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

Thomas Maillart[1], Mingyi Zhao[2], Jens Grossklags[2], and John Chuang[1]

[1] University of California, Berkeley
School of Information
102 South Hall
Berkeley, CA 94720
[2] The Pennsylvania State University
College of Information Science and Technology
329A Information Sciences and Technology Building
University Park, PA 16802

Abstract. 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. Little is known however on the incentives set by bug bounty programs: How they drive new bug discoveries, and how they supposedly improve security through the progressive exhaustion of discoverable vulnerabilities. Here, we recognize that bug bounty programs create tensions, for organizations running them on the one hand, and for security researchers on the other hand. At the level of one bug bounty program, security researchers face a sort of St-Petersburg paradox: The probability of finding additional bugs decays fast, and thus can hardly be matched with a sufficient increase of monetary rewards. Furthermore, bug bounty program managers have an incentive to gather the largest possible crowd to ensure a larger pool of expertise, which in turn increases competition among security researchers. As a result, we find that researchers have high incentives to switch to newly launched programs, for which a reserve of *low-hanging fruit* vulnerabilities is still available. Our results inform on the technical and economic mechanisms underlying the dynamics of bug bounty program contributions, and may in turn help improve the mechanism design of bug bounty programs that get increasingly adopted by cybersecurity savvy organizations.

1 Introduction

On March 2nd, 2016, the Pentagon announced the launch of its first *bug bounty* program [1]. From now on, the most paranoid organization in the United States will incentivize 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 *hard* computational problem [2], which most often requires additional human intelligence. However, given today's computer systems' complexity, individual human intelligence seems to have become insufficient, and organizations interested in drastically increasing their security are tempted to tap the wisdom of crowds [3], just like other disciplines have found ways to mobilize people at scale for their hard problems, such as for sorting galaxies in astronomy [4], folding proteins in biology [5], recognizing words from low quality book scans [6] or to address outstanding mathematics problems [7, 8].

All the above examples involve various aspects of human intelligence, ranging from pattern recognition (Captcha [6]) to highest abstraction levels (mathematical conjectures). It is not clear what kind of intelligence is required to find bugs and vulnerabilities in software, but it generally requires a high level of programming proficiency coupled with *hacking* skills to think out of the box and find unconventional and thus, unintended use for a software. In a nutshell, searching for complicated bugs and vulnerabilities may be a hard and time-consuming task, which is generally not, or at least no longer, considered as a leisure that hackers perform for hedonic pleasure or for the good.

Therefore, nowadays some (monetary) incentives must be set, in order to get security researchers to hunt bugs. Offering rewards for vulnerabilities has been a long endeavor over the last decade [9], with many more or less successful attempts to set incentives right [10, 11, 12]. HackerOne, a leading online service dedicated to helping organizations set up and manage their own bug bounty program, has paved the way to the deployment of bounty programs at scale. Nevertheless, in this pioneering era of bug bounty hunting, it remains unclear how current mechanism designs and incentive structures influence the long-term success of bounty programs. Better understanding of bug discovery mechanisms on the one hand [13], and on the other hand, better characterization of the utility functions of respectively (i) organizations operating a bug bounty program and (ii) security researchers, will help understand how bug bounty programs may evolve in the foreseeable future.

Here, we have investigated a public data set of 35 public bug bounty programs from the HackerOne website. We find that as more vulnerabilities get discovered within a bounty program, security researchers face an increasingly difficult environment in which the probability of finding a bug decreases fast, while reward increases. For this reason, as well as because the probability to find a bug decreases faster compared to the payoff increase, security researchers are incentivized to consistently switch to newly launched research programs, at the expense of older programs. This switching phenomenon has already been found in [12]. Here, we characterize it further, by quantifying the evolution of incentives as more vulnerabilities get discovered in bug bounty program, and how researchers benefit on the long term from switching to newly launched programs.

This article is organized as follows. Related research is presented in Section 2. Important features of the data set used here is detailed in Section 3. We then introduce the main mechanism driving vulnerability discovery in Section 4. Results are presented and discussed in respectively Sections 5 and 6. We finally in conclude in Section 7.

2 Related work

Achieving software reliability has concerned engineers for at least 4 decades [2,14,15]. Early empirical work on software bug discovery dates back to the time of UNIX systems [16], and over years, numbers of models for discovering vulnerabilities have been developed (see [13,17] for some of the most contemporary approaches). However, 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 [18,19], which diverges as the random variable sample increases (i.e., no statistical moment is defined), and thus, it was rightly concluded that there would be software vulnerabilities as long as enough resources and time could be provided to find them. This problem can also be seen from an entropy maximization perspective, which is good for evolution (e.g., in biology) but detrimental in software engineering. Concretely, as explained in [20], given the evolutionary nature of software, new bugs can be found in a software program as long as use perspectives change. The difficulty of bug hunting is therefore not about finding a bug *per se*, but rather about envisioning all possible use situations, which would 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 get rid of bugs at a speed that would allow following the general pace of software evolution and expansion. Thus, human intelligence is still considered as one of the most efficient ways to explore novel situations – 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 [21]. To protect against cyber-criminals, it is also fashionable to hire *ethical hackers*, who have a mindset similar to potential attackers, in order to probe the security of computer systems [22,23,24]. Inherited from the hacking and open source philosophies, the full disclosure policy has been hotly debated as promoting a safer Internet, by forcing software editors to recognize vulnerabilities discovered by independent researchers, and quickly fix them, as a result of publication on public forums [25]. The full-disclosure model has evolved into responsible disclosure, a standard practice in which the security researcher agrees to allow a period of time for the vulnerability to be patched before publishing the details of the flaw uncovered. In most of these successful human-driven approaches, there is a knowledge-sharing component, may it be between two programmers sitting together in front of a screen, ethical hackers being hired to discover and explore the weaknesses of a computer system, or the broader community being exposed to open source code and publicly disclosed software vulnerabilities. Thus, Eric Raymond's famous quote "Given enough eyeballs, all bugs are shallow" [26], tends to hold, even though in practice things are often slightly more complicated [27].

Recognizing the need of human intelligence for tackling security bugs at scale, researchers have considered early on the importance of trading bugs and vulnerabilities as a valuable knowledge, often earned the hard way. *Vulnerability markets* have thus emerged as a way to ensure appropriate incentives for knowledge transfer from security researchers to software and Internet organizations [28], and in particular, to jointly harness the wisdom of crowds and reveal the security level of organizations through a competitive incentive scheme [29]. The efficiency of vulnerability markets has however been nevertheless questioned on both theoretical [30,31] and empirical grounds [32,33].

Early on and building on previous work by Schechter [29], Andy Ozment [34] recognized that in theory most efficient mechanism designs shall not be markets *per se*, but rather auction systems [35]. 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 vulnerability 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 might submit the same vulnerability), but not too early, so that they can maximize their payoff $R(t) = R_0 + \epsilon \cdot t$ with ϵ the linear growth factor, which is also supposed to compensate for the increasing difficulty of finding each new bug. But setting the right incentive structure $\{R_0, \epsilon\}$ is not trivial, because it must account for uncertainties [36], such as work needed, or effective competition (i.e., the number of researchers enrolled in the bug program). Furthermore, the probability of overlap between 2 submissions by different researchers has remained largely unknown.

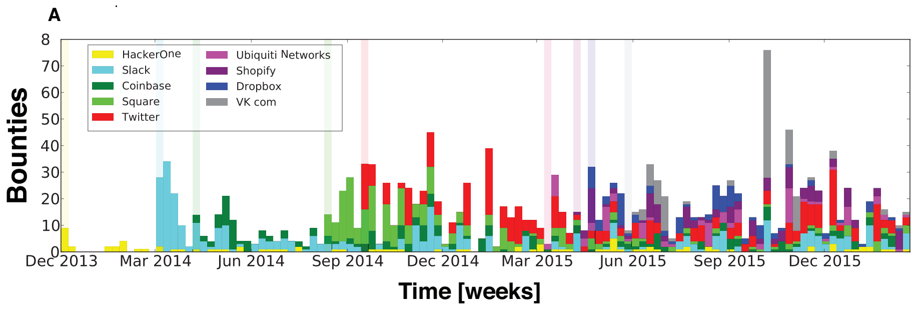
Regardless of theoretical considerations (or perhaps by integrating them), bug bounty programs have emerged as a tool used by the industry, first launched by specific software companies for their own needs and with rather heterogeneous incentive schemes [10], including with no monetary reward [11], and followed by dedicated platforms comparable to trusted third parties in charge of clearing transactions between bug bounty programs launched by organizations and security researchers. These platforms also assist organizations in the design and deployment of their own program. The currently leading platform is HackerOne.[3] HackerOne runs 35 public programs, for organizations across a wide range of business sectors, and for which bounty awards are reported on their website (in addition to a non-disclosed amount of private programs). Previous research has investigated vulnerability trends, response & resolve behaviors, as well as reward structures of participating organizations. In particular, it was found that a considerable number of organizations exhibit decreasing trends for reported vulnerabilities, yet monetary incentives have a significantly positive correlation with the number of vulnerabilities reported [12].

[3] HackerOne, *https://hackerone.com/* (last access March, 4th 2016).

3 Data

The data were collected from the public part of the Hacker One 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 (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, suggesting a *windfall* effect. 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, to sudden change of media exposure or even to fundamental differences of program comparative fitness, which we do not cover here.

The long-memory process of bug discovery following the launch of a bounty program we observe here, is reminiscent of human timing effects: 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 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 [37]. 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 sub-



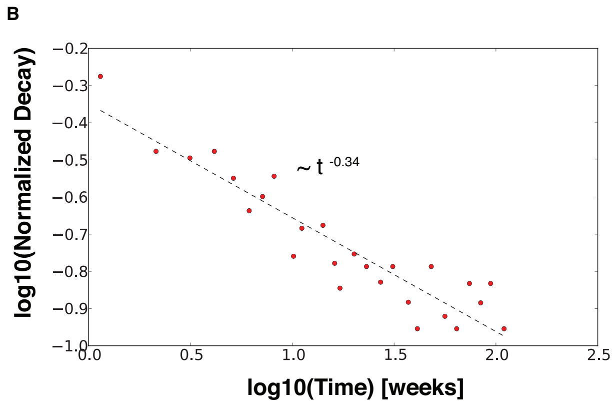


Fig. 1. A. Weekly vulnerability discoveries for the 9 most active programs (with at least 90 bug discoveries as of February 15, 2016). The light colored vertical bars represent the start of the program, occurring when the first bounty is awarded. Most programs exhibit an initial shock, followed by a 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.

missions are made, but don't lead to a discovery associated with a monetary reward.

4 Method

Bug bounty programs work on the premise that humans are efficient at searching and finding vulnerabilities, in particular when large pools of security researchers with a variety of skills can be mobilized for the task. It is in the interest of the organization launching a bounty program to *exhaust* vulnerabilities, or to reduce the probability of finding additional vulnerabilities to a residual level. In addition, incentives must be carefully set. Here, we investigate the interplay between the vulnerability exhaustion process, and the cumulative reward distributed to security researchers within and across bounty programs. When a bug bounty program starts, it attracts a number of security researchers, who in turn submit bugs. Subsequent bug discoveries get increasingly difficult [20], and program managers must reward vulnerabilities accordingly in order to keep security researchers onboard (or to attract new ones according to the current level of difficulty).

Starting from an initial probability of discovering the first vulnerability $P(k = 0) = 1$, we assume that the probability to find a second (and subsequent) vulnerability(ies), is a fraction of the former probability: $P_{k+1} = \beta * P_k$ with β a constant strictly smaller than, yet usually close to 1. 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 ($R_n = \Lambda_n R_{n-1} + R_0$) [38,39]: 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 [40]. 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 2 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 2 if a tail appears on the first toss, 4 if a head appears on the first toss and a tail on the second, 8 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 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, $U_n = \sum_{k=0}^{n} U_k = n$) [40]. For bug bounty programs, the situation is slightly similar because the main question a security researcher may ask herself when enrolling a bug bounty program is the amount of initial effort (i.e., the upfront learning costs) to be devoted in order to make a positive expected net payoff. The payoff conditioned by the expected number of bug discoveries and their associated monetary rewards minus the cost. 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, for the sake of simplicity, we assume that they are integrated in $P(k)$. The security researcher may also decide to stop searching for bugs in a program, at any time. 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 tune Λ. However, in general rules are set upfront and shall not be changed in the course of the bounty program. Changing game rules is risky as it may undermine trust in the program. Here, we assume the bounty program managers don't tune their reward policy after the bug bounty program has started. 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 [41]. 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.

5 Results

The vulnerability 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 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.

5.1 Security researcher enrollment determines the success of a bug bounty program

As presented on Figure 2A, we find that the number B_c of vulnerabilities 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 vulnerability discoveries (i.e., *daughter* events) by the security researcher. Here, each cascade stems from a single security

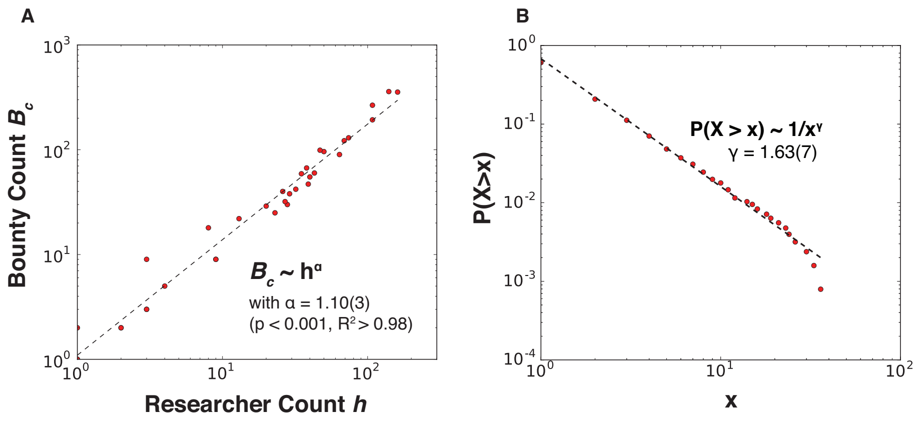


Fig. 2. A. The number of bounty discoveries per program B_c scales as h^α with $\alpha = 1.10(3)$ and h the number of security researchers enrolled in a program. Since $\alpha > 1$, a bounty programs benefits in a super-linear fashion to the enrollment of more researchers. **B.** The tail distribution of bounty discoveries per researcher per program follows a power law distribution $P(X > x) \sim 1/x^\gamma$ with $1 < \gamma = 1.63(7) < 2$. 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.

researcher (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 distribution is therefore heavy-tailed, but not extreme, as the first statistical moment (i.e., the mean) is 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. According to our data and analysis, there is no bug bounty "czar" who could scoop a dominant portion of bugs. These results have somewhat counterintuitive organization design implications that we discuss later on.

5.2 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 vulnerabilities and their associated payoff, as discussed in Section 4. 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

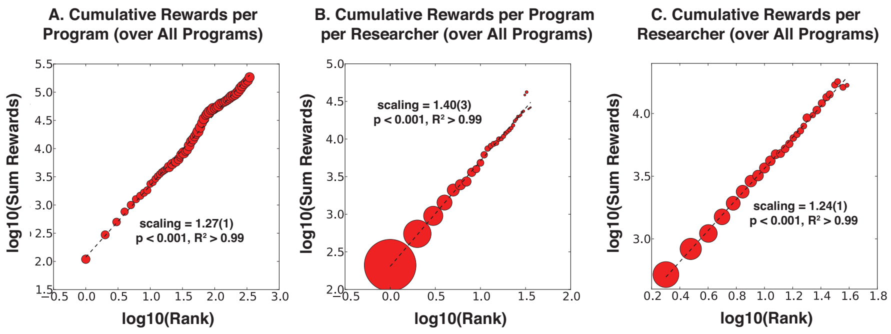


Fig. 3. A. (Log-)binned cumulative down-payment per program over all public programs on the HackerOne platform, scales as $R_k \sim k^{1.27}$ ($p < 0.001$, $R^2 > 0.99$), with k the rank. Each log-bin shows the mean value and the circle sizes depict the number of values in each bin (i.e., the rank frequency). 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 $P_k \sim k^{-1.05}$ ($p < 0.001$, $R^2 = 0.85$).

viewpoint of a 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 $\sim k^{1.27}$ ($p < 0.001$, $R^2 > 0.99$), while the frequency of vulnerabilities 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 viewpoint of the researcher 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 $\sim k^{-0.45}$, 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 $R_k \sim k^{\alpha}$ with $\alpha > 1.40$, which in turn would worsen the down-payment function both in terms of incremental expenditures and in exploration of higher ranks. This approach does not consider possible individual human hard limits, preventing finding additional bugs. In that latter case, setting higher reward incentives would have no effect.

Security researchers tend to switch from one bounty program to another program [11, 12]. The strategy can be assimilated to portfolio diversification [43]. 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.

5.3 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 switching decision may be made by a researcher 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. 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. Here, 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 start with a simple ordinary least square (OLS) regression model, which is specified 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. Following previous work [12], we consider two control variables that could influence researcher's decision. 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.

Table 1. 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

The regression results are shown in Table 1. Consistent with our prediction, 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 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 above results, as researchers have a high incentive to find the low rank low-hanging fruit bugs, even if the reward is small.

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.

Our model specification also allows us to show that the reputation of a bug bounty (A_{it}) has a very strong effect on overall bug submissions. Coupled with the power law decay of bug submission observed following an initial shock (c.f., Figure 1B), one may observe that the overall amount of bugs found over time directly follows bug submissions made right after the program was launched.

6 Discussion

Finding bugs in software code is one of the oldest and toughest problems in software engineering. While algorithm based approaches have been developed over the years, human verification has remained a prime way to debugging and vulnerability hunting. Moreover, the idea of getting "enough eyeballs" to inspect the code has been a cornerstone argument for the open source software movement [26] along with the full-disclosure argument, as well as for vulnerability markets [9]. Bug bounty programs are perhaps the latest successful incarnation of markets for trading bugs and vulnerabilities [9], which set incentives to disclose early, combined with an increase of payoff for more rare and difficult bugs.

Here, we have found 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 skewed, but not extreme. In other words, each researcher enrolled in a bug bounty program may contribute her fair share of valid bugs, and no researcher is found to contribute orders of magnitude more than the average. This result is rather surprising, as the common wisdom at the moment seems to be rather to use bug bounty programs for selecting most talented security researchers [44]. In some way, there is a conceptual flaw in this common wisdom reasoning: If selecting one (resp. a few) particularly talented security researcher(s), then the Coase theorem would apply [45] and it would be more interesting for an organization to internalize the resource by hiring security consultants, or having a in-house auditing team, both of which are already commonly done by organizations. And because bug bounty programs exist and develop, it *proves* that the former security practices are probably insufficient. On the contrary to this common wisdom, we posit that bug bounty programs reach a large and diverse population of security researchers, who can independently look at the focal software from as many different perspectives.

This observation is reminiscent of an early proposition on the topic: Brady et al. [20] took an evolutionary theory perspective to the problem of software reliability, and basically said that software is sensitive to environmental changes (i.e., it is not evolutionary fit) because it is usually designed for one purpose. The purpose however changes over time (think e.g., of software packages in Linux, how they are surprisingly linked together [18], and how they are used in unintended ways). On the contrary to species who adapt by the way of selection (only the fittest portion of the population survives), this feature is essentially absent in software, according to Brady et al. Here, the focal point is a software piece, or more precisely a set of complementary software pieces, which define the service offered by the focal organization. Software fitness is assessed by security researchers, internally (internal audit), externally (ethical hackers), or by resorting to the crowd (bug bounty programs). The software runs in a well-defined environment, and it would be hard, if not impossible, to deploy it in other environments (i.e., for a different use, e.g., by a different population), in order to test its robustness. Note that some very large companies by a matter of fact extensively test their software in a variety of environments given the pervasive nature of their service. One may think of Facebook with more 1.5 billion users worldwide.

Because they all carry their own unique experience, security researchers offer a form of confrontation with alternative environments. Additionally, the more remote from the focal organization, the more original the view on the software piece (without the hassle of deployment). Our results offer a similar conceptual view, as well as with the quote by Eric Raymond "Given enough eyeballs, all bugs are shallow". In sum, diversity of views prevails over accumulated expertise, although we make no claim that expertise is not required. We just observe that its individual effects are just bounded. These results also cast questions on the learning curves, and incentives to keep digging bugs in a program, which the researcher is already familiar with. We observe that overall these incentives become quickly insufficient in comparison with the increasing difficulty for a researcher to find additional bugs.

Moreover, we find that the larger the population of enrolled researchers, the even more bugs are found. In that process, the initial windfall effect of a newly launched program is critical and determines an important portion of the bug discovery timeline, and accordingly researchers are ready to switch their attention towards newly launched programs, at the expense of older ones. These results have critical implications for software security: If we consider an arbitrary focal bug to be discovered, the chance that it will be discovered increases with the number of researchers. If half researchers interested in the security of the focal software are black hats, there is roughly 50% chance that the focal bug will be discovered by a black hat. If the proportion of population types (white and black hats) compounded over time is uneven, then the probability of discovery falling in one of both categories changes accordingly. In other words, in order to be effective (statistically speaking), a bug bounty program must reach much more white hats, compared to the estimated amount of black hats interested in finding holes in the focal software piece.

Most security researchers however participate in multiple bug bounty programs, and when a new program is launched, they face the strategic choice of switching program. Our results show that researchers have a decreasing incentive to explore higher ranks within the same program (Figure 3B), while they have an increasing, yet marginally decreasing, incentive to explore multiple programs (Figure 3C). We further confirm these results with a simple regression model that researchers tend to switch when new programs are launched. This is a strong signal that researchers make rational choices in the bug hunting environment: They have high incentives to switch quickly to a new program and harvest as fast as possible many frequent bugs with little reward, rather than less frequent yet more endowed bugs, even though the reward structure of bug bounty programs seems to incentivize generously the reward of high rank (i.e., less probable) vulnerabilities (Figure 3A). This result raises questions on some hard limits associated with incentives associated with bug discovery by humans: The disincentive clearly stems from the difficulty for individual researchers to reach high ranks (i.e., $P_k \to 0$ when $k \to \infty$), not from the reward scheme. While at first sight this turnover of security researchers may look bad to a bug bounty program manager, it may in the end be beneficial provided that a renewal of security researchers is provided. The bug bounty platform must be designed carefully to ensure that a sufficient inflow of new security researchers are enrolled and scattered among all programs hosted on the platform, including older ones, in order to compensate for researchers switching to new programs.

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 [37]. While some workaround may be envisioned, publicly available data currently limit some desirable investigations, involving timing and discounting effects.

In this study, we have considered the incentive mechanisms at the aggregate level. Managers however organize enrollment, set incentives and tackle the operational pipeline, involving submission reviews and payroll processing. All these aspects, which are unique to each program, may crowd in, or on the contrary crowd out, security researchers from bug bounty programs. In particular their willingness to participate will be affected, but also the amount of effort they are ready to throw in the search of vulnerabilities. It is the hope of the authors to get increasingly fine-grained insights in the future, to compare bug bounty programs, and thus establish benchmarks of most performing programs, in an environment driven by large deviation statistics, and incentives structures, which resemble the St-Petersburg paradox, a well-known puzzle for decision making in behavioral economics.

7 Conclusion

In this paper, we have investigated how crowds of security researchers hunt software bugs and vulnerabilities on the public part of a bug bounty platform. We have found that it is essential for managers to design their program in order to attract and enroll the largest possible number of security researchers. Studying the incentive structure of 35 public bug bounty programs launched at a rate of one per month over 2 years, we have found that security researchers have high incentives to rush to newly launched programs, in order to scoop rewards from numerous easy bugs, and as the program ages (and therefore, the probability of finding a vulnerability decreases), switch to newer "easier" programs. Our results suggest that yet incentives are generously set by program managers, it is quickly getting harder for a researcher to find vulnerabilities, once she has discovered the obvious ones. This windfall effect is positive as it allows security researchers provide their unique perspective in many bug bounty programs. However, the loss of researchers by older bug bounty programs should be compensated with new security researchers in order to ensure renewal of perspectives.

Acknowledgements This research was supported in part by the National Science Foundation through award CCF-0424422 (TRUST - Team for Research in Ubiquitous Secure Technology). One of the authors (TM) acknowledges support from the Swiss National Science Foundation (SNSF; Grants PA00P2_145368 and P300P2_158462). The authors would like to thank Aron Laszka for his valuable comments, as well as the 3 anonymous reviewers who provided insightful comments and thus contributed to a considerable improvement of the manuscript.

References

1. Greenberg, A.: Pentagon launches the feds first bug bounty for hackers `http://www.wired.com/2016/03/pentagon-launches-feds-first-bug-bounty-hackers/`. Last accessed March 4th, 2016.
2. Adams, E.: Optimizing preventive maintenance of software products. IBM Journal of Research and Development **28**(1) (1984) 2–14
3. Surowiecki, J.: The wisdom of crowds. Anchor (2005)

4. Smith, A.M., Lynn, S., Lintott, C.J.: An introduction to the zooniverse. In: First AAAI Conference on Human Computation and Crowdsourcing. (2013)
5. Khatib, F., Cooper, S., Tyka, M.D., Xu, K., Makedon, I., Popović, Z., Baker, D., Players, F.: Algorithm discovery by protein folding game players. Proceedings of the National Academy of Sciences **108**(47) (2011) 18949–18953
6. Von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: Captcha: Using hard ai problems for security. In: Advances in Cryptology, EUROCRYPT 2003. Springer (2003) 294–311
7. Gowers, T., Nielsen, M.: Massively collaborative mathematics. Nature **461**(7266) (2009) 879–881
8. Cranshaw, J., Kittur, A.: 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) 1865–1874
9. Böhme, R.: A comparison of market approaches to software vulnerability disclosure. In: Emerging Trends in Information and Communication Security. Springer (2006) 298–311
10. Finifter, M., Akhawe, D., Wagner, D.: An empirical study of vulnerability rewards programs. In: USENIX Security. (2013)
11. Zhao, M., Grossklags, J., Chen, K.: 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) 51–58
12. Zhao, M., Grossklags, J., Liu, P.: An empirical study of web vulnerability discovery ecosystems. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM (2015) 1105–1117
13. Zhao, M., Liu, P.: Empirical analysis and modeling of black-box mutational fuzzing. In: International Symposium on Engineering Secure Software and Systems (ESSoS). (2016)
14. Littlewood, B., Verrall, J.: A bayesian reliability growth model for computer software. Applied Statistics (1973) 332–346
15. Littlewood, B., Mayne, A.: Predicting software reliability [and discussion]. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **327**(1596) (1989) 513–527
16. Miller, B.P., Fredriksen, L., So, B.: An empirical study of the reliability of UNIX utilities. Communications of the ACM **33**(12) (1990) 32–44
17. Avgerinos, T., Rebert, A., Cha, S.K., Brumley, D.: Enhancing symbolic execution with veritesting. In: Proceedings of the 36th International Conference on Software Engineering, ACM (2014) 1083–1094
18. Maillart, T., Sornette, D., Spaeth, S., Von Krogh, G.: Empirical tests of zipfs law mechanism in open source linux distribution. Physical Review Letters **101**(21) (2008) 218701
19. Saichev, A.I., Malevergne, Y., Sornette, D.: Theory of Zipf's law and beyond. Volume 632. Springer Science & Business Media (2009)
20. Brady, R.M., Anderson, R., Ball, R.C.: Murphy's law, the fitness of evolving species, and the limits of software reliability. Number 471. University of Cambridge, Computer Laboratory (1999)
21. Hulkko, H., Abrahamsson, P.: 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) 495–504
22. Smith, B., Yurcik, W., Doss, D.: Ethical hacking: the security justification redux. In: International Symposium on Technology and Society (ISTAS'02), IEEE (2002) 374–379
23. Saleem, S.A.: Ethical hacking as a risk management technique. In: Proceedings of the 3rd annual conference on Information security curriculum development, ACM (2006) 201–203
24. Bishop, M.: About penetration testing. Security & Privacy, IEEE **5**(6) (2007) 84–87
25. Arora, A., Telang, R., Xu, H.: Optimal policy for software vulnerability disclosure. Management Science **54**(4) (2008) 642–656

26. Raymond, E.: The cathedral and the bazaar. Knowledge, Technology & Policy **12**(3) (1999) 23–49
27. Hafiz, M., Fang, M.: Game of detections: how are security vulnerabilities discovered in the wild? Empirical Software Engineering (2015) 1–40
28. Camp, L.J., Wolfram, C.: Pricing security. In: Economics of information security. Springer (2004) 17–34
29. Schechter, S.: How to buy better testing using competition to get the most security and robustness for your dollar. In: Infrastructure Security. Springer (2002) 73–87
30. Kannan, K., Telang, R.: Market for software vulnerabilities? think again. Management Science **51**(5) (2005) 726–740
31. McKinney, D.: Vulnerability bazaar. Security & Privacy, IEEE **5**(6) (2007) 69–73
32. Ransbotham, S., Mitra, S., Ramsey, J.: Are markets for vulnerabilities effective? ICIS 2008 Proceedings (2008) 24
33. Algarni, A., Malaiya, Y.: Software vulnerability markets: Discoverers and buyers. International Journal of Computer, Information Science and Engineering **8**(3) (2014) 71–81
34. Ozment, A.: Bug auctions: Vulnerability markets reconsidered. In: Third Workshop on the Economics of Information Security. (2004) 19–26
35. Milgrom, P.R., Weber, R.J.: A theory of auctions and competitive bidding. Econometrica: Journal of the Econometric Society (1982) 1089–1122
36. Pandey, P., Snekkenes, E.A.: An assessment of market methods for information security risk management. In: 16th IEEE International Conference on High Performance and Communications, WiP track. (2014)
37. Maillart, T., Sornette, D., Frei, S., Duebendorfer, T., Saichev, A.: Quantification of deviations from rationality with heavy tails in human dynamics. Physical Review E **83**(5) (2011) 056101
38. Kesten, H.: Random difference equations and renewal theory for products of random matrices. Acta Mathematica **131**(1) (1973) 207–248
39. Sornette, D., Cont, R.: Convergent multiplicative processes repelled from zero: power laws and truncated power laws. Journal de Physique I **7**(3) (1997) 431–444
40. Bernoulli, D.: Exposition of a new theory on the measurement of risk. Econometrica: Journal of the Econometric Society (1954) 23–36
41. Loewenstein, G., Prelec, D.: Anomalies in intertemporal choice: Evidence and an interpretation. The Quarterly Journal of Economics (1992) 573–597
42. Sornette, D., Maillart, T., Ghezzi, G.: 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**(8) (2014) e103023
43. Goetzmann, W.N., Kumar, A.: Equity portfolio diversification. Review of Finance **12**(3) (2008) 433–463
44. Zetter, K.: Bug bounty guru katie moussouris will help hackers and companies play nice `https://www.wired.com/2016/04/bug-bounty-guru-katie-moussouris-will-help-hackers-companies-play-nice/`. Last accessed May 9th, 2016.
45. Coase, R.H.: The Nature of the Firm. Economica **4**(16) (November 1937) 386–405

Appendix

Derivations of the Model for homogenous factors $\Lambda_k = \Lambda$ Here, we provide a detailed study of the possible behaviors of the model, also around the critical point $\Lambda = 1$.

Three regimes must be considered:

1. For $\Lambda < 1$, the distribution is given by

$$P_{\Lambda<1}(S \geq s) = (1-\beta)\left(1 - \frac{s}{s_{\max}}\right)^c , \quad s_{\max} := \frac{S_0\Lambda}{1-\Lambda} , \quad c := \frac{\ln\beta}{\ln\Lambda} > 0 \, . \tag{4}$$

This distribution can be approximated in its central part, away from the maximum possible reward $s_{\max}$, by a Weibull distribution of the form

$$\Pr(\mathrm{S} \geq s) \sim e^{-(s/d)^c} \, . \tag{5}$$

For $\Lambda \to 1^-$, we have $s_{\max} \to +\infty$ and, for $s \ll s_{\max}$, expression (4) simplifies into a simple exponential function

$$P_{\Lambda\to 1^-}(S \geq s) \sim e^{-|\ln(\beta)|s/S_0} \, . \tag{6}$$

2. For $\Lambda = 1$, the distribution of rewards is a simple exponential function since $S_n = nS_0$ is linear in the rank n and the probability of reaching rank n is the exponential $P(n) = \beta^n(1-\beta)$. Actually, the expression (7) becomes asymptotical exact as

$$P_{\Lambda=1}(S \geq s) = (1-\beta)e^{-|\ln(\beta)|s/S_0} \, . \tag{7}$$

3. For $\Lambda > 1$, the distribution of rewards is of the form,

$$P_{\Lambda>1}(S \geq s) = \frac{1}{(1+\frac{s}{s*})^c} , \quad s^* := \frac{S_0\Lambda}{\Lambda - 1} , \quad c := \frac{|\ln\beta|}{\ln\Lambda} , \tag{8}$$

which develops to a power law distribution of reward of the form $\Pr(\text{reward} \geq S) = C/S^\mu$ with $\mu = c$, when $\Lambda \to +\infty$.