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

ACM Reference Format:
Soodeh Atefi, Amutheezan Sivagnanam, Afiya Ayman, Jens Grossklags, and Aron Laszka. 2023. The Benefits of Vulnerability Discovery and Bug Bounty Programs: Case Studies of Chromium and Firefox. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583352

1 INTRODUCTION

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 public products and services may be targeted by myriads of attackers, who possess diverse expertise. Spearheaded by Netscape as a forerunner in 1995 [14], *bug-bounty programs*—which are also known as *vulnerability reward programs*—have emerged as a key element of many organizations' security culture [19, 26, 36]. Bug-bounty programs are a form of crowdsourced vulnerability discovery, which enables harnessing the diverse expertise of a large group of external bug hunters [14]. 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 [21]. By rewarding valid reports with bounties, the program incentivizes hackers to spend effort on searching for vulnerabilities and reporting them [1, 37]. 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. [14], Zhao et al. [36], Laszka et al. [20, 21], Maillart et al. [23], Luna et al. [22], Elazari [10], Malladi and Subramanian [24], and Walshe and Simpson [34]). 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

WWW '23, May 1–5, 2023, Austin, TX, USA

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 [30]. 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 [7, 31]. In contrast, other software projects may have relatively stable code bases over time, which also dominates the number of discovered vulnerabilities [29]. 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 this context, Schneier [32] conjectures 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 re-discovered vulnerabilities, constituting 7.69% of all vulnerabilities listed in the bulletins) [28]. 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 [16].

- 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 [6]. 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 [33]. 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 [5].

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*.

Due to lack of space, we describe the technical details of the data collection and cleaning processes in our online appendix [4].

[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.

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

[2] https://bugs.chromium.org/p/chromium/issues/
[3] https://chromium.googlesource.com/infra/infra/+/master/appengine/monorail/api/
[4] https://bugzilla.mozilla.org/home
[5] https://www.mozilla.org/en-US/security/known-vulnerabilities/
[6] https://www.mozilla.org/en-US/security/advisories/
[7] https://www.cvedetails.com/
[8] https://cwe.mitre.org/
[9] https://www.cisa.gov/known-exploited-vulnerabilities-catalog

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*.

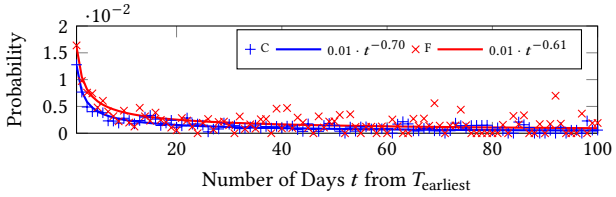
Before drawing any conclusions about the difficulty of finding vulnerabilities, we must also consider the number of unique external reporters working on stable and development releases (see Table 2). We find that in both Chromium and Firefox, *there are considerably more bug hunters who report vulnerabilities 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.

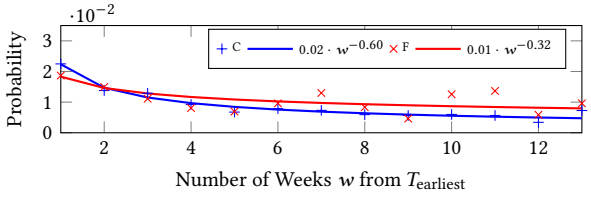
Finding 1. *The rediscovery probabilities, number of bug hunters, and mean patching times in conjunction suggest that vulnerabilities are easier to find in development releases; vulnerabilities that are easy to find are likely to be discovered and patched during development, demonstrating the benefits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

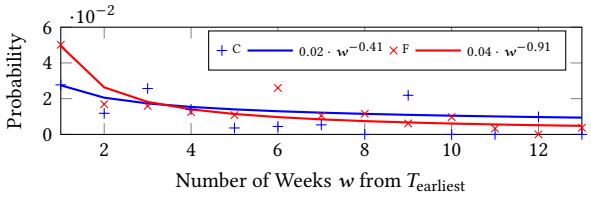
$$\left|\left\{o_d \mid d \in D, o_d \in O_{\text{fix}}, \Delta_{\text{rediscover}_d} = t\right\}\right| / |O_{\text{fix}}| \tag{1}$$



(a) Probability that a vulnerability is rediscovered on the t-th day after it is first reported ($\Pr[Re(t) | t < \Delta_{\text{fix}}]$).



(b) Probability that a vulnerability in a stable release is rediscovered in the w-th week after it is first reported ($\Pr[Re(w) \mid w < \Delta_{\text{fix}}]$).



(c) Probability that a vulnerability in a development release is rediscovered in the w-th week after it is first reported ($\Pr[Re(w) \mid w < \Delta_{\text{fix}}]$).

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.

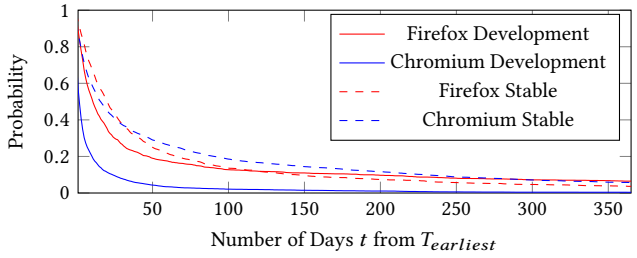


Figure 2: Probability that a vulnerability is not fixed in t days after it is first reported ($\Pr[\Delta_{fix} > t]$) in *development* (*solid lines*) and *stable* releases (*dashed lines*).

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 $\Pr[Re(w) \mid w < \Delta_{\text{fix}}]$ for the w-th week as follows:

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

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

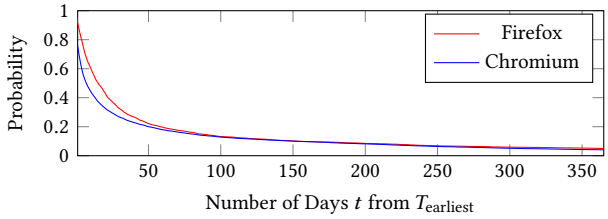


Figure 3: Probability that a vulnerability is not fixed in the t days after it is first reported ($\Pr[\Delta_{fix} > t]$).

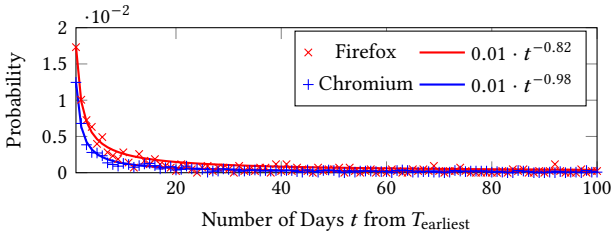


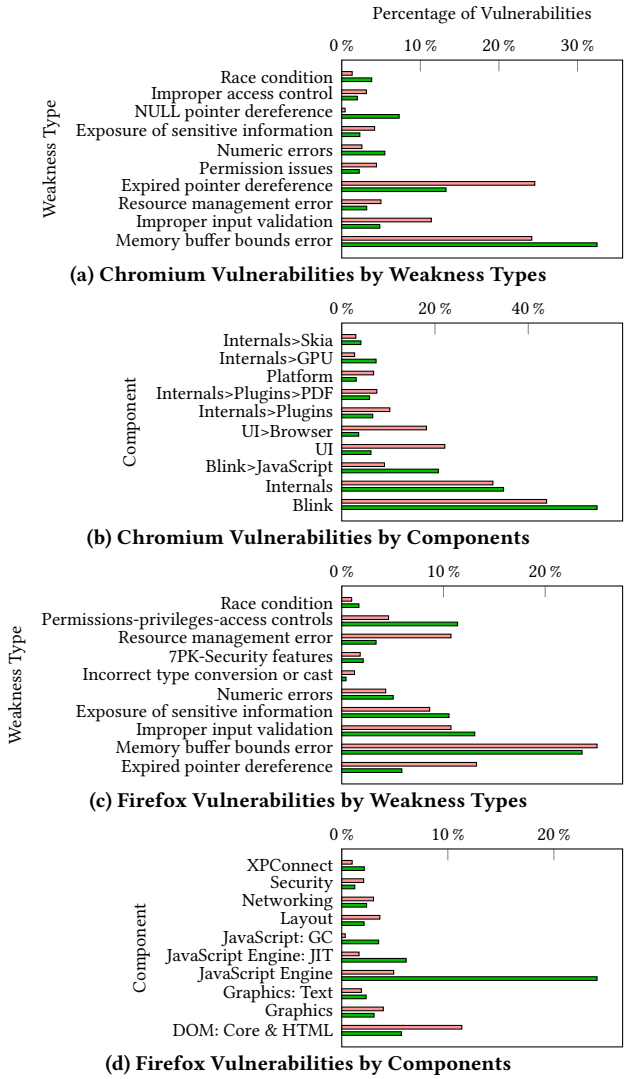
Figure 4: Probability that a vulnerability is rediscovered on the t-th day after it is first reported ($\Pr[Re(t)]$).

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.

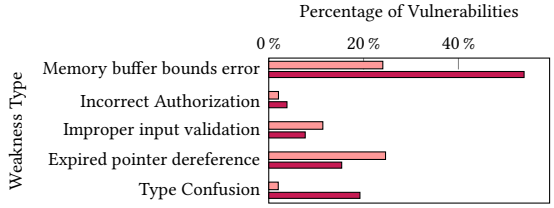
Based on Pearson's chi-squared test, external and internal reports follow significantly different distributions in terms of impacted release channels, severity level, weakness type, affected components, and programming languages for both Chromium and Firefox.

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.*

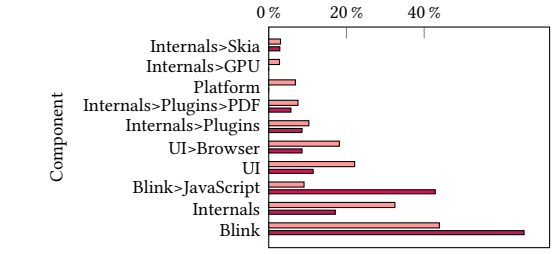
3.2 Management of Bug Bounty Programs

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.

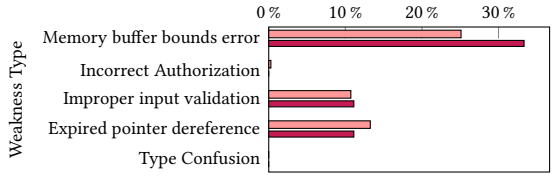
Comparison with Other Externally Reported Issues. Since our focus is on bug-bounty programs, we study the differences and similarities between vulnerabilities that are exploited by threat actors and vulnerabilities that are reported by external bug hunters (Fig. 6).



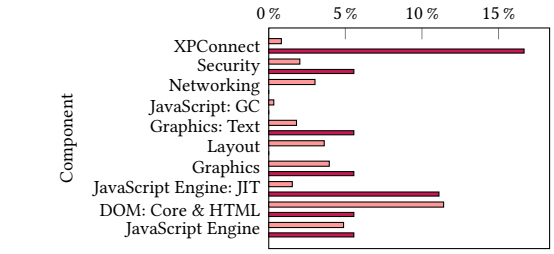
(a) Chromium Vulnerabilities by Weakness Types



(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.

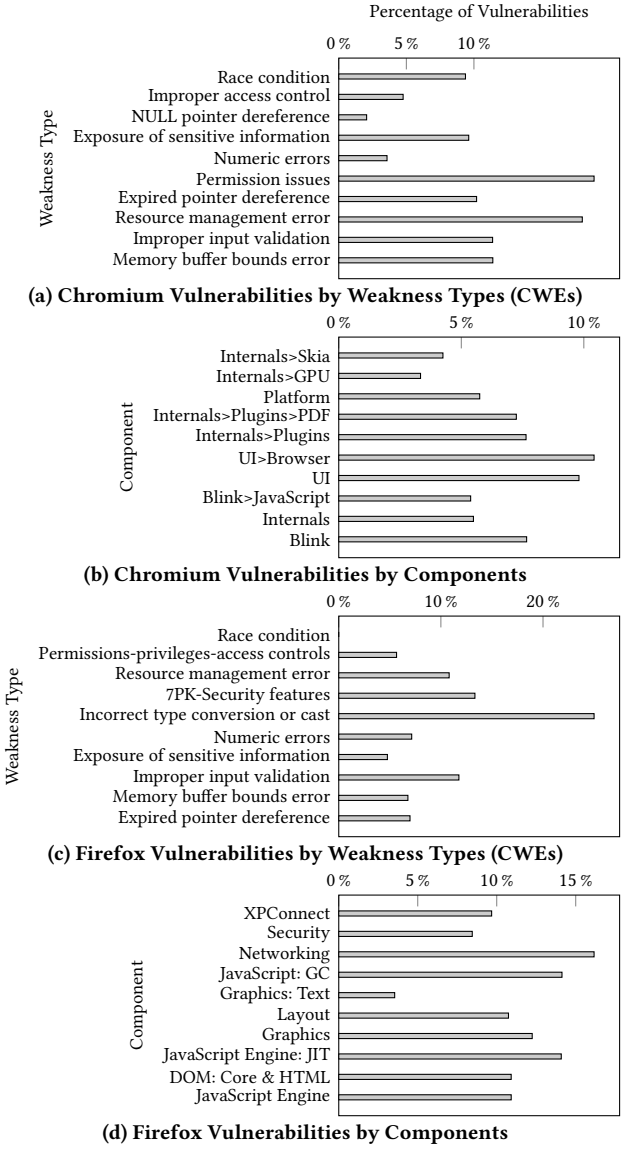
Our exploratory statistical tests show that for Chromium, exploited vulnerabilities and external reports follow significantly different distributions in terms of impacted release channels and security-severity levels; however, this does not hold for affected programming languages. Similarly, in Firefox,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



(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 [8, 11]. Taking an analytical and empirical perspective, Massacci and Nguyen [25] 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, 9, 10, 22, 23, 35, 36]) and security bulletins (e.g., [12, 13]); however, research on rediscovery of vulnerabilities is sparse. Ozment [28] 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. [14] 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. [17] estimated that vulnerability rediscovery occurs more often than previously reported (1% to 9%) in the literature (e.g., [27]) 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. [18] 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 lifecycle 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. [33], and Fulton et al. [15] 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.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1850510. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank the anonymous reviewers for their valuable feedback and suggestions.

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. *The benefits of vulnerability discovery and bug bounty programs: Case studies of Chromium and Firefox (Online Appendix)*. Technical Report. arXiv. https://doi.org/10.48550/arXiv.2301.12092

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

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

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

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

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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

[32] 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/.

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

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*

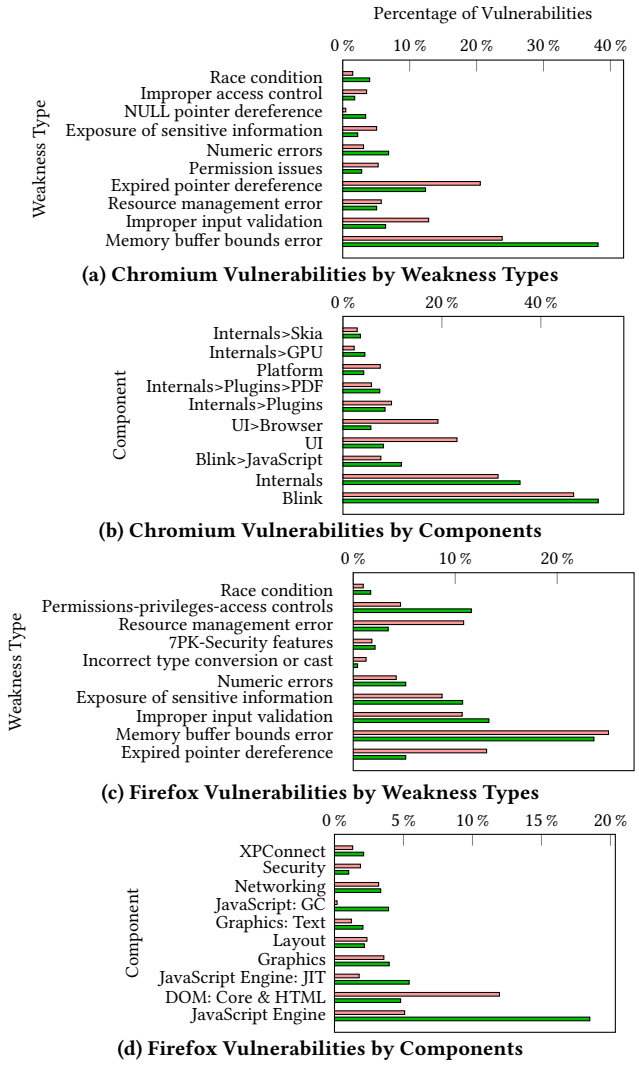


Figure 8: Comparison of internal (■) and external (■) security reports in *stable* releases of Chromium and Firefox.

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.

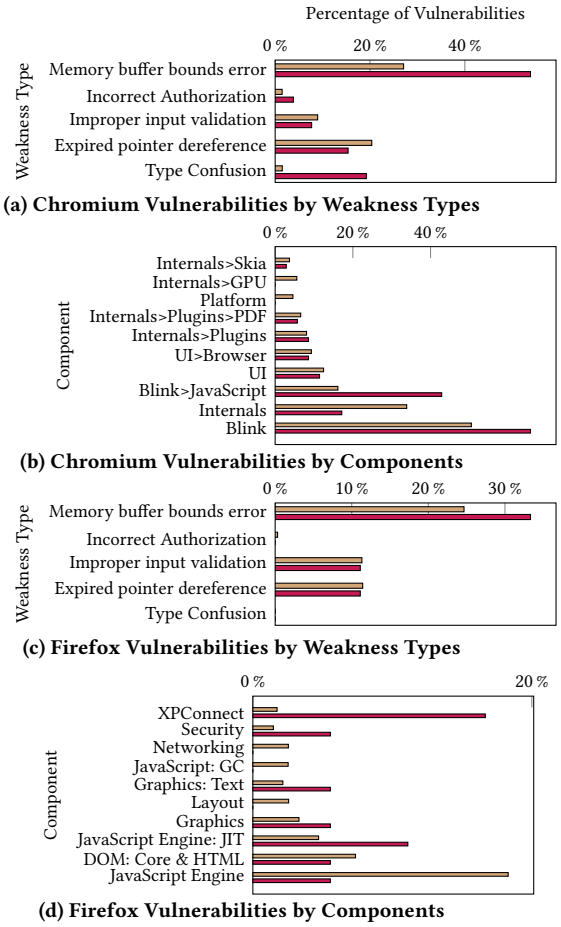


Figure 9: Comparison of exploited vulnerabilities (■) and all other vulnerabilities (■) in Chromium and Firefox.

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. 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).