness types from CVEDetails[19]. For each report with a CVE ID, we collected the following details:

- *CVSS Score*: *Common Vulnerability Scoring System* (CVSS) provides a way of capturing the fundamental characteristics

---

[13]https://chromium.googlesource.com/
[14]https://bugzilla.mozilla.org/home
[15]https://www.mozilla.org/en-US/security/known-vulnerabilities/
[16]https://www.mozilla.org/en-US/security/advisories/

[17]https://www.cvedetails.com/
[18]https://cwe.mitre.org/
[19]https://www.cvedetails.com/

of a vulnerability. This numerical score reflects the severity of the vulnerability.

- *Confidentiality Impact*: Impact of successful exploitation on information access and disclosure.
- *Integrity Impact*: Impact of successful exploitation on the trustworthiness and veracity of information.
- *Availability Impact*: Impact of successful exploitation on the accessibility of information resources.
- *Access Complexity*: Complexity of the attack required to exploit the vulnerability once an attacker has gained access to the system.
- *CWE ID*: *Common Weakness Enumeration* (CWE) is a community-developed list of weakness types. The CWE ID references the type of software weakness associated with the particular vulnerability.

Weakness Types (CWE IDs). The CWE IDs associated with the vulnerabilities represent common types of software weaknesses. We later use CWE ID collected from *cvedetails* to collect broad-type names of CWEs from MITRE for weakness type analyses. Some of these weakness types have a hierarchical relationship with other types. For example, CWE 119 denotes the error "Improper Restriction of Operations within the Bounds of a Memory Buffer." This weakness type is also the parent of other CWEs, including CWE 120 (Classic Buffer Overflow), CWE 125 (Out-of-bounds Read), and CWE 787 (Out-of-bounds Write). For ease of presentation, we group the CWE weakness types together based on their parent-children hierarchy.

E   DATA CLEANING

In this section, we describe the key steps of our data cleaning process for both Chromium and Firefox.

In our analysis, we only consider reports that satisfy at least one of the following three conditions: (1) the report is an original report, and it has at least one security label; (2) the report is an original report, and the value of field BugType is *Bug-Security* for Chromium or has one of security labels in *Keywords* field for Firefox; and (3) the report is a duplicate report, and its original report satisfies at least one of the above conditions.

E.1   Duplicate Reports

E.1.1   Chromium. A report in the issue tracker is considered to be a *duplicate* if the underlying report has already been reported to the Chromium issue tracker (i.e., if this is a rediscovery). We can determine whether a report is a duplicate or not based on the *Status* field of the report: if the *Status* field is marked as *Duplicate*, the report is a duplicate.

To facilitate studying vulnerability rediscovery, we find the original report of each duplicate report as follows. For each duplicate report D, we follow the *MergeInto* field to retrieve the report referenced by it. If that is a duplicate report, we again follow the *MergeInto* field of the referenced report. We continue this process recursively until either one of the following holds:

- We reach a report O that is not a duplicate report. In this case, report O is the *original report* of duplicate report D.
- We reach a report X that is a duplicate report but does not have any references in the *MergedInto* field (or the value of

MergedInto field is malformed). In this case, we say that the duplicate report D does not have an original report.

We include a duplicate report D in our rediscovery analysis if report D has an original report O and report O has at least one security-related label. In order to retrieve duplicates of a vulnerability, we use *Duplicates* field which has references to the duplicate reports. For the cases that references also have a reference to other duplicates, we recursively retrieve duplicate reports until there is not any reference to a duplicate report.

E.1.2   Firefox. We can determine whether a vulnerability was reported earlier (i.e., is a duplicate) or not based on the *Status* field. If the report is a duplicate, *Status* field contains the keyword *Duplicate* and it has reference to the original report. In some cases, the referenced report, which is supposed to be the original report, has *Duplicate* in the status and has reference to another report. In these cases, we recursively, retrieve the report which is referenced in the *Status* field until there is not any reference to a report.

E.2   Valid and Invalid Reports

For both Chromium and Firefox, if the *Status* field of an original report is not marked as *Invalid*, it is considered a *valid original report*. If a duplicate report has a valid original report, then the duplicate report is a *valid duplicate report*. If a vulnerability belongs to either valid original reports or valid duplicate reports, then the vulnerability is considered a *valid vulnerability*.

If the *Status* field of an original report is marked as *Invalid*, it is considered an *invalid original report*. If a duplicate report has an invalid original report, then the duplicate report is an *invalid duplicate report*. If a report belongs either to invalid original reports or invalid duplicate reports, then, the report is considered an *invalid report*.

In Firefox, there are other invalid statuses that we do not consider them as valid statuses for reports. We do not consider duplicate reports that their original report has *INACTIVE*, *INCOMPLETE*, *MOVED*, *WONTFIX*, *WORKSFORME*, or *UNCONFIRMED* in its' *Status* field. However, there is an exception here. By checking some of the reports with the mentioned statuses, we realized that some reports that have *WORKSFORME* or *INCOMPLETE* in their *Status* field, have *fixed* word in their *TrackingFlags* field. Therefore, we keep duplicate reports that their original has *WORKSFORME* or *INCOMPLETE* in its' status and it got fixed in a version (according to the 'fixed' word in the *TrackingFlags* field).

E.3   Type and Product

In Bugzilla (Firefox), there is a *Type* field that contains type of a vulnerability which can be task, enhancement, or defect. We only keep duplicate reports that their original report have *defect* type. As for *Product* field, we keep only duplicate reports that their original report's product contains *Core* or *Firefox*.

E.4   External and Internal Reports

E.4.1   Chromium. The Chromium issue tracker contains reports either reported internally by Google or reported externally by bug hunters. For each report, we use the reporter's email address to classify the report as either an *internal* or an *external report*. However,

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

not all email addresses fall into the internal vs. external classification; thus, we cannot always determine the reported origin based on the email address alone. For each such address, we manually check the activities of the email address, such as vulnerabilities reported and comments posted by this particular email address. We determine the reporter's origin based on the activities associated with a particular email address.

There are also cases where the email address could be misleading. First, some external bug hunters submit reports privately to Google, and internal experts then post these reports on the Chromium issue tracker. Second, sometimes internal reporters import reports from other bug-bounty programs (e.g., Firefox, Facebook). In these cases, we need to identify the actual external reporter for each replicated report by analyzing the CC email address list of the report. We further improve the data cleaning process of distinguishing internal and external reports using the data we collected from Chrome Releases. The detailed cleaning process can be described using the following steps.

Step 1: Initial Classification based on ReporterEmail Field

For each report I, we use the email address of the reporter to classify the report I as either an *internal* or an *external* report. Specifically, if the email address is *ClusterFuzz* or ends with *google.com*, *chromium.org*, or *gserviceaccount.com*, and does not contain any label stating *external_security_report* or label starting *reward-to-external* and contains a non google email (i.e., email address without google.com or chromium.org) then we consider report I to be internally reported; otherwise, we consider it to be externally reported.

Step 2: Identifying Outlier Email Addresses based on Comments

We found that some of the reporters have email addresses that do not fit the rules of Step 1. One exception is the gmail address *scarybeast*, which belongs to an internal reporter. We identified this exception by analyzing the comments posted on reports reported by this email address. Based on the comments, we determined that this reporter is one of the key persons in announcing the confirmation of the reward to external reporters. Thus, we consider reports reported by this email address as internal reports. [20]

We also find some other exceptions where the email address of the reporter *skylined* or *cnardi* end with chromium.org. When we analyze the comments on reports reported by *skylined* with chromium.org address, we realized he served as a team member of the Google Chrome Security Team from 2008 - 2013 and left Google. After leaving Google, he reported few vulnerabilities as an external reporter and received rewards. When we analyze the comments on reports reported by *cnardi* ends with chromium.org, one comment mentions "cnardi@chromium.org as an external reporter regardless of his email address ends with @chromium.org." Accordingly, we classify them as an external reporter. We consider the reports reported by these two reporters as external reports.

Step 3: Analyzing CCDetails Field and Identifying the Actual External Reporters

Some reports that are submitted by internal-reporters are replications of reports privately submitted by external reporters to Google or reports imported from other bug bounty programs (e.g., Firefox,

---

[20]We believe this person was actually a member of the Google Chrome Security Team.

Facebook). Google replicates most of these externally reported reports through the automated tool ClusterFuzz, but sometimes Google replicates them manually using internal reporters (e.g., scarybeasts@gmail.com). For each replicated report, we need to identify the actual external reporter. We use the following approach and identify the email address of the external reporter of those reports.

For each report I for which we have to identify the email address of the actual external reporter, we first extract the CC email addresses (CC_{all}) from the *CCDetails* field. From CC_{all}, we obtain a new list CC_{remain} by removing the email address where the email address belongs to an internal reporter at the end of Step 2. For each email address in the CC_{remain} list, we look into comments of the corresponding report I whether any comment has one of the following phrases "originally reported by", "thanks to", "credits to", "thanks", "credits", "reward", "congratulations" immediately followed by username or email address or full name of the reporter. If the username of the email address or the reporter's full name matches the email address, we add the particular email address to the possible-reporters list.

We repeat the same process for every email address on the list. After the process finishes, we check the possible-reporters list of the report I. If the possible-reporters list is empty, we set the *ReporterEmail* field of the report as empty (there are 50 reports for which we cannot identify the email address of the actual external reporter during this data cleaning process). If the possible-reporters list is not empty, then we set the *ReporterEmail* field of the report with the list of email addresses in the potential-reporters list.

Even though there should be only one reporter for each report (i.e., the length of the potential-reporters list should be one), we observe some reports where multiple reporters are rewarded. This may happen when multiple external bug hunters report the same report to Google (not through the issue tracker). Google replicates these reports by posting a single report on the tracker via an internal reporter. In such cases, we let the reporter's email of the report be a list instead of a single email address. Note that for some of these reports with multiple reporters, we perform an additional verification in Step 4.

Step 4: Cleaning based on Chrome Releases Data

Further, we improve the data cleaning process of internal and external reports based on data collected from Chrome Releases (Appendix D.1). During the last step (Step 3), we mark the *ReporterEmail* field as empty for the reports where we are unable to determine the actual external reporter.

For each report I which marked the *ReporterEmail* field as empty in the last step (Step 3), we look for a data entry DE with the *IdentificationNumber* field same as the Identification Number field of report I. If a data entry exists in the Chrome Releases dataset, then we set the Report Email of report I with the Reporter Name in data entry DE. Accordingly, we are able to identify the actual external reporter details of 14 reports.

Further, during the last step (Step 3), we have more than one email address set to the *ReporterEmail* field for 13 reports. For each report I in those 13 reports, we look for a data entry DE with Identification Number field the same as the *IdentificationNumber* field of report I. If there exists a data entry (DE) in the Chrome

Releases dataset, then we check the value in the ReporterName field of *DE*; if it indicates a single person, then we update the *ReportEmail* field with the single person. We are able to update 8 out of 13 reports with multiple reporters to the actual external reporters. At the end of this step, we were left with 22 reports to go through an additional cleaning process to identify the original *ReporterEmail* field of the report.

For each reporter name *RN* used as the *ReporterEmail* field of the above 22 reports, we list out the reports (L_{RN}) reported by the reporter *RN* based on the Chrome Releases dataset. For each report in L_{RN}, we look for report *I*, which has the same Identification Number and Reporter Email in the email address format. If we obtain the report *I*, then we map the reporter name RN with the *ReporterEmail* field of report *I*; otherwise, we continue the same process with the next report in the list.

Finally, we repeat Step 1 with the cleaned dataset based on Steps 3 and 4. We identify the email address of the actual external reporter for 98% of valid external reports.

E.4.2 Firefox. Reports in Firefox VRP are reported either internally by Firefox internal team members or by external reporters.

We use four steps to separate internal reports from external reports. First, we use *Whiteboard* and *bug-flag* fields on the report information page. If a report has *reporter-external* in *Whiteboard* field or *sec-bounty* in *bug-flag* field, we consider that report as an externally reported report; otherwise, it is considered as an internally reported report. However, there are reports that do not have the above keywords in the mentioned fields, they are reported by external reporters. To separate them, we added three more cleaning steps. In the second step, we leverage the snow-balling technique (on the comments, such as "security@mozilla.org received the following report from") to identify reports (around 650 reports) reported by internal security members of Mozilla and do not have any bounty tag, but their original reporters are external. In the third step, we check reporters with both internal and external reports (around 50). We manually check whether they are internal or external (by reading comments and checking their social networking websites). In the last step of the cleaning process, we leverage *Reporter* field in MFSA and match the name of reporters in their profile names (we got each reporter's profile name from Bugzilla) with the name of the reporter mentioned in MFSA. By applying the above steps, we are able to separate internal versus external reports with 97% accuracy.

E.5 Vulnerability Attributes

For each duplicate report *D*, we clean security-severity, impacted releases, weakness types, components, and programming language attributes to make them consistent with its original report *O*. Accordingly, we perform the following cleaning steps:

- If the duplicate report *D* does not list any value for an attribute *A* and the original report *O* of duplicate report *D* has a value, then we set the value for attribute *A* of duplicate report *D* to the value of the original report *O*
- If all duplicate reports of original report *O* list the same value and original report *O* does not list any value for an attribute *A*, then we set the value for attribute *A* of original report *O* to the value of the duplicate reports.

Further explanations for different attributes based on the datasets are in the following sections.

E.5.1 Security-Severity.

Chromium. There are four severity levels in the Chromium issue tracker, which describe the security severity of a vulnerability: *Critical*, *High*, *Medium*, and *Low*. For each original report, we identify its security-severity level by extracting labels that start with *Security_Severity* from the *AllLabels* field of the report. If a label in the format of *Security_Severity-L* is available in the list of labels (where *L* is one of the four security-severity levels), then the severity of the report is *L*. If no labels are available in the format of *Security_Severity-L* in the list of labels, then we consider the severity of the report to be *unclassified*. For each duplicate report *D*, we use the security-severity level of the corresponding original report *O* instead of the security-severity level of the duplicate report *D*. We exclude *unclassified* vulnerabilities from the security-severity analysis.

Firefox. There are multiple security related keywords in the *Keywords* field of a report. We only include reports in our analysis that contain one of the 6 following security tags in their *Keywords* field: *sec-critical, sec-high, sec-medium, sec-low, sec-vector,* and *sec-other*. If a report has one of the above keywords in its *Keywords* field, we consider that report as a security report and include it for our analysis. There are some reports that have more than one security keyword. For those reports, we keep them as security reports in the dataset but exclude them from the analysis parts related to the security severity. We also realized that most of the collected reports that are opened in 2011 and before that year, do not have any security keywords assigned. Therefore, we only consider reports that are opened in 2012 and later for our analysis. For each duplicate report *D*, we use its corresponding original report's keyword as the duplicate security-severity field. There are reports with other security related keywords in their *Keywords* field which we exclude them since they are not actually security reports. For instance, reports which have *sec-want* which is 'New features or improvement ideas related to security' according to Mozilla keywords explanation are excluded.[21]

E.5.2 Release Channels.

Chromium. Google categorizes release versions as *stable*, *beta*, and *dev*. Stable is the release that is available for end-users. Beta is the release that is available for a limited number of users to test features before releasing a stable release. Dev (commonly referred to as *head*) is the release that is based on the last successful build. We use the term *release channel* to refer to these release versions throughout the paper. Note that the term release version means the type of the release instead of the version number (e.g., Version 90 and Version 91).

Each security vulnerability *I* impacts one or more release channels. To identify which security channel(s) is affected by vulnerability *I*, we check labels in *AllLabels* field that start with *Security_Impact*. Based on these, we identify three release channels during this process: stable, beta, and head. We group beta and head as development release channels and perform our analysis.

---

[21]https://bugzilla.mozilla.org/describekeywords.cgi

The Benefits of Vulnerability Discovery and Bug Bounty Programs

Published in the proceedings of the 2023 ACM Web Conference (WWW 2023).

Firefox. We followed the general approach we mentioned at the beginning of this section (Appendix E.5).

E.5.3 Components.

Chromium. For each report I, we identify the components from its *Component* field. Each report I will contain a set of components C_I. For each component c in the C_I, we extract the set of the group, which indicates a list of all sub-levels from the top level to the bottom level of the component hierarchy. For example, if report I has a component `Internals>Plugins>PDF`, we extract the set of the group as `Internals`, `Internals>Plugins`, `Internals>Plugins>PDF`. We use G_I to denote the set of all groups that correspond to all the components of the report I. Some of the pairs of original report O and its duplicate report D have one of the following inconsistencies.

- If the *Component* field of all duplicate reports of original report O are not the same and the *Component* field of the original report O is empty. Still, all duplicate reports of the original report O contain the same set of groups of components G_D. We set the *Component* field of the original report O to the value of the *Component* field of duplicate reports.
- If the *Component* field of all duplicate reports of original report O are not the same and the *Component* field of the original report O is not empty, then each pair of original report O and duplicate report D, we check whether it satisfies at least one of the conditions. If it is satisfied, then we set the *Component* field of duplicate report D to the value of the *Component* field of original report O
  - All the components of duplicate report D (C_D) in the set (C_O) or the set (G_O).
  - All the groups of the components of duplicate report D (G_D) present either in the set (C_O) or the set (G_O).

Firefox. We followed the general approach we mentioned at the beginning of this section (Appendix E.5).

E.6 Earliest Report and Fixed Timestamps

First reported Timestamp ($T_{earliest}$): the date and time of the first report of a vulnerability. For each valid original report O, we compute $T_{earliest}$ based on either one of the conditions:

- If the valid original report O reported before all of its duplicate reports, then we set $T_{earliest}$ to the value of *Opened-Timestamp* field of the valid original report O.
- If the valid original report O is reported after one or more of its duplicates, we list all the timestamps from the *Opened-Timestamp* field of all the duplicate reports of the valid original report O, then set $T_{earliest}$ to the minimum timestamp from the list of all timestamps.

Fixed Timestamps In Firefox. In the collected data, there are two vulnerabilities that both original and its duplicate has fixed time. We excluded those reports from our analysis related to fix. For some reports with *WORKSFORME* status in Firefox, there are not specific fixed time. If that report has *TrackingFlags* in its information page and it shows that the report got fixed in a version, we consider this report in our dataset as valid report, but we do not consider its fixed time.

E.7 Rediscovery

There are some vulnerabilities that are reported by the same origin reporter multiple times in both Chromium and Firefox. In the rediscovery analysis, we remove redundant reported reports and keep only the earliest report from the same origin. There are also reports that do not have accessible pages in Bugzilla. Since for those reports we cannot identify which report is the earliest one, we excluded them from the rediscovery analysis. In Firefox, some reports are incomplete with respect to replication and patching. For some of them, Mozilla opened a new report of the vulnerability, which was then completed with respect to this information, and marked the first report as a duplicate. We also exclude these vulnerabilities from our analysis since they are not actual rediscoveries.

E.8 Exploited Vulnerabilities

To identify exploited vulnerabilities, we first collect an initial set of exploited vulnerabilities from the Known Exploited Vulnerabilities Catalog of CISA[22]. Then, we extend this set iteratively using a snowballing method by identifying terms in comments related to exploitation (e.g., *exploited in the wild*) and gathering vulnerabilities whose comments include these terms. We manually verify the descriptions of these vulnerabilities to find false positives. Finally, we restrict the set to valid original security vulnerabilities, resulting in a set of 18 and 37 exploited vulnerabilities for Firefox and Chromium respectively.

---

[22]https://www.cisa.gov/known-exploited-vulnerabilities-catalog