Crowdsourced Security Vulnerability Discovery: Modeling and Organizing Bug-Bounty Programs

Mingyi Zhao[1], Aron Laszka[2], Thomas Maillart[2], Jens Grossklags[1]

[1]Pennsylvania State University, [2]University of California, Berkeley

1 Introduction

Despite significant progress in software-engineering practices, software utilized for desktop and mobile computing remains insecure. At the same time, the consumer and business information handled by these programs is growing in its richness and monetization potential, which triggers significant privacy and security concerns. In response to these challenges, companies are increasingly harvesting the potential of external (ethical) security researchers through bug bounty programs to crowdsource efforts to find and ameliorate security vulnerabilities [5,10]. These so-called white hat hackers are often rewarded with monetary bounties and public recognition.

Broadening the appeal of crowdsourced security, several commercial bug bounty platforms have emerged (e.g., HackerOne, BugCrowd, Cobalt) and successfully facilitate the process of building and maintaining bug bounty programs for organizations. For example, on HackerOne, more than 20,000 security vulnerabilities have been reported and fixed for hundreds of organizations. Contributions came from over 2500 different white hat hackers who received bounties of over $7.3M as of May 2016.

Over the last two years, we have begun to systematically study these platforms from an empirical perspective to evidence their growing popularity and practical contributions to the security of deployed code [10,9]. While empirical results imply that bug bounty programs make a significant contribution to security, there also exist several obstacles for running and scaling bug bounty programs. One challenge is to reduce the number of invalid (or low quality) submissions from the crowd. To address this challenge, we have built an economic model for bug bounties and analyzed multiple existing "crowd quality control" policies [8]. We also proposed a new policy and showed the advantages of this new policy over existing ones.

Another challenge of running bug bounty programs is to efficiently allocate valuable but scarce hacker effort over time, and across organizations with different crowdsourcing requirements. In addition, in contrast to many crowdsourcing scenarios, bug discovery requires sophisticated participants, who are partially competing with each other. The competition often leads to multiple hackers discovering the same bug. One bug bounty platform, BugCrowd, has reported that *30% to 40% of the submissions are duplicates* [2]. However, of all duplicates only the first report is rewarded. Therefore, an efficient allocation shall decrease the amount of duplicated effort, while expanding and also diversifying the manpower. We think that addressing this challenge, like other human computation problems [3], requires rigorous mathematical modeling, in order to quantify the strength and limitations of bug bounties and to design more efficient mechanisms. In this paper, we present our ongoing research on modeling and optimizing bug bounty programs.

2 The Bug Bounty Model

Bug bounty utilities. We assume that there are unknown bugs[1] waiting to be discovered in a software product (e.g., a website). The organization responsible for this product creates a bug bounty program.

[1] Usually, bug bounty programs only focus on security bugs, or vulnerabilities. For brevity, we will use the more general word "bug." In addition, we will assume that all bugs have equal impact. This assumption will be relaxed in future work.

It then invites participants from a pool of hackers H at the start of each time step t ($t = 1, 2, \ldots$). We define an *allocation plan* as a vector $A = \{H_1, H_2, \ldots\}$, where $H_t \subseteq H$ is the set of hackers invited for time step t. At the end of t, invited hackers submit r_t bug reports in total to the program. Among these reports, some are duplicates because multiple hackers could find the same issue. On the other hand, the organization is only interested in unique discoveries, whose number is denoted by u_t, and $u_t \leq r_t$. r_t and u_t are calculated by the bug discovery model to be discussed shortly. The organization rewards each unique bug discovery with bounty b (e.g., average around $424 in 2015 [10]), and fixes all discovered bugs at the end of each time step. The organization also incurs cost c_o for processing a submitted report. We also assume that the organization gains V value by fixing a bug. However, the value decreases over time since it is more likely that malicious parties will find and exploit the bug. We use $\delta \in (0, 1)$ to model this time discount. We can write the utility function of an organization from bug bounty as

$$U_o = \sum_{t=1}^{\infty}((\delta^{t-1}V - b)u_t - c_o r_t). \tag{1}$$

Similarly, we assume that it costs c_h for a hacker to find and submit a bug. So the utility function of all invited hackers is

$$U_h = \sum_{t=1}^{\infty}(bu_t - c_h r_t). \tag{2}$$

Bug discovery model. To calculate the expected utilities, we first need to establish a bug discovery model. A bug can be represented as a single or a group of inputs that triggers a specific error in the software or hardware system. Since the input space of any non-trivial system is prohibitively large, a hacker usually discovers bugs based on tools with randomization (e.g., fuzzing), heuristics, experiences, and luck. We propose a bug discovery model. We assume that, for bug i, each invited hacker discovers it with probability p_i independently in one time step, as long as bug i has not been discovered in previous time steps. Probability p_i not only models the randomness in bug discovery, but also captures the difficulty of discovering a bug. Previous research has shown that bugs have different discovery difficulty [1,11]. Particularly, we inferred that in practice p_i follows a discrete power law distribution [11]:

$$p_i = \frac{i^{-\alpha}}{\zeta(\alpha)}, \tag{3}$$

where α is the scaling factor and ζ is the Riemann Zeta function. α reflects the size of the system's attack surface and the system's security quality, and can potentially be estimated from factors like codebase size, maturity of the security development life cycle, etc. Also, we implicitly sort all bugs in descending order of their discovery probability, so bug 1 is the easiest to be found. In addition, a bug belongs to one type from a set of vulnerability types denoted by S [10]. We assume that the probability of a bug being of type s is q_s, where q_s is exogenous and known, and obviously $\sum_{s \in S} q_s = 1$. q_s can be estimated from earlier bug discovery data, obtained through internal security testing or from similar organizations.

Hacker diversity. Existing literature has revealed that hackers have diverse expertise, use different tools, etc., so they are good at discovering different types of bugs [4,10,6,7]. We let S_h be the set of vulnerability types that hacker $h \in H$ can discover, so the probability that hacker h discovers bug i is p_i if $s \in S_h$, where s is the type of bug i, and it is 0 if $s \notin S_h$. S_h can be obtained from data accumulated on bug bounty platforms. Figure 1 illustrates the vulnerability discovery model.

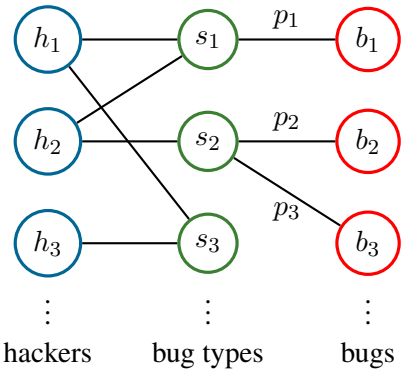

Fig. 1. Illustration of the vulnerability discovery model.

3 Preliminary Results and Discussion

Expected utilities. We calculate expected utilities of the organization and hackers as follows. First, we define $H_{t,s}$ as the set of hackers allocated for time t with the expertise to find bugs of type s. In other words, $H_{t,s} = \{h \in H_t | s \in S_h\}$. We then define $ND_{i,t} := (1 - p_i)^{|H_{t,s}|}$ as the probability that none of the hackers in H_t discover bug i at t. For the organization, the expected utility $\mathrm{E}[U_{oi}|\text{bug } i \text{ is of type } s]$ for discovering bug i of type s, and the total expected utility $\mathrm{E}[U_o]$ from a bug bounty program given an allocation plan are, respectively:

$$\mathrm{E}[U_{oi}|\text{bug } i \text{ is of type } s] = \sum_{t=1}^{\infty} \left(\prod_{k=1}^{t-1} ND_{i,k} \right) \left((\delta^{t-1} V - b) \left(1 - ND_{i,t}\right) - c_o \left(|H_{t,s}|p_i\right) \right) \tag{4}$$

$$\mathrm{E}[U_o] = \sum_{i=1}^{\infty} \sum_{s \in S} q_s \, \mathrm{E}[U_{oi}|\text{bug } i \text{ is of type } s]. \tag{5}$$

Similarly, for all hackers, we have

$$\mathrm{E}[U_{hi}|\text{bug } i \text{ is of type } s] = \sum_{t=1}^{\infty} \left(\prod_{k=1}^{t-1} ND_{i,k} \right) \left(b \left(1 - ND_{i,t}\right) - c_h \left(|H_{t,s}|p_i\right) \right) \tag{6}$$

$$\mathrm{E}[U_h] = \sum_{i=1}^{\infty} \sum_{s \in S} q_s \, \mathrm{E}[U_{hi}|\text{bug } i \text{ is of type } s]. \tag{7}$$

Bug bounty optimization. The practical goal of our work is to help organizations optimize their bug bounty programs. A basic, but often voiced idea is to attract as many hackers as possible. We evaluate this notion using our model and data collected from one bug bounty platform, i.e., Wooyun [10]. Figure 2 shows that the expected utilities of the inviting organization and the invited hackers exhibit inverted U-shapes, and do not scale linearly with the number of hackers. Rather, they start to decrease after a certain number of hackers have joined. The reason is that as more hackers are invited, the number of duplicates increases, which raises the cost of processing reports by the organization, and also decreases the expected bounty received by hackers. This result suggests that, for bug bounty programs and possibly for some other crowdsourcing scenarios that require expertise and competition, *more participation is not always better*. Instead, the bug bounty program shall carefully design its allocation plan to control the competition among participants and to diversify its workforce. In addition, the bug bounty program also needs to offer enough reward for a bug, such that the expected utility of hackers is greater than zero, even as discovering bugs is getting harder over time. We are defining this problem as $\max_{A,b:\mathrm{E}[U_h] \geq 0} \mathrm{E}[U_o]$, which is subject of our ongoing work.

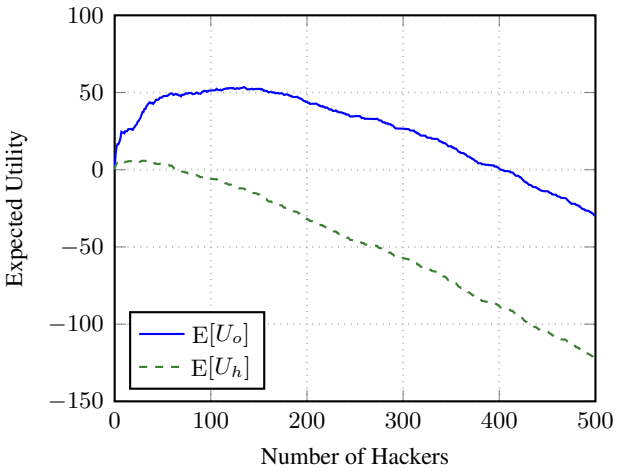

Fig. 2. Expected utilities with different number of hackers. Parameters used: $V = 20$, $b = 5$, $\delta = 0.99$, $c_o = c_h = 1$, $\alpha = 2$.

Acknowledgments

This work was supported in part by the National Science Foundation (CNS-1238959) and by the Air Force Research Laboratory (FA 8750-14-2-0180).

References

1. Brady, R., Anderson, R., Ball, R.: Murphy's law, the fitness of evolving species, and the limits of software reliability. Tech. Rep. 471, University of Cambridge, Computer Laboratory (1999)
2. Bugcrowd: The state of bug bounty (July 2015)
3. Chen, Y., Ghosh, A., Kearns, M., Roughgarden, T., Vaughan, J.W.: Mathematical foundations for social computing. Communications of ACM (2016)
4. Edmundson, A., Holtkamp, B., Rivera, E., Finifter, M., Mettler, A., Wagner, D.: An empirical study on the effectiveness of security code review. In: 5th International Conference on Engineering Secure Software and Systems (ESSoS), pp. 197–212. Springer (2013)
5. Finifter, M., Akhawe, D., Wagner, D.: An empirical study of vulnerability rewards programs. In: 22nd USENIX Security Symposium. pp. 273–288 (2013)
6. Hafiz, M., Fang, M.: Game of detections: How are security vulnerabilities discovered in the wild? Empirical Software Engineering pp. 1–40 (2015)
7. Huang, K., Siegel, M., Madnick, S., Li, X., Feng, Z.: Poster: Diversity or concentration? Hackers' strategy for working across multiple bug bounty programs. In: 37th IEEE Symposium on Security and Privacy (S&P) (2016)
8. Laszka, A., Zhao, M., Grossklags, J.: Banishing misaligned incentives for validating reports in bug-bounty platforms. In: 21st European Symposium on Research in Computer Security (ESORICS). Springer (2016)
9. Maillart, T., Zhao, M., Grossklags, J., Chuang, J.: Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty markets. In: 15th Annual Workshop on the Economics of Information Security (WEIS) (2016)
10. Zhao, M., Grossklags, J., Liu, P.: An empirical study of web vulnerability discovery ecosystems. In: 22nd ACM Conference on Computer and Communications Security (CCS). pp. 1105–1117. ACM (2015)
11. Zhao, M., Liu, P.: Empirical analysis and modeling of black-box mutational fuzzing. In: 8th International Symposium on Engineering Secure Software and Systems (ESSoS). pp. 173–189. Springer (2016)