Given Enough Eyeballs, All Bugs Are Shallow? Revisiting Eric Raymond with Bug Bounty Programs

Thomas Maillart*

University of Geneva, Geneva, Switzerland

Mingyi Zhao

Snap Inc., USA

Jens Grossklags

Technical University Munich, Germany

John Chuang

University of California, Berkeley, USA

(Dated: May 31, 2017)

Bug bounty programs offer a modern platform for organizations to crowdsource their software security, and for security researchers to be fairly rewarded for the vulnerabilities they find. However, little is known on the incentives set by bug bounty programs – how they drive new bug discoveries, and how they may improve security through the progressive exhaustion of discoverable vulnerabilities. This article investigates the strategic interactions among the managers and participants of bug bounty programs. At the level of a single bug bounty program, security researchers face a mechanism reminiscent of the St-Petersburg paradox: the probability of finding additional bugs decays rapidly, and is difficult to compensate for with increasing monetary rewards. Furthermore, bug bounty program managers have an incentive to gather the largest possible crowd of experts, which increases competition among security researchers. As a result, we find that researchers have strong incentives to switch to newly launched programs, for which a reserve of *low-hanging fruit* vulnerabilities is still available. Our results shed light on the technical and economic mechanisms underlying the dynamics of bug bounty program contributions, and

help inform the mechanism design of bug bounty programs as they get increasingly adopted by cybersecurity-savvy organizations.

* thomas.maillart@unige.ch

I. INTRODUCTION

On March 2nd, 2016, the Pentagon announced the launch of its first *bug bounty* program [1]. From this point on, one of the most paranoid organizations in the world would offer incentives to hackers to break into its systems and report found vulnerabilities for a reward. Although bug bounty programs have mushroomed in the last few years, this audacious announcement by a prominent defense administration may set a precedent, if not a standard, for the future of cybersecurity practice.

Software security has long been recognized as a challenging computational problem [2] that requires human intelligence as part of the solution. However, given the complexity of modern computer systems, human intelligence at the individual level is no longer sufficient. Instead, organizations are turning to tap the wisdom of crowds [3] to improve their security. Software security is not alone. Other disciplines have similarly turned to mobilizing people at scale to tackle their hard problems, such as sorting galaxies in astronomy [4], folding proteins in biology [5], recognizing words from low quality book scans [6], and addressing outstanding mathematics problems [7, 8].

These examples involve different aspects of human intelligence, ranging from pattern recognition (e.g., Captcha [6]) to higher abstraction levels (e.g., mathematical conjectures [7, 8]). It is not clear what kind of intelligence is necessary to find bugs and vulnerabilities in software, but it generally requires a high level of programming proficiency coupled with *hacking skills* – out-of-the-box thinking to find unintended uses for a software.

From hedonist pleasure to reputation building, to activism, motivations and incentives for hacking have evolved over time [9]. Among these, reputation and monetary incentives are increasingly put in place to entice security researchers to hunt for bugs. Bug bounty programs and the online platforms help set such incentives [10–13], while facilitating communication and transactions between security researchers and software editors. It however remains unclear how current mechanism designs and incentive structures will influence the long-term success of bounty programs. A better understanding of bug discovery mechanisms [14–16], and a better characterization of the utility functions of security researchers, organizations launching bug bounty programs and bug bounty platforms, will help shape

the way bug bounty programs evolve in the future.

In this study, we investigate a public data set of 35 public bug bounty programs from the *HackerOne* website. We find that with each vulnerability discovered within a bounty program, the probability of finding the next vulnerability in the program decreases more rapidly than the corresponding increase in payoff. Therefore, security researchers rationally switch to newly launched bounty programs at the expense of existing ones. This switching phenomenon has already been found in [13]. Here, we characterize it further, by quantifying how incentives evolve as more vulnerabilities get discovered in a program, and how researchers benefit in the long term by switching to newly launched programs.

[say something about the reflection between our results and Brady et al.]

This article is organized as follows. Related research is presented in Section II. Important features of the data set used here are detailed in Section III. We introduce the main mechanism driving vulnerability discovery in Section IV. Results are presented and discussed in Sections V and VI, respectively. We offer concluding remarks in Section VII.

II. BACKGROUND

Software reliability is an age-old problem [2, 17, 18]. Early empirical work on software bug discovery dates back to the time of UNIX systems [19], and over the years, numerous models for vulnerability discovery have been developed (see [16, 20] for some contemporary approaches). As early as in 1989, it was recognized that the time to achieve a given level of software reliability is inversely proportional to the desired failure frequency level [2]. For example, in order to achieve a 10^{-9} probability of failure, a software routine should be tested 10^9 times. Actually, the random variable $P(T > t) = 1/t$ corresponds to the Zipf's law [21, 22], which diverges as the random variable sample increases (i.e., no statistical moment is defined). Thus, there will always be software vulnerabilities to be discovered as long as enough resources can be provided to find them.

Taking a evolutionary perspective brings additional insights. Finding bugs is compara-

ble to the survival process involved in the selection of species: Defects are are like genes, which get expressed under the pressure of environment changes. Brady et al. [15] have shown that software testing follows the principle of entropy maximization, which preserves *genetic variability* and thus, removes only the minimum possible number of bugs, following the exploration of use cases (the software environment). With her out-of-the-box thinking, the *hacker* security researcher is precisely good about envisioning a broad range of possible use cases, which may reveal a software defect (i.e., program crash) or an unintended behavior.

Software solutions have been developed to systematically detect software inconsistencies and thus potential bugs (e.g., Coverity, FindBugs, SLAM, Astree, to name a few). However, to date, no systematic algorithmic approach has been found to detect and remove bugs at a speed that would keep pace with software evolution and expansion. Thus, human intelligence is still considered as one of the most efficient ways to explore novel use case scenarios – by manual code inspection or with the help of bug testing software – in which a software may not behave in the intended way.

Management techniques and governance approaches have been developed to help software developers and security researchers in their review tasks, starting with pair programming [23]. To protect against cyber-criminals, it is also fashionable to hire *ethical hackers*, who have a mindset similar to potential attackers, to probe the security of computer systems [24–26]. The policy of full disclosure, originating from the hacking and open source communities, have played a significant role in software security by forcing software owners to acknowledge and fix vulnerabilities discovered and published by independent researchers [27]. The full-disclosure model has evolved into responsible disclosure, a standard practice where the security researcher agrees to allow a period of time for the vulnerability to be patched before publishing the details of the uncovered flaw. In most of these successful human-driven approaches, there is a knowledge-sharing component, either between two programmers sitting together in front of a screen, ethical hackers hired to probe the weaknesses of a computer system, or the broader community being exposed to open source code and publicly disclosed software vulnerabilities [28]. Thus, Eric Raymond's famous quote "Given enough eyeballs, all bugs are shallow" [29] tends to hold, even though in practice things are often slightly more complicated [30].

One way to gather enough eyeballs is to resort to the larger crowd of security researchers. For this purpose, bug bounty programs and vulnerability *markets* have emerged in recent years to facilitate the trading of bugs and vulnerabilities. These two-sided markets provide economic incentives to support the transfer of knowledge from security researchers to software organizations [31], as they help simultaneously harness the wisdom of crowds and to reveal the security level of organizations through a competitive incentive mechanism [32]. Nonetheless, the efficiency of bug bounty programs has been questioned on both theoretical [33, 34] and empirical grounds [35, 36].

Building on previous work by Schechter [32], Andy Ozment [37] theorized that the most efficient mechanisms are not markets *per se*, but rather auction systems [38]. In a nutshell, the proposed (monopsonistic) auction mechanism implies an initial reward $R(t = t_0) = R_0$, which increases linearly with time. If a bug is reported more than once, only the first reporter receives the reward. Therefore, security researchers have an incentive to submit a vulnerability early (before other researchers submit the same bug), but not too early, so that they can maximize their payoff $R(t) = R_0 + \epsilon \times t$ with ϵ the linear growth factor, which is meant to compensate for the increasing difficulty of finding each new bug. However, setting the right incentive structure $\{R_0, \epsilon\}$ is non-trivial given uncertainties in the amount of work needed, the level of competition (e.g., the number of researchers enrolled) in the bug bounty program [39], or the nature and likelihood of overlap between two submissions by different researchers. Nevertheless, bug bounty programs have emerged as a tool used by many software organizations, with a range of heterogeneous incentive schemes [11]. For instance, some bug bounty programs include no monetary rewards [12]. Meanwhile, dedicated platforms have been launched to act as trusted third parties in charge of clearing transactions between organizations and security researchers. These platforms also assist organizations in the design and deployment of their own program. One of the leading platforms is HackerOne which runs public and private programs for organizations across a wide range of business sectors. A subset of the private and public programs award bounties, while other bug bounty programs capitalize on incentives associated with reputation building, which is an important motivation driver in the hacker community [40]. These programs report bounty awards on their company program pages on the HackerOne website. Previous research has investi-

gated vulnerability trends, response and resolve behaviors, as well as reward structures of participating organizations [12, 13]. In particular, it was found that a considerable number of organizations experienced diminishing trends for the number of reported vulnerabilities, even as the monetary incentives exhibit a significantly positive correlation with the number of vulnerabilities reported [13].

III. DATA

The data were collected from the public part of the HackerOne website. From 35 public bounty programs, we collected the rewards received by security researchers (in US dollars), with their timestamps (45 other public bounty programs do not disclose detailed information on rewards, and the number of private programs is not disclosed). Since HackerOne started its platform in December 2013, new public programs have been launched roughly every two months, following an essentially memoryless Poisson process ($\lambda = 57$ days, $p < 0.001$ and $R^2 > 0.99$). Figure 1A shows the timeline of the 9 most active programs with at least 90 valid (i.e., rewarded) bug discoveries, as of February 15, 2016. When a new program is launched, we observe an initial peak within weeks after launch, which accounts for the majority of discoveries. Following the initial surge of vulnerability discoveries, bounty awards become less frequent following a decay function with long-memory, following a robust power law decay $\sim t^{\alpha}$ with $\alpha = -0.40(4)$ ($p < 0.001$ and $R^2 = 0.79$) at the aggregate level and over all 35 bounty programs (see Figure 1B). Some programs depart from this averaged trend: For instance Twitter exhibits a steady, almost constant, bug discovery rate and VKontakte exhibits its peak activity months after the initial launch. These peculiar behaviors may be attributed to program tuning and marketing, to sudden change of media exposure or even to fundamental differences of program comparative fitness, for which we do not have specific information.

The long-memory process of bug discovery following the launch of a bug bounty program (c.f., Figure 1B) is compatible with the long-established $\sim 1/\tau$ law of bug discovery in software testing [2]. The difference of exponent $\alpha \approx 1$ instead of 1. in the $\sim 1/\tau$ law of software reliabiltiy, τ refers to testing time, i.e., the number of tests to perform, in order to reach a given reliability level. The long-memory process associated with the decay of bug

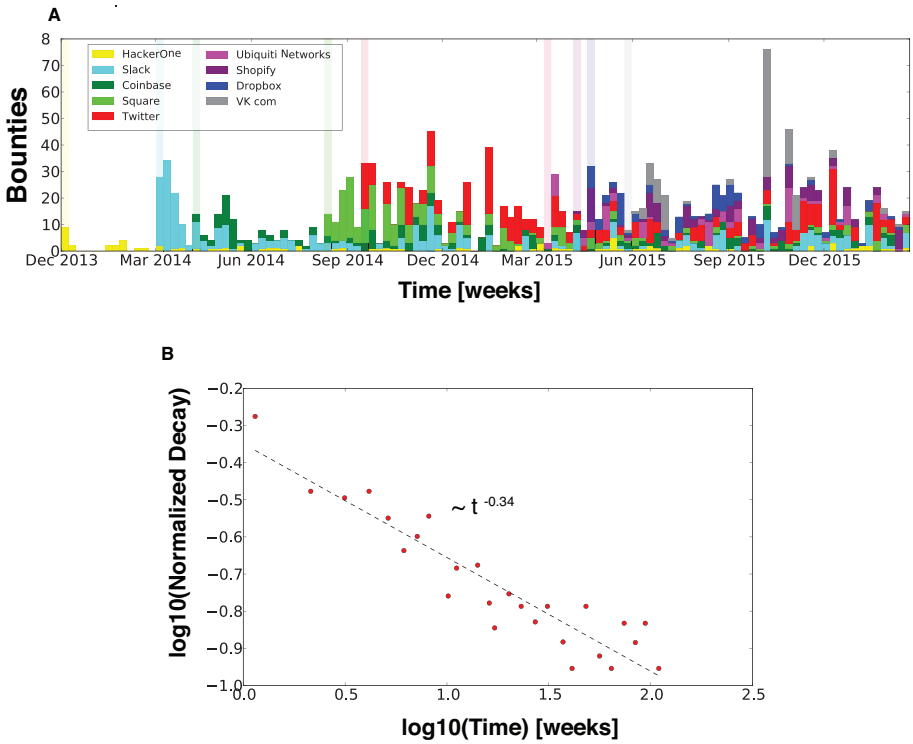

FIG. 1. **A.** Weekly vulnerability discoveries for the 9 most active programs (with at least 90 bug discoveries as of February 15, 2016). The light colored vertical bars represent the start of the program, occurring when the first bounty is awarded. Most programs exhibit an initial shock, followed by a decay of discoveries, which is characterized at the aggregate level by a long-memory process (panel **B**) characterized by a power law decay $\sim t^{\alpha}$ with $\alpha = -0.40(4)$ ($p < 0.001$ and $R^2 = 0.79$). Each data point in the figure is the median of normalized vulnerability numbers of all 35 programs considered in this study.

discovery (c.f., Figure 1B) is presented in absolute time. Long-memory processes associated with human behaviors are often associated with human timing effects, which correspond to task priority queueing and a rationalize use of time as non-storable scarce resource [41]. Long-memory processes observed in human behaviors are also associated with critical cascades of events. **Self-excited Hawkes Process with power law kernel (following human timing)[42]. Here, the rationale for cascades builds on 2 ingredients : (i) each security researcher generates a cascade of bug discoveries, and (ii) each bug discovery attracts new security researchers, who in turn trigger their own bug discovery cascade. Cascades are typically slightly heavy-tailed, however bounded, as we'll later : we will dissect these cascades in conjunction with in-**

centives provided by the accumulation of bounties (reputation, slight chance to find less obvious bugs bring more reputation, and increased bounties rewards).

~~: When the program launches, it takes some time first for the researcher to be exposed to the new program (through the media and social media), second for the researcher to find and submit bugs, and third for the organization managing the bug bounty program to assess the quality of each submission, and assign a proper reward. To account for all these delays, one may resort to priority queueing applied to humans: First, competing attention prevents immediate exposure to the news of a new program; Second, when security researchers get interested in a new program, they may still be actively searching bugs on other programs or performing other tasks (such as e.g., their regular job, leisure, family matters); Third, when subjected to a flow of bug submissions, security teams at organizations leading bug bounty programs assign priorities among submissions, and resolve them with human resources available at the time of submission.~~

~~These delays are best rationalized by human timing contingencies, and moreover, by an economy of time as a scarce, non-storable resource, which is known to generate long-memory responses of the form $\sim t^{-1.5}$ between the arrival and the execution of a task [41]. The observed much slower decay may result from the compound effect of multiple delays, such as those mentioned above. The initial burst of discoveries, followed by a long-memory decay may also result from the increasing difficulty associated with finding new bugs for each bounty program, as the most obvious vulnerabilities get uncovered first. Since we consider only the *time of discovery* as the moment when the validity of the bug submitted is acknowledged by the program manager, we are mostly blind to the human timing effects associated with the long-memory process observed on Figure 1B, including when submissions are made, but don't lead to a discovery associated with a monetary reward.~~

Here, we don't consider the timing effects encompassing delays, effort, processing time, but rather the incremental valid bug discovery and reporting by security researchers [refine here].

IV. METHOD

Bug bounty programs work on the premise that humans as a crowd are efficient at searching and finding bugs. Their mere existence is a *de facto* recognition that market approaches for bug and security vulnerabilities discovery involving large pools of security researchers bring efficiency, beyond in-house security. Bug bounty programs signal that organizations are ready to complement their vertical cost-effective security operations with market approaches, which are traditionally perceived as less cost-effective, yet more adaptive [43]. Brady et al. [15] have early on offered a hint for the existence of such markets for bugs: According to their proposed theory, each researcher has slightly different skills and mindset. When a security researcher tests a software piece, by choosing the inputs, she offers a unique operational environment. This environment is prone to the discovery of new bugs, which may have not be seen by others. The proposed theory by Brady et al. [15] intrinsically justifies the existence of bug bounty programs structures as markets, which provide the necessary diversity to account for the highly uncertain risk horizon of bug discovery. Here, we develop a quantitative method to formalize a mechanism and to test the theory proposed in [15]. This validation step shall help provide organizational design insights for bug bounty programs.

For that, we investigate the interplay between the vulnerability exhaustion process, and the cumulative reward distributed to security researchers within and across 35 public bounty programs hosted on HackerOne. When a bug bounty program starts, it attracts a number of security researchers, who in turn submit bugs. Subsequent bug discoveries get increasingly difficult. The difficulties faced by security researchers can be technical. It can also be the result of insufficient or conflicting incentives. Here, we develop and test a model, which accounts for technical difficulties and insufficient incentives together. We further address conflicting incentives by measuring the effect of newly launched bug bounty program on incumbent programs.

Starting from an initial probability of discovering the first vulnerability $P(k = 0) = 1$, the probability to find a second bug, is a fraction of the former probability: $P_{k+1} = \beta * P_k$ with β a constant strictly smaller than one. The probability that no more discovery will be made after k steps is given by $P_k = \beta^k(1 - \beta)$. Conversely, starting from the initial

reward $R_0 = R(k = 0)$, the subsequent reward $R_1 = \Lambda_1 \cdot R_0$, and further additional reward $R_2 = \Lambda_2\Lambda_1 \cdot R_0$. After n steps, the total reward is the sum of all past rewards:

$$R_n = R_0 \sum_{k=1}^{n} \Lambda_1 ... \Lambda_k. \tag{1}$$

Thus, R_n is the recurrence solution of the Kesten map ($R_n = \Lambda_n R_{n-1} + R_0$) [44, 45]: As soon as amplification occurs (technically, some of the factors Λ_k are larger than 1), the distribution of rewards is a power law, whose exponent μ is a function of β and of the distribution of the factors Λ_k. In the case where all factors are equal to Λ, this model predicts three possible regimes for the distribution of rewards (for a given program): thinner than exponential for $\Lambda < 1$, exponential for $\Lambda = 1$, and power law for $\Lambda > 1$ with exponent $\mu = |\ln \beta| / \ln \Lambda$ (see Appendix). The expected payoff of vulnerability discovery is thus given by,

$$U_k = P_k \times R_k, \tag{2}$$

with both P_k and R_k random variables respectively determined by β and Λ. Because U is a multiplication of two diverging components, its nature is reminiscent of the St. Petersburg paradox (or St. Petersburg lottery), proposed first by the Swiss Mathematician Nicolas Bernoulli in 1713, and later formalized by his brother Daniel in 1738 [46]. The St. Petersburg paradox states the problem of decision-making when both the probability and the reward are diverging when $k \to \infty$: A player has a chance to toss a fair coin at each stage of the game. The pot starts at two and is doubled every time a head appears. The first time a tail appears, the game ends and the player wins whatever is in the pot. Thus the player wins two if a tail appears on the first toss, four if a head appears on the first toss and a tail on the second, eight if a head appears on the first two tosses and a tail on the third, and so on. The main interest of Bernoulli was to determine how much a player would be ready to pay for this game, and he found that very few people would like to play this game even though the expected utility increases (in the simplest case proposed by Bernoulli, $U_n = \sum_{k=0}^{n} U_k = n$) [46]. The situation of a security researcher differs from the St. Petersburg lottery as bug search costs are incurred at every step. Since these costs influence the probability to find an additional bug, we could assume that they are integrated into the utility as $U_k^* = U_k - c_k$. The security researcher may also decide

to stop searching for bugs in a program, at any time k. This is equivalent to setting $P_{k+1} = 0$.

The expected payoff U_k therefore determines the incentive structure for security researchers, given that the bounty program manager can tune R_0 and in some cases, the manager may also set Λ. The utility function may also incorporate non-monetary incentives, such as reputation: Finding a long series of bugs may signal some fitness for a bug bounty program and as thus, create a permanent job opportunity [47]. Similarly, discovery of a rare bug that no other researcher has found before has a strong signaling effect, which can help build a career. However, these strategies are high-risk high-return, therefore they carry additional fame. In the next section, we will calibrate our model to the bug discovery process associated with 35 bounty programs publicly documented on the HackerOne platform.

~~In principle, the manager could set R_0 to influence P_0 and indirectly P_k. Mapping the discovery rank k into the rate of discovery may also help considering discounting aspects in presence of competing opportunities and inter-temporal choices under uncertainty [48]. A new public bounty program is launched at a Poisson rate, approximately every 2 months, and each launch brings its windfall effect, leaving the researcher with the choice to either keep digging increasingly harder vulnerabilities (rare but with higher reward), or turning to the low hanging fruit (frequent but with low reward) of a new program. We shall therefore verify whether newly launched programs actually influence security researchers.~~

V. RESULTS

The discovery process in a bug bounty program is driven by the probability to find an additional bug given that k bugs have already been discovered (i.e., the exhaustion process), and program managers aim to maximize B_c, the total number of bugs found. Our results show that the number of bugs discovered is a super-linear function of security researchers who have enrolled in the program (see Figure 2A). While bug bounty programs benefit from the work of a large number of researchers, researchers overall benefit from diversifying their efforts across programs (see Figure 3C). This benefit is particularly tangible regarding the cumulative reward they can extract from their bug hunting activity. In particular, we illustrate how researchers take the strategic decision to enroll in a newly launched program,

at the expense of existing ones they have formerly been involved in.

A. Security researcher enrollment determines the success of a bug bounty program

As presented on Figure 2A, we find that the number B_c of bug discovered scales as $B_c \sim h^{\alpha}$ with $\alpha = 1.10(3)$ and h the number of security researchers enrolled in a program. Since $\alpha > 1$, a bounty program benefits in a super-linear fashion from the enrollment of more researchers. This result is reminiscent of productive bursts and critical cascades of contributions in open source software development [42]: Each enrollment (i.e., *mother* event) initiates a cascade of bug discoveries (i.e., *daughter* events). Here, each cascade stems from a single security researcher (with some exceptions researchers mostly search bugs alone) and the nature of these cascades is captured at the aggregate level by their size as a random variable. As shown on Figure 2B, the distribution of bounty discoveries per researcher and per program follows a power law tail $P(X > x) \sim 1/x^{\gamma}$ with $\gamma = 1.63(7)$. The first moment of the distribution (i.e., the mean) is however well-defined (as a result of $\gamma > 1$). Moreover, we observe an upper cut-off of the tail with $x_{max} \approx 400$ bounties. Thus, each enrollment of a security researcher in a program provides a statistically bounded amount of new bug discoveries.

B. Security researchers are incentivized to diversify their contributions across bug bounty programs

For security researchers, the main metric is the expected cumulative payoff earned from the accumulation of bounty awards over all programs. This expected payoff is governed by the probability to find a given number of bugs and their associated payoff, as discussed in Section IV. To fully understand the incentive mechanisms at work, we consider 3 perspectives: (i) the expected cumulative down-payment made by bug bounty program managers (see Figure 3A), the expected cumulative payoff from the viewpoint of a security researcher for (ii) one program (see Figure 3B), and for (iii) all programs (see Figure 3C).

The average cumulative down-payment per program exhibits a super-linear scaling as

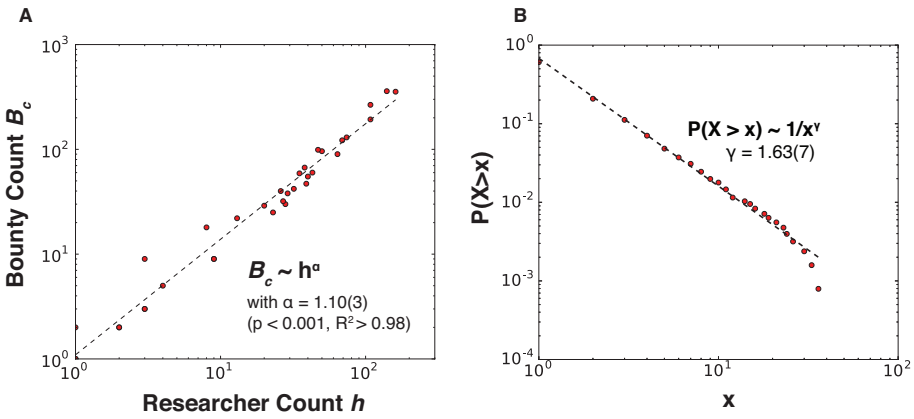

FIG. 2. **A.** The number of bounty discoveries per program B_c scales as h^{α} with $\alpha = 1.10(3)$ and h the number of security researchers enrolled in a program (fit and confidence interval were obtained by ordinary least squares of the logarithm of researcher and bounty counts). Since $\alpha > 1$, a bounty program benefits in a super-linear fashion to the enrollment of more researchers. **B.** The tail distribution of bounty discoveries per researcher per program follows a power law distribution $P(X > x) \sim 1/x^{\gamma}$ with $1 < \gamma = 1.63(7) < 2$ (obtained by maximum likelihood estimation and confidence interval bootstrapping, following [21, 49]). The distribution is therefore relatively well bounded (with the first moment being well-defined). Furthermore, we observe an upper cut-off of the tail with $x_{max} \approx 400$ bounties. Thus, from **A.** and **B.** combined, we find that the number of vulnerabilities is mainly driven by the number of researchers enrolled in programs.

$\sim k^{1.27}$ ($p < 0.001$, $R^2 > 0.99$), while the frequency of bugs P_k is only slightly upwards trended, increasing as $\sim k^{0.13}$ ($p < 0.001$, $R^2 = 0.40$). The expected down-payment by bug bounty program managers therefore scales as $\sim k^{1.40}$. This is a considerable super-linear increase (as $k \to \infty$), which casts questions on the long-term sustainability of bug bounty programs.

From the security researcher's viewpoint and her expected payoff from a single bug bounty program, the increase of average cumulative reward ($R_k \sim k^{1.40}$) does not offset the fast decay of probability ($P_k \sim k^{-1.85}$) to find a vulnerability of rank k. The expected payoff therefore follows $U_k \sim k^{-0.45}$, which clearly does not bring high incentives to explore in depth a bug bounty program. It is important to note however, that the bug bounty manager cannot fix P_k, which is a genuine feature of bug discovery by humans. To maintain positive individual incentives, the manager should set an incremental reward such that $R_k \sim k^{\alpha}$

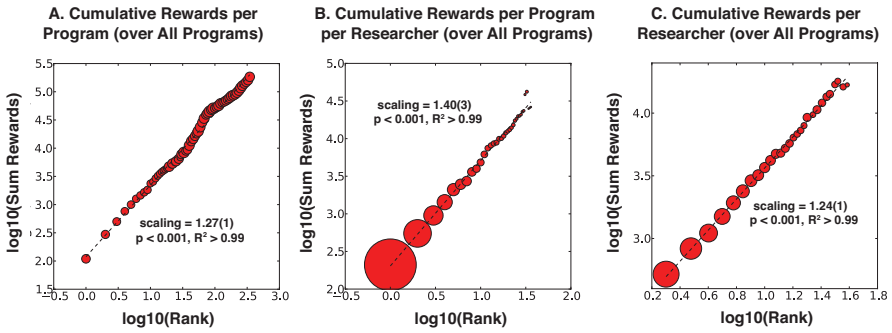

FIG. 3. **A.** (Log-)binned cumulative down-payment per program over all public programs on the HackerOne platform, scales as $R_k \sim k^{1.27}$ ($p < 0.001$, $R^2 > 0.99$), with k the rank. Each log-bin shows the mean value and the circle sizes depict the number of values in each bin (i.e., the rank frequency). The super-linear *scaling* relationship between the cumulative reward and the rank shows that reward increases as a function of k. However, the frequency of vulnerabilities P_k is only slightly upwards trended increasing as $\sim k^{0.13}$ ($p < 0.001$, $R^2 = 0.40$). **B.** Considering the opposite figure from the viewpoint of researcher's expected payoff when enrolling in a single bug bounty program, the super-linear effect is much stronger ($R_k \sim k^{1.40}$ with $p < 0.001$ and $R^2 > 0.99$). However, the frequency decays following a power law of the form $P_k \sim k^{-1.85}$ ($p < 0.001$, $R^2 = 0.97$). **C.** Over all bug bounty programs, security researchers have another expected payoff: the reward scaling is smaller ($R_k \sim k^{1.24}$ with $p < 0.001$, $R^2 > 0.99$), yet the frequency of bug discoveries decays much slower as a function of rank $P_k \sim k^{-1.05}$ ($p < 0.001$, $R^2 = 0.85$).

with $\alpha > 1.40$, which in turn would worsen the down-payment function both in terms of incremental expenditures and in exploration of bugs with higher ranks. This approach does not consider possible individual human hard limits preventing finding additional bugs, which would drive iterative costs c_k sufficiently high as k gets large. In that latter case, setting higher reward incentives would have no effect.

Security researchers tend to switch from one bounty program to another program [12, 13]. The strategy can be assimilated to portfolio diversification [50]. Over all bug bounty programs, security researchers have another much more favorable expected payoff: The reward scaling is smaller ($R_k \sim k^{1.24}$ with $p < 0.001$, $R^2 > 0.99$), yet the frequency of bug discoveries decays much slower as a function of rank $P_k \sim k^{-1.05}$ ($p < 0.001$, $R^2 = 0.85$).

Therefore, over all bounty programs, security researchers have an increasing, yet marginally decreasing, incentive to explore higher ranks as $U_k \sim k^{0.19}$. In a nutshell, security researchers have an incentive to keep searching for bugs on a large variety of bug bounty programs.

C. Influence of newly launched programs on researcher behaviors

As security researchers weigh their strategic choice to switch their attention from one program to another, the time factor is determinant because the expected payoff is dependent on the current vulnerability rank, which maps into the time dimension (i.e, the duration between two discoveries is drawn from a characteristic random variable, which is not considered here). While a researcher may decide to switch at any time, the most obvious moment is when a new program is being launched: Incentives shift suddenly, and security researchers may decide to abandon older programs at the expense of the new program with fresh bug discovery opportunities. However, a number of factors may influence their decision: The reputation of the organization launching the program (it brings more fame to submit a bug to e.g., Twitter compared to a less famous organization), the amount of reward, and the relative time between an old program and the newest one. To encompass the effects of new public bug bounty programs on incumbent programs, we aim to test 3 hypotheses:

- **H1:** An existing bounty program will receive less reports when more new programs are launched,
- **H2:** An existing bounty program will receive less reports when bounty rewards provided by newly launched programs is higher,
- **H3:** The number of newly launched programs has a larger impact on the contribution to older programs.

We specify a simple ordinary least square (OLS) regression model as follows:

$$V_{it} = \beta_0 + \beta_1 dP_t + \beta_2 T_{it} + \beta_3 A_{it} + \beta_4 B_{it} + \epsilon_{it}. \tag{3}$$

V_{it} is the number of vulnerability reports received by bounty program i in the month t. dP_t is the number of new programs launched in month t. Hypothesis **H1** predicts that its coefficient (β_1) is negative. T_{it} is the number of months since bounty program i launched.

We consider two control variables that could influence researcher's decision [13]. We first incorporate A_i the log of the Alexa rank, which measures web traffic as a proxy of popularity for organization i. B_i is the log of the average amount of bounty paid per bug by bounty program i. Both A_i and B_i are assumed to remain constant over time. Finally, ϵ_{it} is the unobservable error term. In models **2-4**, we extend the basic model (model **1**) to further study competition occurring between bounty programs. These alternative specifications include:

- **Average bounty of newly launched programs:** Intuitively, if new programs offer higher rewards, they should attract more researchers from existing programs. We calculate the average bounty for all new programs in month t as NB_t in models **2-4**.

- **Interaction between dP_t and T_{it}:** Conceivably, the effect of new programs on existing programs depends on certain characteristics of the latter, such as age. In particular, we ask if a new entrant has more negative effects on older programs compared to younger programs? To examine this, we consider an interaction term between the number of new programs (dP_t) and the age of the program (T_{it}) in models **3-4**. Hypothesis H3 predicts that its coefficient is negative.

- **Program fixed effect:** To better control for program-specific, time-invariant characteristics, e.g., the reputation among researchers, we add program fixed effect in model **4**. The addition of this fixed effect allows us to examine how bug discovery changes over time within each program i.

The regression results are shown in Table I. Consistent with our prediction in Hypothesis **H1**, the coefficient of dP_t is negative and statistically significant in all 4 specifications. *Ceteris paribus*, the launch of new programs reduces the number of vulnerabilities reported to existing programs. In other words, the entry of new programs indeed attracts researcher's attention away from existing programs, which is consistent with the fast decreasing expected payoff for individuals searching bugs for a specific program. Also, the average bounty paid by new programs ($B_{new,t}$) has a negative effect on existing programs as well, but the coefficient is only significant in model **4**. Again this result is consistent with the theory and Hypothesis **H2**, as researchers have a higher incentive to switch to new programs, if they

TABLE I. Regression results.

	(1)	(2)	(3)	(4)
VARIABLES	V_{it}	V_{it}	V_{it}	V_{it}
dP_t	-1.235***	-1.350***	-2.310***	-1.236**
	(0.305)	(0.327)	(0.603)	(0.515)
A_i	-23.61***	-23.72***	-23.72***	-7.188**
	(2.140)	(2.156)	(2.152)	(3.473)
B_i	16.64***	16.56***	16.75***	-7.414
	(1.311)	(1.315)	(1.339)	(5.698)
T_{it}	-0.690	-0.658	-3.312***	-3.758***
	(0.426)	(0.427)	(1.239)	(1.128)
$B_{new,t}$		-0.0445	-0.0312	-0.0321*
		(0.0280)	(0.0277)	(0.0184)
$T_{it} \times dP_t$			0.106**	0.0755*
			(0.0431)	(0.0406)
Constant	160.2***	170.4***	190.3***	136.5***
	(16.12)	(18.80)	(23.17)	(26.17)
Observations	1,212	1,212	1,212	1,212
R-squared	0.314	0.316	0.319	0.647
Program FE	No	No	No	Yes

Robust standard errors in parentheses

*** p<0.01, ** p<0.05, * p<0.1

have more low-hanging fruits and higher bounty.

The interaction coefficients for term $T_{it} \times dP_t$ in models **3** and **4** are positive and statistically significant, so they do not support Hypothesis **H3**. The result shows that the impact of newly launched programs depends on the age of the existing programs: Compared to younger programs, the negative impact of dP_t is smaller for programs with a longer history,

i.e., those with larger T_{it}. At first sight, this results may look at odds with the fact that individual expected payoff from a specific program decreases as a function of rank k, and presumably the older a program the more likely it has a high rank. Thus, the switching effect should be stronger. Perhaps our OLS regression model is limited in the sense that it does account for the absolute activity (which decreases very slowly as $t \to \infty$, as shown on Figure 1B), instead of the variation rate. Consistent with previous research [13], the regression results also show that a program with higher reputation (A_i) or higher bounty (B_i) is associated with more bugs received in a month. The regression results also show that a program with longer age (T_{it}) is associated with less bugs found. This observation corresponds to the power law decay of bug submission observed following a program launch (c.f., Figure 1B).

VI. DISCUSSION

Finding bugs in software code is one of the oldest and toughest problems in software engineering [2]. While algorithm based approaches have been developed over the years [20], human verification has remained a prime way for bug hunting. While resorting to the crowd for finding bugs is not new [51], bug bounty programs have recently been promoted by the emergence of bug bounty platforms. Here, we have studied incentive mechanisms across 35 bug bounty programs on HackerOne, one of such platforms. Our results show that the number of discovered bugs and vulnerabilities in a bounty program is super-linearly associated with the number of security researchers. However, the distribution of bugs found per researcher per program is bounded: In a given bug bounty program, the marginal probability of finding additional bugs is decreasing rapidly. On the contrary, security researchers have high incentive to switch among multiple bug bounty programs. We find indeed that each newly launched program has a negative effect on submissions to incumbent bug bounty programs. Furthermore, controlling specifically for monetary incentives, we find that the reward amount for valid bugs in newly launched programs has a negative effect on the number of bug submissions to incumbent programs.

We find quantitative evidence for the incentive mechanisms associated with bug discovery by crowds of security researchers, and help draw practical organization design recommen-

dations for bug bounty platforms such as HackerOne, as well as for organizations managing bug bounty programs.

Our results provide an essential validation step to the theory formulated by Brady et al. [15]. No single security researcher is able to find most bugs in one program. On the contrary, a good bug bounty program involves submissions from a diverse crowd of security researchers. Borrowing to the formulation by Brady et al. of software security as a phenomenon of evolutionary pressure dictated by environmental changes, we shall propose that any additional security researcher involved in a bug bounty program brings a unique combination of skills and mindset, which is at least partially different from all other researchers. This unique perspective is comparable to a slightly changing environment for the software piece under scrutiny, which brings unique opportunities for each security researcher, regardless of the opportunity level of other researchers focused on the same software piece. Yet because people cannot easily change their skills and mindset, once the opportunity has been exploited, finding additional bugs gets much harder, hence pushing toward other opportunities, such as newly launched bug bounty programs.

The intrinsic limited capability by each single security researcher to find a large number of bugs for a specific program (i.e., on a specific software piece) carries a strong justification for the existence of bug bounty programs as a tool for engaging a large and diverse crowd of security researchers, beyond internalized software testing and security research teams. While we do not make any recommendation on the seniority and trustworthiness level of the security crowd targeted and engaged in a specific program, we may however discuss the *Uber* bug bounty program launched in 2016 [47]. This organization has designed its program as a way to select and hire a hand "security czars" from a larger crowd. Although there is certainly nothing wrong with hiring a security expert in an opportunistic manner, our results rather suggest that systematically using bug bounty programs for hiring purpose may reveal counter-productive on the long term. Security researchers will be likely to tune their expectations and behaviors toward getting a job. The approach implemented by Uber may reduce engagement of researchers who don't expect to get a job offer. Thus, it could limit the involvement of a larger crowd and its renewal, as more permanent security positions get filled.

The strategy followed by Uber for its bug bounty program is also interesting from a theoretical perspective: Uber is following a well-known strategy first described by Ronald Coase [43], which prescribes that if an organization has repeated interactions on the market with the same counterpart – or *ceteris paribus* a similar counterpart, then the organization is better-off internalizing the resource in order to avoid repeated transaction costs. Hence, Uber considers that security researchers as *substitute goods*, while our results, as well as the mere existence of bug bounty programs, rather demonstrate that security researchers are *complements*, at least partially. The distinction between substitutes and complements regarding security researchers, brings a fundamental justification for the existence and the future development of bug bounty programs as market places for trading bugs and vulnerabilities [10].

Yet, the organization designs of bug bounties programs and online platforms supporting them are still pretty much empirical. The validation step performed here provide additional theoretical insights. On the one hand, these insights shall be useful to formulate recommendations. On the other hand, they help identify blind spots, which will deserve future theoretical and empirical research efforts.

A. Recommendations

We propose 3 major recommendations that may significantly improve the efficiency of bug bounty programs, the platforms hosting them, as well as maximize the engagement of security researchers.

1. Encourage renewal, mobility and enrollment

Bug bounty programs shall encourage mobility by devoting resources to the recruitment of security researchers who were not previously involved in the program, rather than making effort to keep security researchers who have already well performed engaged. Mobility increases chances to find security researchers with genuine skills and mindset, who in turn will find additional bugs. Similarly, bug bounty platforms have the possibility to encourage

mobility across bug bounty programs. We have found that mobility across programs already exists, in particular mobility from old to newly launched programs. We also advocate the active recruitment of new security researchers by both bug bounty programs and platforms. For example, the bug bounty platform may feature older programs to security researchers who have recently enrolled on the platform.

2. Feature major changes for front-loading

The launch of a new bug bounty program is a unique moment that attracts a large number of security researchers. Yet front-loading may also be organized when a software piece received a major update with higher probability of finding bugs. Bug bounty program managers are encouraged to feature significant changes in the codebase, to help security researchers focus on issues they have not been exposed previously. Organizing engagement in order to match demand with supply is a token of fairness for security researchers who may appreciated to be guided to windows of high opportunity. This approach may also help the bug bounty program manager to steer the attention of security researchers toward more pressing security issues. Front-loading can also be organized with a temporary bug bounty reward increase. Additionally, dynamic adaptation of incentives can help manage contingencies associated with the management of submissions, for instance by reducing reward during a temporary overload.

3. Organize fluid legitimate markets

One overarching concern with trading bugs and vulnerabilities is the temptation by security researchers to consider selling their bugs on the black market. One way is to streamline transactions costs associated with bug submission and reward operations. Encouraging mobility without providing market fluidity indeed exposes to the risk that more bugs get submitted more often on the black market. Our proposition is reminiscent of the strategy followed by *Apple Inc*: By offering cheap enough online music, easy to download on *iTunes*, Apple Inc. was able to capture most of the online music black market, such on *Napster*. Bug bounty programs face similar challenges and opportunities to capture a market by reducing transaction costs and alleviating uncertainties associated with being involved on the black

market.

B. Importance

Our recommendations stem directly from the validation step performed. They show the importance of having a clear view of the theoretical and empirical underpinnings of the mechanisms of bug bounty programs and of platforms organizing them. Mobility, front-loading and fluidity may be organized either through top-down bureaucracy, or by setting market incentives right. The relative advantages bureaucracy and market organizations shall be further studied. Our recommendations for designing bug bounty programs, apply indistinctly to public and private bug bounties. Private bug bounty programs however face additional effort to encourage renewal, mobility and enrollment and front-loading, while at the same time curating the security researchers invited to participate in the private program.

C. Limitations

- limitations : (i) costs of searching vulnerabilities : for now it is embedded in the probability to find. (ii) no much knowledge about security researcher specialization (type of vulnerability + bug bounty programs), (iii) The nature of "skills and mindset" remains to be explored, (iv)

Evidence that security researchers specialize : The migration of researchers from one program to another is also related to researchers' specialization. Since vulnerability discovery is a difficult and also competitive activity, researchers typically need to have specialized knowledge and skills. Previous research revealed that there are at least two types of specialization in bug bounty: program-specific and vulnerability-specific [12, 13]. *Program-specific* specialization contains the knowledge, experience and skill of finding vulnerabilities in websites and software products under one particular program. Since specialization is relatively unique to the program, a specialized researcher has less incentive to switch. *Vulnerability-specific* specialization focuses on knowledge and skills of a particular type of vulnerability, which can exist in many different products. Therefore, for researchers with vulnerability-specific specialization, they have stronger incentive to try out their skills on different bug bounty programs. Incorporating researcher specialization into the models discussed in this work

would be an interesting future work.

Using *rankings* provides handy insights on the processes governing the vulnerability discovery process, and to some extent, associated incentives. However, the rank is an arbitrary measure of time, which hardly accounts for the effort spent on researching bugs, as well as for discounting effects. For instance, if the time required to find a vulnerability increases with the rank, then the expected payoff shall be discounted accordingly. Other aspects enter the equation: While most submissions occur early on after the program launch, this is also the moment when an organization might be less prepared to respond to a large flow of tasks, which in turn may trigger priority queueing and contingent delays [41]. While some workaround may be envisioned, publicly available data currently limit some desirable investigations, involving timing and discounting effects.

VII. CONCLUSION

In this paper, we have investigated how crowds of security researchers hunt software bugs and vulnerabilities and report to a bug bounty platform. Consistent with the famous adage "Given enough eyeballs, all bugs are shallow" by Eric Raymond [29], we have found that security researchers face challenging difficulties when trying to uncover large numbers of bugs in a same bounty program: The super-linear reward increase for each new bug found does not counterbalance the sharply decreasing probability of finding new bugs by the same person. This result is consistent with the theory proposed by Brady et al. on maximized entropy of bug discovery as an evolutionary process, following adaptation to changing environments [15]: Each security researcher tests software within an environment bounded by her skills and mindset. This result brings a fundamental justification for the existence of markets for bugs, beyond internalized security operations and research: Bug bounty programs offer a way to capitalize on these *environments* provided by the involvement of many security researchers. Yet difficulties for researchers to find large numbers of bugs in one bug bounty program bring incentives for mobility across programs. In particular, we find that the launch of new bug bounty programs has a negative effect on incumbent programs regarding bug submissions. We thus bring forth 3 organization design recommendations. First, mobility shall be encouraged across bug bounty programs and perhaps across bug bounty platforms. Second, one major duty of bug bounty platforms shall be to enroll large number of security researchers, and guide them towards low-hanging fruits, which for new researchers, may not necessarily be new bug bounty programs. Finally, bug bounty program manager shall feature significant changes in the codebase, such as major releases, to help security researchers focus on issues they have not been exposed previously. Such strategy shall help retain more long-term engagement, while not crowding out motivation in face of difficulty.

---

[1] Andy Greenberg, "Pentagon launches the feds' first 'bug bounty' for hackers," (2016), *http://www.wired.com/2016/03/pentagon-launches-feds-first-bug-bounty-hackers/*. Last accessed March 4th, 2016.

[2] EN Adams, "Optimizing preventive maintenance of software products," IBM Journal of Research and Development **28**, 2–14 (1984).

[3] James Surowiecki, The wisdom of crowds (Anchor, 2005).

[4] Arfon M Smith, Stuart Lynn, and Chris J Lintott, "An introduction to the zooniverse," in First AAAI Conference on Human Computation and Crowdsourcing (2013).

[5] Firas Khatib, Seth Cooper, Michael D Tyka, Kefan Xu, Ilya Makedon, Zoran Popović, David Baker, and Foldit Players, "Algorithm discovery by protein folding game players," Proceedings of the National Academy of Sciences **108**, 18949–18953 (2011).

[6] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford, "Captcha: Using hard ai problems for security," in Advances in Cryptology, EUROCRYPT 2003 (Springer, 2003) pp. 294–311.

[7] Timothy Gowers and Michael Nielsen, "Massively collaborative mathematics," Nature **461**, 879–881 (2009).

[8] Justin Cranshaw and Aniket Kittur, "The polymath project: lessons from a successful online collaboration in mathematics," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (ACM, 2011) pp. 1865–1874.

[9] Steven Levy, Hackers: Heroes of the Computer Revolution, 3rd ed., edited by O'Reilly (O'Reilly, 2010).

[10] Rainer Böhme, "A comparison of market approaches to software vulnerability disclosure," in Emerging Trends in Information and Communication Security (Springer, 2006) pp. 298–311.

[11] Matthew Finifter, Devdatta Akhawe, and David Wagner, "An empirical study of vulnerability rewards programs." in USENIX Security (2013).

[12] Mingyi Zhao, Jens Grossklags, and Kai Chen, "An exploratory study of white hat behaviors in a web vulnerability disclosure program," in Proceedings of the 2014 ACM Workshop on Security Information Workers (ACM, 2014) pp. 51–58.

[13] Mingyi Zhao, Jens Grossklags, and Peng Liu, "An empirical study of web vulnerability discovery ecosystems," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (ACM, 2015) pp. 1105–1117.

[14] Peter Bishop and Robin Bloomfield, "A conservative theory for long-term reliability-growth prediction [of software]," Reliability, IEEE Transactions on **45**, 550–560 (1996).

[15] Robert M Brady, Ross Anderson, and Robin C Ball, Murphy's law, the fitness of evolving species, and the limits of software reliability, 471 (University of Cambridge, Computer Laboratory, 1999).

[16] Mingyi Zhao and Peng Liu, "Empirical analysis and modeling of black-box mutational fuzzing," in International Symposium on Engineering Secure Software and Systems (ESSoS) (2016).

[17] Bev Littlewood and JL Verrall, "A bayesian reliability growth model for computer software," Applied Statistics , 332–346 (1973).

[18] B Littlewood and AJ Mayne, "Predicting software reliability [and discussion]," Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **327**, 513–527 (1989).

[19] Barton P Miller, Louis Fredriksen, and Bryan So, "An empirical study of the reliability of UNIX utilities," Communications of the ACM **33**, 32–44 (1990).

[20] Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, and David Brumley, "Enhancing symbolic execution with veritesting," in Proceedings of the 36th International Conference on Software Engineering (ACM, 2014) pp. 1083–1094.

[21] T Maillart, D Sornette, S Spaeth, and G Von Krogh, "Empirical tests of zipfs law mechanism in open source linux distribution," Physical Review Letters **101**, 218701 (2008).

[22] Alexander I Saichev, Yannick Malevergne, and Didier Sornette, Theory of Zipf's law and beyond, Vol. 632 (Springer Science & Business Media, 2009).

[23] Hanna Hulkko and Pekka Abrahamsson, "A multiple case study on the impact of pair programming on product quality," in Proceedings of the 27th International Conference on Software Engineering (ACM, 2005) pp. 495–504.

[24] Bryan Smith, William Yurcik, and David Doss, "Ethical hacking: the security justification redux," in International Symposium on Technology and Society (ISTAS'02) (IEEE, 2002) pp. 374–379.

[25] Syed A Saleem, "Ethical hacking as a risk management technique," in Proceedings of the 3rd annual conference on Information security curriculum development (ACM, 2006) pp. 201–203.

[26] Matt Bishop, "About penetration testing," Security & Privacy, IEEE **5**, 84–87 (2007).

[27] Ashish Arora, Rahul Telang, and Hao Xu, "Optimal policy for software vulnerability disclosure," Management Science **54**, 642–656 (2008).

[28] Hasan Cavusoglu and S Raghunathan, "Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge," IEEE Transactions on Software Engineering **33** (2007).

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

[30] Munawar Hafiz and Ming Fang, "Game of detections: how are security vulnerabilities discovered in the wild?" Empirical Software Engineering , 1–40 (2015).

[31] L Jean Camp and Catherine Wolfram, "Pricing security," in Economics of Information Security (Springer, 2004) pp. 17–34.

[32] Stuart Schechter, "How to buy better testing using competition to get the most security and robustness for your dollar," in Infrastructure Security (Springer, 2002) pp. 73–87.

[33] Karthik Kannan and Rahul Telang, "Market for software vulnerabilities? think again," Management Science **51**, 726–740 (2005).

[34] David McKinney, "Vulnerability bazaar," Security & Privacy, IEEE **5**, 69–73 (2007).

[35] Sam Ransbotham, Sabyasachi Mitra, and Jon Ramsey, "Are markets for vulnerabilities effective?" ICIS 2008 Proceedings , 24 (2008).

[36] A Algarni and Y Malaiya, "Software vulnerability markets: Discoverers and buyers," International Journal of Computer, Information Science and Engineering **8**, 71–81 (2014).

[37] Andy Ozment, "Bug auctions: Vulnerability markets reconsidered," in Third Workshop on the Economics of Information Security (2004) pp. 19–26.

[38] Paul R Milgrom and Robert J Weber, "A theory of auctions and competitive bidding," Econometrica: Journal of the Econometric Society , 1089–1122 (1982).

[39] Pankaj Pandey and Einar Arthur Snekkenes, "An assessment of market methods for information security risk management," in 16th IEEE International Conference on High Performance and Communications, WiP track (2014).

[40] Karim Lakhani and Robert Wolf, "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects." in Perspectives on Free and Open Source Software, edited by Joesph Feller, Brian Fitzgerald,

Scott Hissam, and Karim Lakhani (MIT Press, Cambridge, 2005).

[41] Thomas Maillart, Didier Sornette, Stefan Frei, Thomas Duebendorfer, and Alexander Saichev, "Quantification of deviations from rationality with heavy tails in human dynamics," Physical Review E **83**, 056101 (2011).

[42] Didier Sornette, Thomas Maillart, and Giacomo Ghezzi, "How much is the whole really more than the sum of its parts? 1? 1= 2.5: Superlinear productivity in collective group actions," Plos one **9**, e103023 (2014).

[43] Ronald H Coase, "The nature of the firm," Economica **4**, 386–405 (1937).

[44] Harry Kesten, "Random difference equations and renewal theory for products of random matrices," Acta Mathematica **131**, 207–248 (1973).

[45] Didier Sornette and Rama Cont, "Convergent multiplicative processes repelled from zero: power laws and truncated power laws," Journal de Physique I **7**, 431–444 (1997).

[46] Daniel Bernoulli, "Exposition of a new theory on the measurement of risk," Econometrica: Journal of the Econometric Society , 23–36 (1954).

[47] Kim Zetter, "Bug bounty guru katie moussouris will help hackers and companies play nice," (2016), *https://www.wired.com/2016/04/bug-bounty-guru-katie-moussouris-will-help-hackers-companies-play-nice/*. Last accessed May 9th, 2016.

[48] George Loewenstein and Drazen Prelec, "Anomalies in intertemporal choice: Evidence and an interpretation," The Quarterly Journal of Economics , 573–597 (1992).

[49] Aaron Clauset, Cosma R. Shalizi, and M. E. J. Newman, "Power-Law Distributions in Empirical Data," SIAM Review **51**, 661–703 (2009), arXiv:0706.1062.

[50] William N Goetzmann and Alok Kumar, "Equity portfolio diversification," Review of Finance **12**, 433–463 (2008).

[51] The first known bug bounty is the reward check program implemented by Donald Knuth in 1985 for debugging his book *TeX: The Program* (*http://truetex.com/knuthchk.htm*).

[52] Mingyi Zhao, Aron Laszka, Thomas Maillart, and Jens Grossklags, "Crowdsourced security vulnerability discovery: Modeling and organizing bug-bounty programs," in The HCOMP Workshop on Mathematical Foundations of Human Computation (2016).