Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes

Daniel Votipka, Rock Stevens, Elissa M. Redmiles, Jeremy Hu, and Michelle L. Mazurek
Department of Computer Science
University of Maryalnd
College Park, Maryland 20742
Email: {dvotipka,rstevens,eredmiles,jhu,mmazurek}@cs.umd.edu

***Abstract*—Identifying security vulnerabilities in software is a critical task that requires significant human effort. Currently, vulnerability discovery is often the responsibility of software testers before release and white-hat hackers (often within bug bounty programs) afterward. This arrangement can be ad-hoc and far from ideal; for example, if testers could identify more vulnerabilities, software would be more secure at release time. Thus far, however, the processes used by each group — and how they compare to and interact with each other — have not been well studied. This paper takes a first step toward better understanding, and eventually improving, this ecosystem: we report on a semi-structured interview study (n=25) with both testers and hackers, focusing on how each group finds vulnerabilities, how they develop their skills, and the challenges they face. The results suggest that hackers and testers follow similar processes, but get different results due largely to differing experiences and therefore different underlying knowledge of security concepts. Based on these results, we provide recommendations to support improved security training for testers, better communication between hackers and developers, and smarter bug bounty policies to motivate hacker participation.**

I. Introduction

Software security bugs, also known as vulnerabilities, continue to be an important and expensive problem. There has been significant research effort toward preventing vulnerabilities from occurring in the first place, as well as toward automatically discovering vulnerabilities, but so far these results remain fairly limited: Human intelligence is often required to supplement automated tools, and will continue to be needed in the foreseeable future [1]–[9]. For now, the job of finding vulnerabilities prior to release is often assigned to software testers who typically aim to root out all bugs — performance, functionality, and security — prior to release. Unfortunately, general software testers do not typically have the training or the expertise necessary to find all security bugs, and thus many are released into the wild [10].

Consequently, expert freelancers known as "white-hat hackers" examine released software for vulnerabilities that they can submit to bug bounty programs, often aiming to develop sufficient credibility and skills to be contracted directly by companies for their expertise [11], [12]. Bug bounty programs offer "bounties" (e.g., money, swag, or recognition) to anyone who identifies a vulnerability and discloses it to the vendor. By tapping into the wide population of white-hat hackers, companies have seen significant benefits to product security, including higher numbers of vulnerabilities found and improvements in the expertise of in-house software testers and developers as they learn from the vulnerabilities reported by others [12]–[17].

This vulnerability-finding ecosystem has important benefits, but overall it remains fairly ad-hoc, and there is significant room for improvement. Discovering more vulnerabilities prior to release would save time, money, and company reputation; protect product users; and avoid the long, slow process of patch adoption [18]–[23]. Bug bounty markets, which are typically dominated by a few highly-active participants [13]–[16], lack cognitive diversity[1], which is specifically important to thoroughly vet software for security bugs [3], [12]. Bug bounty programs can also exhibit communication problems that lead to low signal-to-noise ratios [17]. Evidence suggests that simply raising bounty prices is not sufficient to address these issues [25], [26].

To improve this overall ecosystem, therefore, we must better understand how it works. Several researchers have considered the economic and security impact of bug bounty programs [16], [27]–[30]; however, little research has investigated the human processes of benign vulnerability finding. In this work, we take a first step toward improving this understanding. We performed 25 semi-structured interviews with software testers and white-hat hackers (collectively, *practitioners*), focusing on the process of finding vulnerabilities in software: why they choose specific software to study, what tools they use, how they develop the necessary skills, and how they communicate with other relevant actors (e.g., developers and peers).

We found that both testers and hackers describe a similar set of steps for discovering vulnerabilities. Their success in each step, however, depends on their vulnerability discovery experience, their knowledge of underlying systems, available access to the development process, and what motivates them to search for vulnerabilities.

Of these variables, practitioners report that experience — which differs greatly between testers and hackers – is most significant to success in vulnerability finding. Differences in experience stem primarily from the fact that hackers are typically exposed to a wider variety of vulnerabilities through a broad array of sources including employment, hacking

[1]The way people think and the perspectives and previous experiences they bring to bear on a problem [24, pg. 40-65].

exercises, communication with peers, and prior vulnerability reports. On the other hand, we find that testers are typically exposed to only a narrow set of vulnerabilities through fewer sources, as testers primarily search for vulnerabilities in only a single code base, only read bug reports associated with that program, and only participate in small, internal hacking exercises, if any.

Access to the development process and motivation also differ notably between hackers and testers. While participants report that more experience is always better, their opinions on access and motivation are less straightforward: more access can help or hinder vulnerability finding depending on circumstances, and the relationship between motivation and success can be highly variable.

From these findings, we distill recommendations to improve human-driven vulnerability discovery for both populations.

II. Related work

In this section, we review prior work in four key areas.

A. Bug identification process

Previous work has studied how different populations perform the task of bug identification. Aranda et al. studied how developers and testers found 10 performance, security, and functionality bugs in a production environment [31]. They reviewed all reporting artifacts associated with the bugs and interviewed the developers and testers who found and fixed them. They found that bugs were most commonly discovered through manual testing; close cooperation and verbal communication were key to helping developers fix bugs.

Fang et al. surveyed hackers who disclosed vulnerabilities in the SecurityFocus repository, asking how participants choose software to investigate, what tools they use, and how they report their findings [32], [33]. They found that hackers typically targeted software they were familiar with as users, predominantly preferred fuzzing tools to static analysis, and preferred full disclosure. Summers et al. studied problem-solving mental models through semi-structured interviews of 18 hackers [34]. They find that hackers require a high tolerance for ambiguity, because they seek to find problems that may or may not exist in a system they did not design. Additionally, Summers et al. observed that hackers rely on discourse with others or visualization techniques (i.e., mapping system semantics on a white-board) to deal with ambiguity and identify the most probable issues.

We expand on these prior studies by comparing white-hat hackers and testers specifically in the domain of security and including testers and hackers from multiple companies and bug bounty programs. Also, we thoroughly investigate participants' processes, communication about vulnerabilities and reporting strategies, skill-development, and reasons for using specific tools.

B. Tester and hacker characteristics

Lethbridge et al. discuss the wide breadth of software testers' backgrounds, estimating that only 40% possess a computing-related education and a majority lack formal training in software engineering practices [35]. They recommend expanding interactive educational opportunities for testers to support closing gaps in essential knowledge. Relatedly, Bertolino et al. examined how testers can harness their domain-specific knowledge in a distributed fashion to find more bugs more quickly [36]. We expand on this previous work to provide the first exploration of how software testers currently learn and expand their knowledge of vulnerability discovery practices.

Al-Banna et al. focus on external security professionals, asking both professionals and those who hire them which indicators they believed were the most important to discern security expertise [37]. Similarly, Cowley interviewed 10 malware reverse engineering professionals to understand the necessary skills and define levels of professional development [38]. We borrow the concept of task analysis from this work to guide our interviews while expanding the scope of questions and comparing hackers to software testers.

Criminology research has also examined why some individuals who find vulnerabilities become cyber criminals, finding that although most hackers work alone, they improve knowledge and skills in part through mentoring by peers [11], [39], [40]. While we explicitly do not consider black-hat hackers, we build on these findings with further analysis of how hackers learn skills and create communities.

C. Measurement of bug bounty programs

Several researchers have investigated what factors (e.g., money, probability of success) most influence participation and productivity in bug bounty programs. Finifter et al. studied the Firefox and Chrome bug bounty programs [16]. They found that a variable payment structure based on the criticality of the vulnerability led to higher participation rates and a greater diversity of vulnerabilities discovered as more researchers participated in the program. Maillart et al. studied 35 public HackerOne bounty programs, finding that hackers tend to focus on new bounty programs and that a significant portion of vulnerabilities are found shortly after the program starts [12]. The authors suggest that hackers are motivated to find "low-hanging fruit" (i.e., easy to discover vulnerabilities) as quickly as possible, because the expected value of many small payouts is perceived to be greater than for complex, high-reward vulnerabilities that might be "scooped" by a competitor.

While these studies suggest potential motivations for hacker behavior based on observed trends, we directly interview hackers about their motivations. Additionally, these studies do not explore the full decision process of bug bounty participants. This exploration is important because any effective change to the market needs to consider all the nuances of participant decisions if it hopes to be successful. Additionally, prior work does not compare hackers with software testers. This comparison is necessary, as it suggests ways to best train and allocate resources to all stakeholders in the software development lifecycle.

D. Other studies with developers and security professionals

Similarly to our work, many researchers have investigated the specific needs and practices of developers and other security experts in order to understand how to improve application and code security [41]. For example, researchers have focused on understanding how and why developers write (in)secure software [42]–[51] and have investigated the usability of static analysis tools for vulnerability discovery [3], [52]–[60], network defense and incident response [61]–[69], malware analysis [70], and corporate security policy development and adherence [71]–[78]. While these works investigate different topics and questions than the work presented here, they highlight the benefits of the approach taken in our research: studying how experts approach security.

III. Methodology

To understand the vulnerability discovery processes used by our target populations, we conducted semi-structured interviews with software testers and white-hat hackers (henceforth *hackers* for simplicity) between April and May 2017. To support rigorous qualitative results, we conducted interviews until new themes stopped emerging (25 participants) [79, pg. 113-115]. Because we interview more than the 12-20 participants suggested by qualitative research best practices literature, our work can provide strong direction for future quantitative work and generalizable design recommendations [80].

Below, we describe our recruitment process, the development and pre-testing of our interview protocol, our data analysis procedures, and the limitations of our work. This study was approved by our university's Institutional Review Board (IRB).

A. Recruitment

Because software testers and hackers are difficult to recruit [31]–[33], we used three sources to find participants: software testing and vulnerability discovery organizations, public bug bounty data, and personal contacts.

Related organizations. To recruit hackers, we contacted the leadership of two popular bug bounty platforms and several top-ranked Capture-the-Flag (CTF) teams. We gathered CTF team contact information when it was made publicly available on CTFTime.org [82], a website that hosts information about CTF teams and competitions.

To reach software testers, we contacted the most popular Meetup [83] groups with "Software Testing" listed in their description, all the IEEE chapters in our geographical region, and two popular professional testing organizations: the Association for Software Testing [84] and the Ministry of Testing [85].

Public bug bounty data. We also collected publicly available contact information for hackers from bug bounty websites. One of the most popular bug bounty platforms, HackerOne [86], maintains profile pages for each of its members which commonly include the hacker's contact information. Additionally, the Chromium [87] and Firefox [88] public bug trackers provide the email addresses of anyone who has submitted a bug report. To identify reporters who successfully submitted vulnerabilities, we followed the process outlined by Finifter et. al. by searching for specific security-relevant labels [16].

Personal contacts. We asked colleagues in related industries to recruit their co-workers. We also used snowball sampling (asking participants to recruit peers) at the end of the recruitment phase to ensure we had sufficient participation. This recruitment source accounts for three participants.

Advertisement considerations. We found that hackers were highly privacy-sensitive, and testers were generally concerned with protecting their companies' intellectual property, complicating recruiting. To mitigate this, we carefully designed our recruiting advertisements and materials to emphasize the legitimacy of our research institution and to provide reassurance that participant information would be kept confidential and that we would not ask for sensitive details.

Participant screening. Due to the specialized nature of the studied populations, we asked all volunteers to complete a 20-question survey to confirm they had the necessary skills and experience. The survey assessed participants' background in vulnerability discovery (e.g., number of security bugs discovered, percent of income from vulnerability discovery, programs they have participated in, types of vulnerabilities found) and their technical skills (e.g., development experience, reverse engineering, system administration). It also concluded with basic demographic questions. We drew these questions from similar surveys distributed by popular bug bounty platforms [13], [15]. We provide the full set of survey questions in Appendix A.

We selected participants to represent a broad range of vulnerability discovery experience, software specializations (i.e., mobile, web, host), and technical skills. When survey responses matched in these categories, we selected randomly. To determine the participants' software specialization, we asked them to indicate the percent of vulnerabilities they discovered in each type of software. We deem the software type with the highest reported percentage the participant's speciality. If no software type exceeded 40% of all vulnerabilities found, we consider the participant a generalist (i.e., they do not specialize in any particular software type).

B. Interview protocol

We performed semi-structured, video teleconference[2] interviews, which took between 40 and 75 minutes. All interviews were conducted by a single interviewer. Using a semi-structured protocol, the interviewer focused primarily on the set of questions given in Appendix B, with the option to ask follow-ups or skip questions that were already answered [89]. Each interview was divided along three lines of questioning: general experience, task analysis, and skill development.

Prior to the main study, we conducted four pilot interviews (two testers, two hackers) to pre-test the questions and ensure

[2]Interviews were conducted via video teleconference because it was geographically infeasible to meet face-to-face.

validity. We iteratively updated our protocol following these interviews, until we reached the final protocol detailed below.

General experience. We began the interviews by asking participants to expand on their screening-survey responses regarding vulnerability discovery experience. Specifically, we asked their motivation behind doing this type of work (e.g. altruism, fun, curiosity, money) and why they focus (or do not focus) on a specific type of vulnerability or software.

Task analysis. Next, we asked participants what steps they take to find vulnerabilities. Specifically, we focused on the following sub-tasks of vulnerability discovery:

- **Program selection.** How do they decide which pieces of software to investigate?
- **Vulnerability search.** What steps are taken to search for vulnerabilities?
- **Reporting.** How do they report discovered vulnerabilities? What information do they include in their reports?

To induce in-depth responses, we had participants perform a hierarchical task analysis focused on these three sub-tasks. Hierarchical task analysis is a process of systematically identifying a task's goals and operations and decomposing them into sub-goals and sub-operations [90]. Each operation is defined by its goal, the set of inputs which conditionally activate it, a set of actions, and the feedback or output that determine when the operation is complete and which follow-on operations are required. Hierarchical task analysis was developed to analyze complex, non-repetitive, cognitively loaded tasks to identify errors or inefficiencies in the process. We adopted this for our study, as it provides a useful framework for eliciting details from experts who typically perform some parts of tasks automatically and subconsciously [90].

For each sub-operation identified, we also asked participants to discuss any specific tools they use, what skills are useful to complete this step, how they learned and developed their process for completing the necessary actions, and how the steps they take differ across software and vulnerability types.

Skill development. Finally, we asked participants to describe how they developed the skills necessary to find vulnerabilities. Here, we focused on their learning process and how they interact with other members of their respective communities.

During the task analysis portion of the interview, we asked participants to explain how they learned how to complete certain tasks. In this segment, we broadened this line of questioning and asked what development steps they recommend to newcomers to the field. This question was intended to elicit additional learning sources that may have been missed previously and highlight steps participants believe are the most important.

Finally, we asked each participant to describe their interactions with other members of their local community and the vulnerability discovery and software tester community at large. Specifically, we discussed who they interact with, the forms of their interaction (e.g., one-to-one, large groups), how these interactions are carried out (e.g., conferences, online forums, direct messaging), and what types of information are discussed.

C. Data analysis

The interviews were analyzed using iterative open coding [91, pg. 101-122]. When all the interviews were completed, four members of the research team transcribed 10 interviews. The remaining 15 interviews were transcribed by an external transcription service. The interviewer and another researcher independently coded each interview, building the codebook incrementally and re-coding previously coded interviews. This process was repeated until all interviews were coded. The codes of the two interviewers were then compared to determine inter-coder reliability using the ReCal2 software package [92]. We use Krippendorff's Alpha (α) to measure inter-coder reliability as it accounts for chance agreements [93].

The α after coding all the interviews was .68. Krippendorff recommends using α values between .667 and .80 only in studies "where tentative conclusions are still acceptable" [94]; and other work has suggested a higher minimum threshold of .70 for exploratory studies [95]. To achieve more conclusive results, we recoded the 16 of our 85 codes with an α less than .70. For each code, the coders discussed a subset of the disagreements, adjusted code definitions as necessary to clarify inclusion/exclusion conditions, and re-coded all the interviews with the updated codebook. After re-coding, the α for the study was .85. Additionally, all individual codes' αs were above .70.

Next, we grouped the identified codes into related categories. In total, there were six categories describing the participants' discovery process (i.e., Information Gathering, Program Understanding, Attack Surface, Exploration, Vulnerability Recognition, and Reporting) and four categories regarding factors that influenced participants' implementation of this process (i.e., Vulnerability Discovery Experience, Underlying System Knowledge, Access to the Development Process, and Motivation). We then performed an axial coding to find connections between categories and between codes within categories [91, pg. 123-142]. Based on the categories and connections between them, we derive a theory describing the process practitioners use to find vulnerabilities, the factors that influence their implementation of this process, and how testers and hackers differ with respect to their process and implementation.

D. Limitations

Our study has several limitations common to exploratory qualitative research. A lack of complete recall is especially prominent in studies like ours, where participants are asked to describe expert tasks [90]. We employ a hierarchical task analysis in our interview protocol to improve the thoroughness of information elicited. Participants may have also adjusted their answers to portray themselves as more or less skilled, if they were concerned with the interviewer's perception of them [96], [97]. Additionally, there could be selection bias among the testers and hackers studied. Because we explicitly stated the purpose of the study when recruiting, it is possible that those with experience or an interest in security were more likely to respond to our request. Also, since some hackers tend to be more privacy sensitive, some may have decided not

to participate in order to protect their identity or intellectual property. To partially mitigate these issues, we recruited through a wide variety of sources and interviewed a diverse pool of participants to increase the likelihood that relevant ideas would be stated by at least one participant. Finally, for each finding, we give the number of testers and hackers that expressed a concept, to indicate prevalence. However, if a participant did not mention a specific idea, that does not necessarily indicate disagreement; they may have simply failed to state it. For these reasons, we do not use statistical hypothesis tests for comparison among participants. Our results do not necessarily generalize beyond our sample; however, they suggest many directions for future work and provide novel insights into the human factors of software vulnerability discovery.

IV. Participants

We received 49 responses to our screening survey. We selected 10 testers and 15 hackers. (Themes related to their vulnerability discovery process converged more quickly with testers than with hackers, so we required fewer interviews [79]). Table I shows our participants' demographics, including their self-reported vulnerability-discovery skill level (on a scale from 0-5, with 0 indicating no skill and 5 indicating an expert), self-reported number of vulnerabilities they have discovered, company size (only applicable for testers), and the method used to recruit them.

Hacker demographics match prior surveys. Our study demographics are relatively congruent with hacker demographics reported in studies from the popular bug bounty services HackerOne [13] and BugCrowd [98]. 90% of HackerOne's 70,000 users were younger than 34; 60% of BugCrowd's 38,000 users are 18-29 and 34% are 30-44 years old. Our hacker population was 60% under 30 and 90% under 40 years old. Regarding education, 84% of BugCrowd hackers have attended college and 21% have a graduate degree; 93% of our hackers have attended college and 33% have a graduate degree.

Testers are more diverse than hackers. In contrast to our hacker population, none of our software testers were under 30 and only 60% were under 40 years old. All of our testers have some college education, but only one has a graduate degree. With respect to ethnicity and gender, the software tester group was much more diverse, at 60% male and 60% Caucasian.

Hackers reported higher vulnerability discovery skills. As expected, there is a contrast in vulnerability finding skills between testers and hackers. The hacker population self-reported an average skill level of 3.5, whereas software testers self-reported an average skill of 2.5. This self-reported measure cannot be used to directly compare participants' abilities; instead, it indicates their self-efficacy, telling us that testers tend to be less confident in their ability to find security bugs.

Interestingly, despite the hacker population possessing more vulnerability finding experience, more software testers self-reported having discovered more than 500 vulnerabilities. However, we note that the number of vulnerabilities is not necessarily representative of participant skill. It may instead depend on their specialization. For example, participants who focused on web applications reported finding more vulnerabilities, but these are generally considered less complex and therefore are less profitable in bug bounties [99].

ID[1,2]	Gender: Age:Race[3]	Educ.	Skill	Vulns. Fnd	Org. Sz	Source[4]
T1W	M:30-39:H	B.S	1	0-3	>20K	O
T2W	F:40-49:W	B.S.	2	0-3	100	O
T3W	F:30-39:W	B.S	3	26-50	150	O
T4G	M:30-39:A	SC	5	>500	200-10K	O
T5W	M:30-39:W	B.S.	4	>500	200	O
T6W	M:50-59:A	B.S.	3	51-100	60K	O
T7G	M:30-39:A	B.S.	4	>500	50	O
T8H	F:50-59:W	Assoc.	0	101-500	2K	O
T9W	F:30-39:W	B.S.	0	0-3	2K	C
T10W	M:40-49:W	M.S.	3	0-3	10-50K	O
H1H	M:18-29:W	B.S.	4	11-25	-	O
H2H	M:30-39:W	B.S.	4	51-100	-	O
H3G	M:30-39:W	M.S.	5	>500	-	O
H4H	F:18-29:W	M.S.	4	26-50	-	O
H5M	M:18-29:W	B.S.	4	101-500	-	O
H6G	M:18-29:H	M.S.	3	101-500	-	O
H7W	M:18-29:W	M.S.	3	26-50	-	O
H8M	M:30-39:W	SC	5	101-500	-	C
H9G	M:18-29:W	H.S.	4	26-50	-	O
H10H	M:18-29:W	SC	2	11-25	-	O
H11W	M:18-29:W	B.S.	4	51-100	-	O
H12W	M:40-49:B	B.S.	1	11-25	-	O
H13W	M:30-39:W	B.S.	4	>500	-	C
H14W	M:30-39:W	M.S.	4	101-500	-	P
H15W	M:18-29:W	B.S.	2	26-50	-	P

[1] IDs are coded by population (T: Tester, H: Hacker) in date order
[2] Software Specialization – W: Web, H: Host, M: Mobile, G: General
[3] W: White, B: Black, A: Asian, H: Hispanic
[4] Recruitment method – O: Related Organization, P: Public Bug Bounty Data, C: Personal Contact

TABLE I: Participant demographics.

V. Vulnerability discovery process

Perhaps our most surprising result is that hackers and testers described a similar exploratory process for vulnerability finding. They first focus on learning what the program does, then use their intuition and experience to find ways to perform unintended, malicious actions. Across participants, this process was generally broken into five phases: *Information Gathering*, *Program Understanding*, *Attack Surface*, *Exploration*, *Vulnerability Recognition*, and *Reporting*. Participants described the second (Program Understanding) through fourth (Exploration) phases as a loop that they iterate until a vulnerability is found. Figure 1 shows the process our participants described, as well as the factors that influence the process. In all the category graphs in this paper, we represent process categories as hexagons, influencing categories as rectangles, and items that determine the influencing categories as ovals. Additionally, we represent relationships between categories with arrows whose direction indicates the direction of influence. For readability, we color arrows from influencers to process categories to indicate the influence category they are derived from.

In this section, we briefly outline the overall vulnerability discovery process. In the following section, we discuss in detail

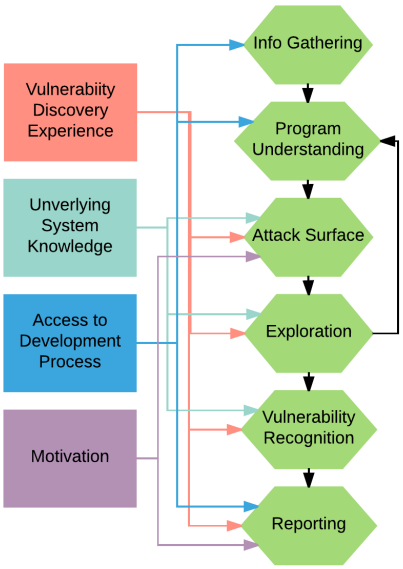

Fig. 1: Vulnerability-finding process and influencing factors.

the factors which influence the execution of each phase, and which exhibit greater differences between our two populations.

Information gathering. In the initial phase of vulnerability discovery, practitioners quickly collect preliminary information about the program to develop context prior to reading any code or executing the program (T=6, H=14). This includes finding any previous bugs reported in the program (T=3, H=9), determining the age of the code and update history (i.e., looking at change logs to identify old or recently updated code segments) (T=5, H=9), and identifying the libraries used (T=1, H=2). This phase's goal is to develop an understanding of prior efforts, as well as the technologies the program is built on. Information Gathering is also used specifically by some hackers to decide whether to expend additional effort or move on to a different target (T=0, H=3).

Program understanding. Next, our participants try to determine how the program operates, how it interacts with its environment (i.e., the user, network, etc.), and how individual components interact with each other. Initially, this step is based on communication with developers (T=7, H=0) or on reading documentation (T=5, H=5), when available. Through iterations of the loop (i.e., Program Understanding, Attack Surface, and Exploration), as they execute the program and read code, practitioners learn about the program by its behavior (T=6, H=11). T4G described iteratively building up an idea of the program structure by "touching a little bit everything, and then you are organizing that structure in your head…[and] you can formalize it [with the code]." H9G tries to get into the developer's mindset by rewriting the code himself. He starts by thinking about the possible "inputs from the management or business side that [go] into the specification," then he writes a version of the program himself and "look[s] for matches between the machine code I'm seeing [the original program] and the machine code that my C++ program produces."

Attack surface. Our participants then identify how a user can interact with the program (T=9, H=15). Their goal is to determine what an attacker can manipulate and how they can influence program execution. This allows our participants to focus only on critical components of the program. H4H explains "I look at things that I can touch, [for example] what can I get to from the network. …That's necessary to narrow down the target[s]." Our participants discussed looking for direct program inputs (T=9, H=10), such as website form fields, as well as indirect inputs (T=4, H=10), such as data pulled from their social media page or backend communication to a server.

Exploration. Next, practitioners explore the effect of a range of inputs to see whether it is possible to perform some malicious action by providing data the program mishandles. H5M described this as "enumerating all the variable[s] and all the parameters…I can quickly make a bunch of accounts and see how the user ID changes and how it associates one user with multiple devices." Our participants described a range of approaches to exploring program behavior, typically a combination of executing the program with a set of test inputs (T=9, H=11) and code inspection (T=6, H=12).

Of the tools mentioned during our interviews, almost all were used in this phase. Our participants reported preferring tools that automate simple, repetitive tasks so that they can focus on more complicated problems (T=4, H=13). Such tasks include quickly searching through code or network captures (T=2, H=10), providing suggestions for test cases (T=5, H=3), or making code easier to read (e.g., callee-caller method cross-referencing, variable renaming) (T=1, H=6). We found that hackers were much more likely to utilize tools to automate this phase of the process, preferring dynamic analyses (e.g., fuzzing) (T=5, H=12) over static analyses (e.g., symbolic execution) (T=0, H=2), which matches Hafiz and Fang's findings [32].

On the other hand, seven hackers mentioned specifically focusing on doing things manually, or using tools that aided them in doing so, because they feel this gives them a competitive advantage (T=0, H=7). For example, H15W says he avoids fully automated tools because "I assume that [large companies] already run static and dynamic analysis tools…so there's not much of a point of me doing it."

Vulnerability recognition. Participants iterate through the prior three phases until they eventually identify a vulnerability. This phase can be as simple as seeing a crash that produces a known bad behavior or getting a tool output that indicates a problem. However, in most cases our participants described relying on their intuition and system knowledge to recognize where an assumption is violated or a simple crash shows a bigger security problem (T=6, H=14).

Reporting. Finally, once the vulnerability is found, it must be reported. In their reports, our participants focus on making sure the information is presented in a way that is easily understandable by developers (T=8, H=11). Specifically, they stressed communicating the importance of fixing the vulnera-

bility (T=10, H=12). T4G explained that even after you find a vulnerability, "you have to have the weight of someone else to agree that [it] is a bug…you do have to [convince] someone that there's a risk…It's quite timely [time consuming], running a ticket." This emphasis on selling the importance of the vulnerability mirrors the findings of Haney and Lutters [100].

Exception to the process. Four hackers in our study reported a notable exception to the order of phases described above. These hackers stated that they, in some cases, first recognize a vulnerability and reverse the normal process by looking for an execution path to the insecure code (T=0, H=4). This occurs whenever they find known insecure code (e.g., *memcpy* or *printf* in a C program) using a string search or other simple static analysis. Then they trace the execution path back through the code manually to find any input that triggers the vulnerable code. While this is a different order of operations, the general phases of the process remain the same.

VI. Influencing Factors

While all our participants described a similar process, their implementation of this process differed. These differences can be loosely grouped into four influencing factors: *Vulnerability Discovery Experience*, *Underlying System Knowledge*, *Access to the Development Process*, and *Motivation*. We found that both groups of practitioners expect increases in Vulnerability Discovery Experience and Underlying System Knowledge to improve vulnerability discovery success. Further, we found that hackers and testers reported similar levels of underlying system knowledge, yet the most important difference between our hackers and testers was in their vulnerability discovery experience. To our surprise, we did not find a straightforward relationship between increased access to the development process and successful vulnerability finding. Finally, the impact of different motivational influencing factors was likewise more complex than expected.

A. Vulnerability discovery experience

Overall, hackers and testers agreed that prior experience finding vulnerabilities significantly improves their vulnerability discovery process (T=10, H=13); the key difference is that hackers reported notably more experience than testers.

In particular, we find that regardless of role, experience improves a practitioner's ability to efficiently identify the attack surface, select test cases, recognize vulnerabilities, and sell the resulting report. Both groups reported that the best approaches to gaining the relevant experience are real-world code analysis, hacking exercises, learning from their community, and prior bug reports. However, hackers were more likely than testers to rely on hacking exercises and bug reports. Further, hackers are exposed to a wider variety of vulnerabilities across all these learning approaches. Figure 2 shows the effect of vulnerability discovery experience on phases of the process and the ways practitioners develop experience.

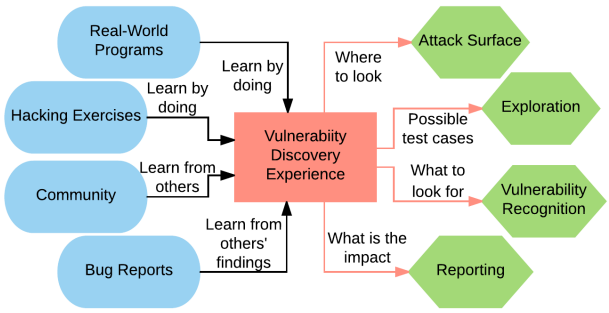

Fig. 2: Vulnerability Discovery Experience Category Graph.

1) How does experience affect the process? Across both groups of practitioners, participants identified several key ways that experience adds to vulnerability discovery success.

Helps recognize a problem quickly. Our participants frequently mentioned learning to recognize patterns that indicate a problem based on prior experience (T=6, H=14). For example, as he explores a program for possible bugs, H10H stated that he has a set of potential problems that he knows could occur based on prior experience. He said "I know that if there's a loop [that's] going through my input, it could be going out of bounds on the array, could be underflow, overflow." Relatedly, most participants discussed maintaining a mental or physical list of all the vulnerabilities they have seen through past experience and checking for these whenever they test a new piece of software (T=9, H=11).

Informs test case selection. In complex real-world systems, it is impractical to perform a complete search of all possible program inputs, so our participants stated that they rely on their intuition, learned from prior experience, to triage (T=9, H=8). For example, T3W discussed creating a list of standard test cases "based on things we've found from Rapid7 [web security scanning tool]" or after asking "one of the developers…if there is other security testing we should be doing." From her experience, she "broadened the scope of security testing at the time [was just SQL injection], and brought in cross-site scripting." H2H explained how he built a set of file formats that he tries to open "in some random image parser, and half the time it would [cause a crash]." He said that he created his list based on his experience working with other security professionals in an "apprentice"-like situation where "You watch them, they watch you, and soon you're doing it on your own."

Helps identify the attack surface. We observed that only participants with more experience mentioned indirect inputs as part of the attack surface (T=4, H=10). Indirect inputs are more difficult to identify because they require a complex combination of events that may not occur frequently. Typically, our practitioners suggested that they only know to look for these complex interactions because they have seen something similar previously. T3W discussed learning about indirect inputs

after incidentally finding a vulnerability in the way a program accepted user input from a LinkedIn third-party login service, "as soon as I found the LinkedIn problem, I made sure to test [FB and Twitter] to make sure [they were processed correctly]. And if we did allow login with another 3rd party in the future, I would check that too."

Helps describe a vulnerability's impact. Testers and hackers leverage prior experience to explain how a vulnerability could be used by a malicious actor when arguing its impact to developers. T10W recalled a time where he used the story of a denial of service attack, caused by the same type of vulnerability, to explain the importance of fixing a new problem quickly.

Without experience, slower and more random. Without prior experience guiding triage, our practitioners relied on stumbling across vulnerabilities incidentally (T=5, H=4); or on their curiosity (T=8, H=5), personal creativity (T=3, H=6), and persistence (T=2, H=9) with ample time (T=4, H=9) to dig through the complexity of a program. Such incidental discovery is time consuming and haphazard, with little consistency in results. H1H described looking for a vulnerability in a complex program and "spent the whole summer on it and failed", but after reviewing bug reports for similar programs, he returned to searching the same program and found a vulnerability after "about a month". Thus, prior experience provides a useful and, in the opinion of some of our hackers, critical stimulus to the bug finding process (T=0, H=4).

2) How is experience developed? Our participants developed experience through hands-on practice, supplemented by support from their peers and by reading other practitioners' vulnerability reports. Most notably, hackers reported a greater variety of learning methods, and more diverse experiences within each method, than testers; as a result, hackers developed more, and more valuable, experience.

Gained by searching real-world programs. All of our testers mentioned gaining experience through their job, supporting findings from Lethbridge et al [35]. Six reported gaining vulnerability-discovery experience by incidentally finding security vulnerabilities while seeking out functionality bugs. Four reported learning something from their company's security-training best practices, but also reported that these best practice guides provide, at best, limited information.

The hackers in our study also develop hands-on experience through employment, which tended to be in security-specific roles such as full-time bug bounty participation and contracted penetration testing (H=13). As might be expected, this security-focused experience provides a strong advantage. Additionally, the ad-hoc and frequently changing nature of hackers' employment exposes them to a wider variety of programs, and therefore types of vulnerabilities, compared to testers who focus only on a single program or a few programs developed by their company and change focus less frequently.

In addition to their full-time jobs, we found that many of our hackers and some testers performed vulnerability discovery on real-world programs as a hobby (T=3, H=11). These participants explained that they searched for vulnerabilities, though there was no expected economic benefit, for the purpose of hands-on learning and personal enjoyment.

Gained through hacking exercises. Many of our hackers and some of our testers participate in hacking exercises like capture-the-flag competitions or online war games [101], [102] (T=4, H=13). These exercises expose players to a variety of vulnerabilities in a controlled, security-specific setting with little program functionality aside from vulnerable components. H3G explained that hacking exercises help players focus on important details without becoming "overloaded"; these exercises also offer a "way of measuring the progress of your skills." Notably, the four testers had participated in only a few narrowly-focused workplace competitions, while our hackers mentioned many broad-ranging exercises.

Learned from their community. Both hackers and testers reported similar experiences learning through colleagues, both within and external to their workplace. Participants mention learning from co-workers (T=7, H=7); from hobbyist (T=7, H=10) and professional (T=2, H=0) organizations in which they are a member; and from informal personal contacts (T=6, H=12). Within these communities, practitioners are taught by those with more experience (T=10, H=13) and learn by working through and discussing difficult problems with their peers (T=6, H=9). For example, T5W described "Just watching other people test, grabbing what one person uses and then another and adding it to your own handbook." H2H explained that starting his career at a security company with "a lot of institutional knowledge" was critical to his development because he had "a lot of folks that I was able to pick their brain." Whenever personal contacts are not sufficient, practitioners also seek out information published online, typically in the form of expert blog articles and web forum posts (T=6, H=10).

Learned from prior vulnerability reports. Additionally, many participants—but particularly hackers—regularly read other individuals' vulnerability reports or discussed vulnerabilities found by colleagues to learn about new vulnerability types and discovery techniques, essentially gaining practical experience vicariously (T=6, H=15). H1H described using bug reports to test his vulnerability finding skills. Before reading a report, he asks himself, "Can I see the bug?" in the vulnerable version of the program. If the answer is no, he looks at the report to see "what the issue was and what the fix was and then where in the source the bug was." However, testers commonly only look at internal reports (T=5, H=0), whereas hackers view reports from a variety of programs (T=1, H=15), exposing them to a wider range of experiences.

Rarely learned through formal education. Finally, some participants mentioned more formal training such as books (T=1, H=7), academic courses (T=2, H=6), and certifications (T=0, H=1). In all cases, however, these methods were perceived to only support attaining the skills to participate in hands-on methods, not to be sufficient on their own.

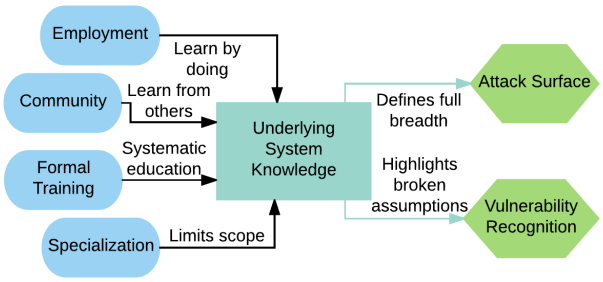

Fig. 3: Underlying System Knowledge Category Graph.

B. Underlying system knowledge

Almost all participants in both populations (T=8, H=15) emphasized the importance of underlying system knowledge (e.g., operating systems, programming languages, network protocols, and software libraries) for successful vulnerability discovery. Unlike with vulnerability discovery experience, our hackers and testers expressed similar levels of system knowledge. Instead, the biggest variation is in which underlying systems participants understand, primarily due to software specialization. Many participants reported focusing on a particular type of software (e.g., web, mobile, host) out of necessity, such as limited time to maintain proficiency in all software types (T=5, H=11). We found that practitioners in both populations limit their vulnerability searches based on their specialty (e.g., mobile specialists only consider a mobile app and not its associated web server).

Both populations indicated that system knowledge plays a role in the Attack Surface phase; hackers were more likely to report that it also plays a role in the Vulnerability Recognition phase (See Figure 3.)

1) How does system knowledge affect the process? Understanding the underlying system components allows practitioners to recognize vulnerabilities caused by discrepancies between the developer's assumptions about how the system behaves and what actually occurs (T=2, H=6). H14W described how his understanding of Mozilla web add-ons helps him recognize vulnerabilities in other developers' code, saying that add-on developers "have no idea what they are doing there, and I see they do horrible stuff."

Strong system knowledge helps practitioners identify more input vectors into a program, as well as the full range of potential inputs (T=5, H=12). H1H gave an example of better system understanding improving his view of the attack surface: "I took [Operating Systems] where I was writing a kernel, and that was incredibly important. It wasn't until I took this that I really understood what the attack surfaces really were... The idea of being the [Virtual Machine] host where you're communicating with the GPU via some channel, I wouldn't have thought about that layer if I hadn't written a kernel."

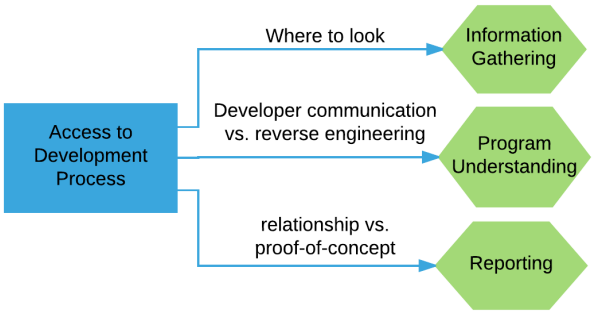

Fig. 4: Access to development process category graph.

2) How is system knowledge developed? The development of system knowledge closely parallels the development of vulnerability discovery experience, with participants relying on hands-on experience and the community. Participants indicated learning through a mixture of on-the-job learning as a tester or hacker (T=10, H=13) and experience as a developer (T=6, H=11) or systems administrator (T=0, H=3). H5M discussed the impact of his prior employment; "I worked at an antivirus company and you had to look at a lot of samples really quick... and [now] it's easy to say 'Ok, here's where they're doing this'... and just quickly looking at it."

Participants also mentioned using community (T=7, H=7) and online resources (i.e., expert blogs and web forums) (T=7, H=8) to supplement their experiential learning. Participants also learn from standards documents such as network protocol RFCs and assembly language instruction set documentation (T=2, H=4). Again, very few mentioned formal education (T=2, H=2), and of those who did, none considered it necessary. H6G explained that he has read some good books and has a computer science degree, but finds hands-on learning the best because "For me I need ten hours reading a book. It's the same as 2 [or] 3 hours trying to solve a challenge."

C. Access to development process

Another factor that influences the vulnerability discovery process is whether a practitioner has access to the development process. Figure 4 shows the effect of this access on the phases of vulnerability discovery. Clearly, because testers serve in an internal role, they have greater access to the source code, program requirements, and the developers themselves; they are also commonly involved in program-design decisions (T=7, H=0). All of our hackers, conversely, are (intentionally) outsiders (H=15) approaching the program as a black box with access at most to the source code if it is an open-source project or unobfuscated client-side script. Our participants report that both perspectives have key advantages and disadvantages: as outsiders by design, hackers are not biased by the assumptions of the developers, but testers have potentially valuable inside knowledge as well as an advantage in communicating findings to developers.

Internal efforts rely on documentation and direct developer

communication. When gathering information, most testers rely on internal code tracking databases and communication with developers and other testers to determine the results of prior vulnerability discovery efforts (T=9, H=0). When trying to understand the program, because testers are involved in the program's design, they get into the mindset of developers by talking to them (T=8, H=0). T6W described participating with developers and other stakeholders in "a design session. That's where we are going to put down the requirements." This session includes discussions about how to build the system and "what kind of testing we are going to [do]."

Having internal access to the development process can reveal flawed assumptions that would never be found by an outsider. However, knowing too much about the program going into the vulnerability search can blind the investigator to certain issues. T4G explains this, saying, "I try to learn as much about it without knowing too much about it. . . . It's hard to ignore certain details once you know about certain areas already." However, T4G still recognized the value of communicating with his developers, saying, "You can give feedback to your teammates, your developers, your product owners. . . . You're coming back with information, and then they react on it. Then you have to go back there [to explore] again."

External efforts use black-box testing and reverse engineering techniques. Because hackers do not have access to internal resources, they have to use more complicated methods to directly interrogate the system. When initially gathering information about a program, hackers use network-scanning tools and other black-box enumeration techniques to determine how the program is built and what underlying technologies it uses (T=0, H=5). During the program understanding phase, hackers rely only on their ability to reverse engineer the developer's intentions by reading and executing the code in lieu of talking to developers (T=0, H=15). H9G builds a clear picture of how the developer is thinking and what they are trying to do by looking directly at their code, because "when you look at the binary. . . you get a more intimate look into how the programmer was thinking." He reads the code to "see certain implementations and certain patterns in the code. . . that can potentially allow you to make an assumption about a part of the specification."

Building rapport with developers. In contrast to the mixed effect on the vulnerability search, our participants indicated that having greater access to the development process provides an advantage when reporting the vulnerability. Our testers discussed using this connection to develop a shared language about the program (T=8, H=0) and build a relationship where they can go to the developers directly to discuss the issue and mitigate the problem (T=9, H=0). T1W stated that he tries "to use the same verbiage, so if for example I'm testing an application and I'm referencing certain parts, . . . [I'll] see how they name those specific fields. . . and I'll try to use the terms they're using versus regular colloquial terms." He explained that the shared language and relationship allows him to avoid misunderstandings that could slow or even stop the remediation process.

Our hackers rarely have the same rapport because, as external participants, they communicate with developers only when they find a vulnerability, which may only occur once per program. In a few cases, our hackers were able to develop a strong relationship with a particular company (H=2), but this only occurred after they submitted multiple reports to that company. H8M focuses on a very specific program type, mobile device firmware, and therefore has developed a relationship with most of the major companies in this area. He described adjusting the information he reports depending on previous interactions. For less security-proficient companies he needs "to go into full details, as well as sending a fully compiled, weaponized exploit," but for companies he has a better relationship with, he just says "In this application in this class, you don't handle this right," and they can identify and fix the issue quickly. This avoids wasting his time creating a lengthy report or developers' time reading it.

Hackers make up for lack of access with proofs-of-concept. Because most hackers have minimal communication with developers, they stressed the necessity of proving the existence and importance of the vulnerability with a *proof-of-concept* exploit to avoid spending significant amounts of time explaining the problem. H3G explained that "including the proof-of-concept takes more time to develop, but it saves a lot of time and communication with the [developers], because you show that you can do an arbitrary [code execution]. . . and that this theoretical vulnerability cannot be mitigated." While this approach is straightforward, developing an exploit can be the most time-consuming part of the process (H=2), and developers may not accept a report even in the face of evidence (T=7, H=9) or appropriately fix the code because they do not understand the root of the problem (T=7, H=9). H15W gave an example of a time when he was reviewing a bug report and found that "they didn't fix it properly. [It was] still exploitable in other ways." Testers overcome these challenges by spending time in discussion with the developers to clear up misunderstandings, but hackers typically do not have these necessary relationships and access to developers.

D. Motivation

The final influencing factor on the discovery process is a practitioner's motivation for looking for vulnerabilities. Figure 5 illustrates how motivations affect the discovery process. Most of our participants select which programs to search and what parts of the code to look at based on a calculation of likelihood to find vulnerabilities versus the value of the vulnerabilities found (T=10, H=11). The four hackers who did not describe this likelihood versus value calculation still consider likelihood as a factor (T=10, H=15), but either are paid a fixed rate as part of an internal security team or contracted review or are motivated by some non-monetary benefit (see Section VI-D). Overall, our hackers and testers estimate vulnerability likelihood similarly, but differ significantly when determining value. Additionally,

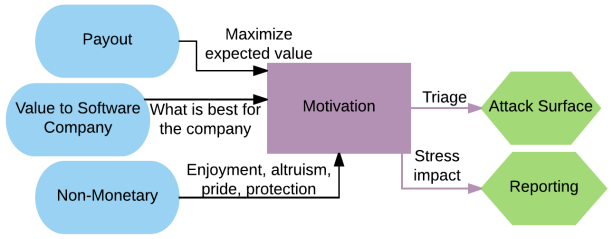

Fig. 5: Motivation category graph.

we found that all participants were motivated to report clearly, no matter their likelihood vs. value calculation.

Estimating likelihood. Because most modern code bases are very large, both populations expressed the need to triage which program components to search based on the likelihood of finding a vulnerability. Both populations described several similar heuristics for this. First, practitioners focus on code segments that they expect were not heavily tested previously (T=5, H=11). H7W, for example, considers where developers are "not paying attention to it [security] as much."

Next, testers and hackers look at parts of the code where multiple bugs were previously reported (T=3, H=9). As T2W said, "There were issues with those areas anyway…so I figured that that was probably where there was most likely to be security issues…bugs cluster."

Both populations mentioned situations when code is new (e.g., rushed to release to fix a major feature issue) (T=5, H=5), or when they do not think the developers understand the underlying systems they are using (e.g., they noticed an odd implementation of a standard feature) (T=1, H=3). Additionally, some hackers also looked at old code (e.g., developed prior to the company performing stringent security checks) (T=0, H=7) and features that are rarely used (T=0, H=3).

Testers determine value by impact to company. As we would expect, testers determine value by estimating the negative effect to the company if exploited (T=8, H=3) or if the program fails a mandated audit (e.g., HIPAA, FERPA) (T=4, H=0). Because of this motivation, they tend to focus on features that are most commonly used by their user base (T=2) and areas of the code that handle sensitive data (e.g., passwords, financial data) (T=8). T5W said he considers "usage of the site, [that is] how many people are going to be on a certain page or certain area of the site, [and] what's on the page itself, [such as] forms" to determine where a successful attack would have the most impact.

Hackers maximize expected payout using several strategies. Previous research has shown that hackers are more likely to participate in a program whenever the bounties are higher [17], and bounty prices increase with vulnerability severity [16]. We also observed that hackers cite the size of the bounty payout as their key motivator; however, we found that hackers follow one of two strategies when deciding how to best maximize their collective payouts.

The first strategy seeks out programs where the hacker has a competitive advantage based on specialized knowledge or experience that makes it unlikely that others will find other similar vulnerabilities (H=9). Hackers following this strategy participate in bug bounties even if they are unlikely to receive immediate payouts, because they can gain experience that will help them later find higher-payout vulnerabilities. H1H said that he focuses on more complex problems even though "I had no success for the first year, I knew that the barrier to entry was so high, that once I got good enough, then it would work out consistently that I could find bugs and get rewards…once you get good at it there's less competition."

The other payout maximizing strategy we observed is to primarily look for simple vulnerabilities in programs that have only recently started a bug bounty program (H=8). In this strategy, hackers race to find as many low-payout vulnerabilities as possible as soon as a program is made public. Hackers dedicate little time to each program to avoid the risk of report collisions and switch to new projects quickly. H12W said that he switches projects frequently, just looking for "low-hanging fruit," because "somebody else could get there before you, while you are still hitting your head on the wall on this old client." This aligns with the phenomenon observed by Maillart et al., where hackers switch quickly to new bug bounties because they are more likely to have more vulnerabilities [12]. We found that hackers typically consider this approach when searching for web vulnerabilities, which have a "lower barrier to entry" than finding vulnerabilities in host software for which "the process to become proficient is higher [harder]" (H1H).

Additionally, some hackers completely avoid any company they have previously had poor relations with, either because they do not think it is likely they will be compensated fairly for their efforts or because the payment is not worth the administrative struggle (T=0, H=6). H9G described submitting a remote-code-execution vulnerability, but never receiving a response, when it should have garnered a large bounty based on the company's published policy. He said that "When we encounter that hostile behavior, that's pretty much an instant turn-off" from working with that company again.

Some participants also consider non-monetary value. Specifically, participants cited motivations including altruism (i.e., bounty paid to charity or improved security for the greater good) (T=2, H=7), enjoyment (T=1, H=11), peer pressure (T=0, H=1), and personal protection (i.e., fix security bugs in products they use to avoid personal exploitation) (T=0, H=2). However, these factors are commonly secondary to monetary value.

All practitioners are motivated to report well. Practitioners' motivations also influence how they communicate with developers when reporting. Both populations expressed the need to make developers aware of the importance of fixing these bugs (T=10, H=12). Testers are only able to prevent harm to the company if developers accept and adequately fix the vulnerabilities they report. Hackers, motivated by a bug bounty payout, receive their payment only when the company accepts their report and are only paid at the level they expect if the

developers agree with their assessment of severity. Participants defined importance as a function of the business impact on the company (T=8, H=3), how much control the vulnerability gives the attacker (e.g., limited data leakage vs. arbitrary code execution) (T=3, H=6), and how easily an attacker can exploit the vulnerability (T=2, H=6). T10W said ,"You need to be able to express not only what the problem is and where the problem lies, but also how this could be used to do X amount of damage." Additionally, some hackers discussed spending time after finding a vulnerability to understand the full implications of the issue (T=0, H=4). H9G said "When I find an issue, I don't necessarily rush to the developer... I could probably chain the vulnerability to other vulnerabilities to be more impactful and more impressive... [and] I get paid more, which is certainly a factor."

Our practitioners also emphasized the need to make their reports easy for developers to understand by considering the technical and security background of their audience (T=7, H=11). As T2W stated, when "there's not enough experience with security across the [development] team, I tend to give them more information to make it easier." Some practitioners also use phrasing and wording that are easy to read (T=4, H=5). T1W said he checks to see if "I missed anything grammar-wise... does it have proper flow?" If he thinks it might be hard to read, he "pull[s] another tester and say[s], 'Hey, does this make sense?' " In some cases, practitioners use a fixed format (T=6, H=3) so that developers know where to look for specific information based on previous reports or by looking at the headings. Finally, many participants discussed maintaining an open-minded, respectful tone when discussing the vulnerability to avoid triggering a defensive response (T=8, H=5). T2W stressed the importance of respectful tone, saying, "Probably the biggest thing is keeping it factual and neutral. Some developers take any [report] as an attack on their ability to code."

VII. Discussion and Recommendations

Our key findings can be summarized as follows:

- The two factors most critical to vulnerability discover success are vulnerability discovery experience and underlying system knowledge.
- Both hackers and testers typically develop sufficient system knowledge through their employment and interactions with their community.
- Although hackers and testers develop vulnerability discovery experience through similar means, hackers are exposed to a wider variety of programs and vulnerabilities through the different types of employments, exercises, and communities they are involved in and the more diverse bug reports they read. This provides hackers an important advantage over testers.
- Access to the development process is a mixed blessing. Access facilitates reporting for testers by building rapport and shared language, but "outsider by design" status allows hackers to recognize mistaken assumptions.
- Hackers attempting to maximize value typically pursue one of two strategies: identify "low-hanging fruit" quickly or develop a deep knowledge advantage.

With these findings in mind, we suggest recommendations for organizations and individuals involved in software vulnerability discovery and directions for future work.

A. Training in the workplace

Our results suggest that extending testers' vulnerability discovery experience will improve their efficacy at finding vulnerabilities before release. We suggest two approaches for use within testers' existing work context; we also recommend future work to explore how to expand that context.

Security champions. Many of our testers described learning from more experienced testers (T=8). As a first change, we recommend hiring a small number (one or two) hackers to work alongside testers, highlighting potential vulnerabilities and sharing security-testing techniques. Deliberately introducing hackers to the team should cultivate learning opportunities. T8H discussed the success of this approach in her organization, saying, "I had two gentlemen... who were really into security testing. ... They eventually went on to create a whole new security team. ... Most of my security testing is all from what I've learned from them." T8H emphasized that this effort, which began with two testers experienced in security pointing out problems to their less experienced co-workers, led within three years to development of a company-wide security consciousness. Further, she said that external security reviews of their product now find many fewer vulnerabilities than they did prior to introducing security champions.

Bug-report-based exercises. Many of our testers spend time discussing interesting bugs found by their peers in regular training sessions (T=7). However, simply discussing a vulnerability does not allow the hands-on practice our participants considered necessary. Instead, we suggest hands-on training based on vulnerabilities previously found in the company's code, either via formal exercises or simply by asking testers to search the pre-fix code and try to find the vulnerability (as suggested by H1H in Section VI-A2. Such exercises will allow testers not only to learn about different vulnerabilities, but also to gain practical experience looking for them.

Future work to increase variety of experiences. The aforementioned approaches, however, will still only expose testers to a limited range of vulnerabilities within the program(s) on which they work. Further research is required to determine the best way to provide broader exposure to testers. Many of our testers participate in internal hacking exercises (T=6), but it was not clear why they do not participate in external exercises. Prior research has found that these exercises typically require a significant time commitment and prior knowledge, which we hypothesize are not a good fit for testers [103], [104]. One possible solution is to create tailored CTFs with hints that slowly introduce new concepts, as some CTFs currently do [105]. This approach is referred to as "scaffolding" in

education literature and provides students the necessary support to allow learning and avoid despair [106].

Similarly, many testers cited internal bug reports as a learning source (T=6), but they do not spend time reading external reports like hackers do (T=1, H=13). One possible reason could be that it is difficult to find vulnerability reports without knowing the correct online resources to consult. Currently bug reports are dispersed among corporate vulnerability disclosure sites [81], [86], [107], personal GitHub repos [108], community mailing lists [109], and public vulnerability databases [110], [111]. Creating a single aggregated repository or searchable source for bug reports and discovery tutorials, and pointing testers to it, could expose testers to a wider range of information.

Further work should evaluate these techniques and develop others to encourage testers to expand the variety of their vulnerability discovery experience.

B. Hacker-developer relationships

While improving testers' vulnerability-finding skills could meaningfully improve security, companies will likely still need security experts to find the most complex problems. Unfortunately, many of our hackers described difficulties communicating with developers, resulting in their findings either not being accepted or not being fixed properly (T=9). To solve this challenge, we look to learn from the strengths of our testers.

Establish consistent relationships early. We found that testers have an advantage in the reporting phase because they have built a relationship with developers through their inherent access to the development process. Additionally, the two hackers who mentioned cultivating a close relationship with a particular company described similar benefits. We therefore recommend companies make efforts to build relationships with hackers as early as possible.

First, we recommend that organizations maintain a consistent point of contact with hackers, so that any time a hacker reports a vulnerability, they communicate with the same person and build a shared language, understanding, and trust. Obviously, a single point of contact is not always useful because a hacker may only report one vulnerability. In these cases, it is important for companies to be as open as possible when providing requirements and expectations to the hacker. Some potential improvements might be to provide more detailed templates and examples of effective reporting, to give feedback or ratings on (specific sections of) reports and how they did or didn't help developers, and answer hacker questions throughout the process to avoid confusion.

Further, our results support industry-wide standardization of vulnerability reporting procedures. This includes agreeing on report templates, "good" reporting examples, and vulnerability definitions. Standardizing expectations and vocabulary should provide consistency across programs and reduce the burden to build individual relationships with each company.

Hackers as security advocates. Additionally, further work is needed to understand how hackers can best convey the importance of a vulnerability, given limited communication channels and time to influence developer decisions. Future research should therefore focus on improving hacker reporting through improved resources and training. For example, a centralized repository of real-world cases where an attacker has exploited a vulnerability that hackers can use as examples could help with demonstrating a vulnerability's importance. Relatedly, Haney and Lutter suggest providing hackers with formal training in how to best advocate for cybersecurity within complex organizational and structural environments [100].

C. Tailor compensation to motivation

Assuming we can improve testers' vulnerability discovery skills so they can find a greater number of relatively simple vulnerabilities, bug bounty policies should be adjusted to focus hacker searches on more challenging vulnerabilities. Based on our results, we suggest two possible bug bounty policy changes below. Further research is necessary to evaluate the efficacy of these changes in real-world settings.

Adjust payout structure as security posture matures. Initially, a company could offer high payouts for all vulnerabilities, attracting hackers via a high likelihood-to-value ratio. This higher participation will likely generate a large number of bug reports that testers can learn from. As the company grows more security-mature internally, it may be possible to reduce payouts for low-level vulnerabilities and shift these funds to pay for more complex vulnerabilities. Further, they may wish to reward hacker specialization by offering bonuses for finding multiple vulnerabilities.

Use non-monetary motivators. In addition to increasing monetary funding, companies can also take advantage of non-monetary motivators to increase the overall payout without committing additional dollars. Most bounties already take advantage of recognition and fun through the use of leaderboards or "walls of fame" as well as the innate enjoyment our participants report deriving from finding a vulnerability. However, companies should also consider the negative effects of their actions during the reporting process, such as delaying or not publishing a report due to company politics (T=0, H=2) or dismissing the report without providing sufficient feedback (T=7, H=9). These actions depress the recognition and enjoyment value for the hacker. Companies can take advantage of hackers' altruistic tendencies by indicating the impact an exploited vulnerability could have on the affected user base in the project's description. Finally, companies could attract hackers seeking personal growth by highlighting skills that could be developed while looking for vulnerabilities and offering online resources to support learning.

ACKNOWLEDGMENTS

We thank Michael Hicks and the anonymous reviewers for their helpful feedback; the two major bug bounty platform companies and the many CTF teams and testing groups that supported our recruitment efforts; and Cynthia Wu and the DC

Agile Software Testing Group for providing valuable insights into the world of software testing.

REFERENCES

[1] Y. Shoshitaishvili, M. Weissbacher, L. Dresel, C. Salls, R. Wang, C. Kruegel, and G. Vigna, "Rise of the hacrs: Augmenting autonomous cyber reasoning systems with human assistance," in *Proc. of the 24th ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017.

[2] N. Rutar, C. B. Almazan, and J. S. Foster, "A comparison of bug finding tools for java," in *Proc. of the 15th International Symposium on Software Reliability Engineering*, ser. ISSRE '04. IEEE Computer Society, 2004, pp. 245–256.

[3] D. Baca, B. Carlsson, K. Petersen, and L. Lundberg, "Improving software security with static automated code analysis in an industry setting." *Software: Practice and Experience*, vol. 43, no. 3, pp. 259–279, 2013.

[4] A. Doupé, M. Cova, and G. Vigna, "Why johnny can't pentest: An analysis of black-box web vulnerability scanners," in *Proc. of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA'10. Springer-Verlag, 2010, pp. 111–131.

[5] A. Austin and L. Williams, "One technique is not enough: A comparison of vulnerability discovery techniques," in *Proc. of the Fifth International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '11. IEEE Computer Society, 2011, pp. 97–106.

[6] N. Antunes and M. Vieira, "Comparing the effectiveness of penetration testing and static code analysis on the detection of sql injection vulnerabilities in web services," in *Proc. of the 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, ser. PRDC '09. IEEE Computer Society, 2009, pp. 301–306.

[7] L. Suto, "Analyzing the effectiveness and coverage of web application security scanners," BeyondTrust, Inc, Tech. Rep., 2007. [Online]. Available: https://www.beyondtrust.com/resources/white-paper/analyzing-the-effectiveness-and-coverage-of-web-application-security-scanners/

[8] L. Suto, "Analyzing the accuracy and time costs of web application security scanners," BeyondTrust, Inc, Tech. Rep., 2010. [Online]. Available: https://www.beyondtrust.com/wp-content/uploads/Analyzing-the-Accuracy-and-Time-Costs-of-Web-Application-Security-Scanners.pdf

[9] G. McGraw and J. Steven, "Software [in]security: Comparing apples, oranges, and aardvarks (or, all static analysis tools are not created equal," Cigital, 2011, (Accessed 02-26-2017). [Online]. Available: http://www.informit.com/articles/article.aspx?p=1680863

[10] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner, "An empirical study on the effectiveness of security code review," in *Proc. of the 5th International Conference on Engineering Secure Software and Systems*, ser. ESSoS'13. Springer-Verlag, 2013, pp. 197–212.

[11] Z. Xu, Q. Hu, and C. Zhang, "Why computer talents become computer hackers," *Communications of the ACM*, vol. 56, no. 4, pp. 64–74, Apr. 2013.

[12] T. Maillart, M. Zhao, J. Grossklags, and J. Chuang, "Given enough eyeballs, all bugs are shallow? revisiting eric raymond with bug bounty programs," in *Proc. of the 15th Workshop on the Economics of Information Security*, ser. WEIS '16, 2016.

[13] Hackerone, "2016 bug bounty hacker report," Hackerone, Tech. Rep., September 2016. [Online]. Available: https://hackerone.com/blog/bug-bounty-hacker-report-2016

[14] A. Mein and C. Evans, "Dosh4vulns: Google's vulnerability reward programs," Google, 2011, (Accessed 02-26-2017). [Online]. Available: https://software-security.sans.org/downloads/appsec-2011-files/vrp-presentation.pdf

[15] Bugcrowd, "The state of bug bounty," Bugcrowd, Tech. Rep., June 2016. [Online]. Available: https://pages.bugcrowd.com/2016-state-of-bug-bounty-report

[16] M. Finifter, D. Akhawe, and D. Wagner, "An empirical study of vulnerability rewards programs," in *Proc. of the 22nd USENIX Security Symposium*, ser. USENIX Security '13, 2013, pp. 273–288.

[17] M. Zhao, J. Grossklags, and P. Liu, "An empirical study of web vulnerability discovery ecosystems," in *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1105–1117.

[18] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.

[19] M. Zhivich and R. K. Cunningham, "The real cost of software errors," *IEEE Security & Privacy*, vol. 7, no. 2, 2009.

[20] M. Soni, "Defect prevention: reducing costs and enhancing quality," *IBM:iSixSigma.com*, vol. 19, 2006. [Online]. Available: https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/

[21] W. Baziuk, "BNR/NORTEL: path to improve product quality, reliability and customer satisfaction," in *Sixth International Symposium on Software Reliability Engineering, ISSRE 1995, Toulouse, France, October 24-27, 1995*, 1995, pp. 256–262. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.1995.497665

[22] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. of the IEEE*, vol. 68, no. 9, pp. 1060–1076, Sept 1980.

[23] K. Olmstead and A. Smith, "Americans and cybersecurity," Pew Research Center, 2017, (Accessed 07-15-2017). [Online]. Available: http://www.pewinternet.org/2017/01/26/americans-and-cybersecurity/

[24] J. Surowiecki, *The wisdom of crowds*. Anchor, 2005.

[25] M. Zhao, A. Laszka, T. Maillart, and J. Grossklags, "Crowdsourced security vulnerability discovery: Modeling and organizing bug-bounty programs," in *Proc. of the 4th AAAI Workshop on Mathematical Foundations of Human Computation*, ser. HCOMP '16, November 2016.

[26] K. Huang, M. Siegel, S. Madnick, X. Li, and Z. Feng, "Poster: Diversity or concentration? hackers' strategy for working across multiple bug bounty programs," in *Proc. of the 37th IEEE Symposium on Security and Privacy*, ser. SP '16, 2016.

[27] S. Ransbotham, S. Mitra, and J. Ramsey, "Are markets for vulnerabilities effective," *MIS Quarterly*, vol. 36, no. 1, pp. 43–64, 2012.

[28] A. Ozment, "Bug auctions: Vulnerability markets reconsidered," in *Third Workshop on the Economics of Information Security*, 2004.

[29] K. Kannan and R. Telang, "Market for software vulnerabilities? think again," *Manage. Sci.*, vol. 51, no. 5, pp. 726–740, May 2005.

[30] A. Algarni and Y. Malaiya, "Software vulnerability markets: Discoverers and buyers," *International Journal of Computer, Information Science and Engineering*, vol. 8, no. 3, pp. 71–81, 2014.

[31] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proc. of the 31st International Conference on Software Engineering*, ser. ICSE '09. IEEE Computer Society, 2009, pp. 298–308.

[32] M. Hafiz and M. Fang, "Game of detections: how are security vulnerabilities discovered in the wild?" *Empirical Software Engineering*, vol. 21, no. 5, pp. 1920–1959, 2016.

[33] M. Fang and M. Hafiz, "Discovering buffer overflow vulnerabilities in the wild: An empirical study," in *Proc. of the Eighth International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. ACM, 2014, pp. 23:1–23:10.

[34] T. C. Summers, K. J. Lyytinen, T. Lingham, and E. Pierce, "How hackers think: A study of cybersecurity experts and their mental models," in *Proc. of the 3rd International Conference on Engaged Management Scholarship*, ser. EMS '13. EDBAC, 2013.

[35] T. C. Lethbridge, J. Diaz-Herrera, R. J. J. LeBlanc, and J. B. Thompson, "Improving software practice through education: Challenges and future trends," in *Future of Software Engineering*. IEEE Computer Society, 2007, pp. 12–28.

[36] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering*. IEEE Computer Society, 2007, pp. 85–103.

[37] M. Al-Banna, B. Benatallah, and M. C. Barukh, "Software security professionals: Expertise indicators," in *Proc. of the 2nd IEEE International Conference on Collaboration and Internet Computing*, ser. CIC '16, 2016, pp. 139–148.

[38] J. Cowley, "Job analysis results for malicious-code reverse engineers: A case study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-2014-TR-002, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=91548

[39] T. J. Holt, "Lone hacks or group cracks: Examining the social organization of computer hackers," *Crimes of the Internet*, pp. 336–355, 2009.

[40] T. J. Holt, "The attack dynamics of political and religiously motivated hackers," *Cyber Infrastructure Protection*, pp. 161–182, 2009.

[41] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security apis," *IEEE Security Privacy*, vol. 14, no. 5, pp. 40–46, Sept 2016.

[42] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Proc. of the 37th IEEE Symposium on Security and Privacy*, ser. SP '16, May 2016, pp. 289–305.

[43] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "How internet resources might be helping you develop faster but less securely," *IEEE Security Privacy*, vol. 15, no. 2, pp. 50–60, March 2017.

[44] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the usability of cryptographic apis," in *Proc. of the 38th IEEE Symposium on Security and Privacy (SP)*, ser. SP '17, May 2017, pp. 154–171.

[45] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong? A qualitative usability study," *CoRR*, vol. abs/1708.08759, 2017.

[46] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos, and Y. Zhuang, "It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots," in *Proc. of the 30th Annual Computer Security Applications Conference*, ser. ACSAC '14. ACM, 2014, pp. 296–305.

[47] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. A. F. Weir, and S. Fahl, "A stitch in time:supporting android developers in writing secure code," in *Proc. of the 24nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017.

[48] C. Weir, A. Rashid, and J. Noble, "I'd like to have an argument, please: Using dialectic for effective app security," in *Proc. of the 2nd European Workshop on Usable Security*, ser. EuroUSEC '17, 04 2017.

[49] K. Ermoshina, H. Halpin, and F. Musiani, "Can johnny build a protocol? co-ordinating developer and user intentions for privacy-enhanced secure messaging protocols," in *Proc. of the 2nd European Workshop on Usable Security*, ser. EuroUSEC '17, 04 2017.

[50] L. Lo Iacono and P. Gorski, "I do and i understand. not yet true for security apis. so sad," in *Proc. of the 2nd European Workshop on Usable Security*, ser. EuroUSEC '17, 04 2017.

[51] P. Morrison, B. H. Smith, and L. Williams, "Surveying security practice adherence in software development," in *Proc. of the 5th Hot Topics in Science of Security: Symposium and Bootcamp*, ser. HoTSoS '17. ACM, 2017, pp. 85–94.

[52] D. Hovemeyer and W. Pugh, "Finding more null pointer bugs, but not too many," in *Proc. of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, ser. PASTE '07. ACM, 2007, pp. 9–14.

[53] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proc. of the 35th International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, pp. 672–681.

[54] L. Layman, L. Williams, and R. S. Amant, "Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools," in *Proc. of the First International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '07. IEEE Computer Society, 2007, pp. 176–185.

[55] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. Murphy-Hill, "Just-in-time static analysis," in *Proc. of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '17. ACM, 2017, pp. 307–317.

[56] N. Ayewah and W. Pugh, "A report on a survey and study of static analysis users," in *Proc. of the 1st Workshop on Defects in Large Software Systems*, ser. DEFECTS '08. ACM, 2008, pp. 1–5.

[57] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in *Proc. of the 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE '15. ACM, 2015, pp. 248–259.

[58] T. W. Thomas, H. Lipford, B. Chu, J. Smith, and E. Murphy-Hill, "What questions remain? an examination of how developers understand an interactive static analysis tool," in *Proc. of the 12th Symposium on Usable Privacy and Security*, ser. SOUPS '16. USENIX Association, 2016.

[59] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann, "Quantifying developers' adoption of security tools," in *Proc. of the 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE '15. ACM, 2015, pp. 260–271.

[60] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *Proc. of the 37th International Conference on Software Engineering*, ser. ICSE '15. IEEE Press, 2015, pp. 598–608.

[61] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh, and S. R. Rajagopalan, "A human capital model for mitigating security analyst burnout," in *Proc. of the 11th Symposium On Usable Privacy and Security*, ser. SOUPS '15. USENIX Association, 2015, pp. 347–359.

[62] D. Botta, R. Werlinger, A. Gagné, K. Beznosov, L. Iverson, S. Fels, and B. Fisher, "Towards understanding it security professionals and their tools," in *Proc. of the 3rd Symposium on Usable Privacy and Security*, ser. SOUPS '07. ACM, 2007, pp. 100–111.

[63] P. Jaferian, D. Botta, F. Raja, K. Hawkey, and K. Beznosov, "Guidelines for designing it security management tools," in *Proc. of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*, ser. CHiMiT '08. ACM, 2008, pp. 7:1–7:10.

[64] N. F. Velasquez and S. P. Weisband, "Work practices of system administrators: Implications for tool design," in *Proc. of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*, ser. CHiMiT '08. ACM, 2008, pp. 1:1–1:10.

[65] S. C. Sundaramurthy, J. McHugh, X. Ou, M. Wesch, A. G. Bardas, and S. R. Rajagopalan, "Turning contradictions into innovations or: How we learned to stop whining and improve security operations," in *Proc. of the 12th Symposium on Usable Privacy and Security*, ser. SOUPS '16. USENIX Association, 2016, pp. 237–251.

[66] S. Fahl, Y. Acar, H. Perl, and M. Smith, "Why eve and mallory (also) love webmasters: A study on the root causes of ssl misconfigurations," in *Proc. of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. ACM, 2014, pp. 507–512.

[67] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, ""i have no idea what i'm doing" - on the usability of deploying HTTPS," in *Proc. of the 26th USENIX Security Symposium*, ser. USENIX Security '17. USENIX Association, 2017, pp. 1339–1356.

[68] S. Watson and H. R. Lipford, "A proposed visualization for vulnerability scan data," in *Proc. of the 13th Symposium on Usable Privacy and Security*, ser. SOUPS '17. USENIX Association, 2017.

[69] A. M'manga, S. Faily, J. McAlaney, and C. Williams, "Folk risk analysis: Factors influencing security analysts' interpretation of risk," in *Proc. of the 13th Symposium on Usable Privacy and Security*, ser. SOUPS '17. USENIX Association, 2017.

[70] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study," *2016 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 158–177, 2016.

[71] A. Beautement, I. Becker, S. Parkin, K. Krol, and A. Sasse, "Productive security: A scalable methodology for analysing employee security behaviours," in *Proc. of the 12th Symposium on Usable Privacy and Security*, ser. SOUPS '16. USENIX Association, 2016, pp. 253–270.

[72] S. Pahnila, M. Siponen, and A. Mahmood, "Employees' behavior towards it security policy compliance," in *Proc. of the 40th Hawaii International Conference on System Sciences*, ser. HICSS '07, Jan 2007, pp. 156b–156b.

[73] M. Workman, W. H. Bommer, and D. Straub, "Security lapses and the omission of information security measures: A threat control model and empirical test," *Computers in human behavior*, vol. 24, no. 6, pp. 2799–2816, Sep. 2008.

[74] M. Siponen, S. Pahnila, and M. Adam Mahmood, "Employees' adherence to information security policies: An empirical study," in *Proc. of the 22nd International Information Security Conference*, ser. SEC '07. Springer US, 2007, pp. 133–144.

[75] J. D'Arcy, T. Herath, and M. Shoss, "Understanding employee responses to stressful information security requirements: A coping perspective," *Journal of Management Information Systems*, vol. 31, pp. 285–318, 10 2014.

[76] K. Guo, Y. Yuan, N. Archer, and C. Connelly, "Understanding nonmalicious security violations in the workplace: A composite behavior model," *Journal of management information systems*, vol. 28, no. 2, pp. 203–236, Oct. 2011.

[77] J. M. Blythe, L. Coventry, and L. Little, "Unpacking security policy compliance: The motivators and barriers of employees' security behav-

iors," in *Proc. of the 11th Symposium On Usable Privacy and Security*, ser. SOUPS '15. USENIX Association, 2015, pp. 103–122.
[78] I. Becker, S. Parkin, and M. A. Sasse, "Finding security champions in blends of organisational culture," in *Proc. of the 2nd European Workshop on Usable Security*, ser. EuroUSEC '17, 04 2017.
[79] K. Charmaz, *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. SagePublication Ltd, London, 2006.
[80] G. Guest, A. Bunce, and L. Johnson, "How many interviews are enough? an experiment with data saturation and variability," *Field methods*, vol. 18, no. 1, pp. 59–82, 2006.
[81] Bugcrowd, "Home - bugcrowd," Bugcrowd, 2016, (Accessed 02-18-2017). [Online]. Available: http://bugcrowd.com
[82] CTFTime, "Ctftime.org / all about ctf (capture-the-flag)," CTFTime, 2017, (Accessed 06-08-2017). [Online]. Available: https://ctftime.org
[83] Meetup, "We are what we do | meetup," Meetup, 2017, (Accessed 06-08-2017). [Online]. Available: https://www.meetup.com
[84] "Association of software testing | software testing professional association," Association of Software Testing, 2017, (Accessed 06-08-2017). [Online]. Available: https://www.associationforsoftwaretesting.org/
[85] "Ministry of testing - co-creating smart testers," Ministry of Testing, 2017, (Accessed 06-08-2017). [Online]. Available: https://www.ministryoftesting.com/
[86] HackerOne, "Hackerone: Vulnerability coordination and bug bounty platform," HackerOne, 2016, (Accessed 02-18-2017). [Online]. Available: http://hackerone.com
[87] Google, "Issues - chromium," Google, 2016, (Accessed 02-18-2017). [Online]. Available: https://bugs.chromium.org/p/chromium/issues/list
[88] Mozilla, "Components for firefox," Mozilla, 2016, (Accessed 02-18-2017). [Online]. Available: https://bugzilla.mozilla.org/describecomponents.cgi?product=Firefox
[89] M. C. Harrell and M. A. Bradley, "Data collection methods. semi-structured interviews and focus groups," Rand National Defense Research Institute, Tech. Rep., 2009.
[90] J. Annett, "Hierarchical task analysis," *Handbook of cognitive task design*, vol. 2, pp. 17–35, 2003.
[91] A. Strauss and J. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Newbury Park, CA: Sage, 1998, vol. 15.
[92] D. G. Freelon, "Recal: Intercoder reliability calculation as a web service," *International Journal of Internet Science*, vol. 5, no. 1, pp. 20–33, 2010.
[93] A. F. Hayes and K. Krippendorff, "Answering the call for a standard reliability measure for coding data," *Communication methods and measures*, vol. 1, no. 1, pp. 77–89, 2007.
[94] K. Krippendorff, "Reliability in content analysis," *Human Communication Research*, vol. 30, no. 3, pp. 411–433, 2004.
[95] M. Lombard, J. Snyder-Duch, and C. C. Bracken, "Content analysis in mass communication: Assessment and reporting of intercoder reliability," *Human communication research*, vol. 28, no. 4, pp. 587–604, 2002.
[96] A. L. Holbrook, M. C. Green, and J. A. Krosnick, "Telephone versus face-to-face interviewing of national probability samples with long questionnaires: Comparisons of respondent satisficing and social desirability response bias," *Public opinion quarterly*, vol. 67, no. 1, pp. 79–125, 2003.
[97] R. Tourangeau and T. Yan, "Sensitive questions in surveys." *Psychological bulletin*, vol. 133, no. 5, p. 859, 2007.
[98] BugCrowd, "Inside the mind of a hacker," BugCrowd, 2016, (Accessed 02-18-2017). [Online]. Available: https://pages.bugcrowd.com/inside-the-mind-of-a-hacker-2016
[99] Bugcrowd, "Defensive vulnerability pricing model," Bugcrowd, Tech. Rep., 2015. [Online]. Available: https://pages.bugcrowd.com/whats-a-bug-worth-2015-survey
[100] J. Haney and W. Lutters, "Skills and characteristics of successful cybersecurity advocates," in *Proc. of the 13th Symposium on Usable Privacy and Security*, ser. SOUPS '17. USENIX Association, 2017.
[101] CTFTime, "Ctf? wtf?" CTFTime, 2017, (Accessed 06-08-2017). [Online]. Available: https://ctftime.org/ctf-wtf/
[102] T. of Bits, "Find a ctf - ctf field guide," Trail of Bits, 2017, (Accessed 06-08-2017). [Online]. Available: https://trailofbits.github.io/ctf/intro/find.html
[103] R. S. Cheung, J. P. Cohen, H. Z. Lo, F. Elia, and V. Carrillo-Marquez, "Effectiveness of cybersecurity competitions," in *Proc. of the International Conference on Security and Management*, ser. SAM '12. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012, pp. 1–5.
[104] D. H. Tobey, P. Pusey, and D. L. Burley, "Engaging learners in cybersecurity careers: Lessons from the launch of the national cyber league," *ACM Inroads*, vol. 5, no. 1, pp. 53–56, Mar. 2014.
[105] ForAllSecure, "Hackcenter," ForAllSecure, 2017, (Accessed 07-25-2017). [Online]. Available: https://hackcenter.com/about
[106] M. C. Kim and M. J. Hannafin, "Scaffolding problem solving in technology-enhanced learning environments (teles): Bridging research and theory with practice," *Computers & Education*, vol. 56, no. 2, pp. 403 – 417, 2011.
[107] Syncack, "Synack - crowdsourced security," Synack, 2017, (Accessed 06-08-2017). [Online]. Available: https://www.synack.com
[108] J. Doyle, "Google-nest-cam-bug-disclosures," 2017, (Accessed 08-1-2017). [Online]. Available: https://github.com/jasondoyle/Google-Nest-Cam-Bug-Disclosures
[109] Fyodor, "Full Disclosure Mailing List," SecLists.Org, 2017, (Accessed 08-01-2017). [Online]. Available: http://seclists.org/fulldisclosure/
[110] U.S. Department of Commerce, "National Vulnerability Database," NIST, 2017, (Accessed 08-01-2017). [Online]. Available: https://nvd.nist.gov/
[111] Offensive Security, "Offensive security's exploit database archive," 2017, (Accessed 08-1-2017). [Online]. Available: https://www.exploit-db.com/

APPENDIX A
SURVEY QUESTIONNAIRE

Vulnerability-discovery experience.

1) On a scale from 1-5, how would you assess your vulnerability discovery skill (1 being a beginner and 5 being an expert)?
2) Please select the range which most closely matches the number of software vulnerabilities you have discovered. (Choices: 0-3, 4-6, 7-10, 11-25, 26-50, 51-100, 101-50, > 500)
3) How many total years of experience do you have with vulnerability discovery?
4) Please select the range that most closely matches the number of hours you typically spend performing software vulnerability discover tasks per week. (Choices: < 5, 5-10, 10-20, 20-30, 30-40, > 40)
5) Please specify the range that most closely matches the number of hours you typically spend on non-vulnerability discovery, technical tasks per week (e.g. software or hardware programming, systems administration, network analysis, etc.). (Choices: < 5, 5-10, 10-20, 20-30, 30-40, > 40)
6) What percentage of the bugs you have discovered were found in the following contexts?
   a) Bug Bounty programs (i.e., sold specific bug to vendor)
   b) General Software Testing
   c) Penetration Testing
   d) Vulnerability finding exercise (e.g., Capture-the-Flag competition, security course)
   e) Unrelated to a specific program (i.e., for fun or curiosity)
   f) Other
7) What percentage of the bugs you have discovered were in software of the following types?
   a) Host (e.g., Server or PC)
   b) Web Application

c) Network Infrastructure
d) Mobile Application
e) API
f) iOT or Firmware
g) Other

8) What percentage of the bugs you have discovered were of the following types?
   a) Memory Corruption (e.g., buffer overflow)
   b) Input Validation (e.g., XSS, SQL injection, format string misuse)
   c) Cryptographic Error (e.g., weak key use, bad random number generator)
   d) Configuration Error (e.g., unpatched network device)
   e) Incorrect Calculation (e.g., integer overflow, off-by-one error)
   f) Protection Mechanism Error (e.g., incorrect access control, improper certificate check)
   g) Poor Security Practices (e.g., insecure data storage or transmission)

Note: In the previous three questions, we did not provide our own definitions for software type or program type and only provided examples for vulnerability types. We selected terms used by multiple popular bug bounty platforms (i.e., SynAck, HackerOne, and BugCrowd) and allowed participants to select options based on their own definitions. During the interview, we included follow-up questions to understand their definitions for the top ranked item in each category.

Technical skills.

1) On a scale from 1 to 5, how would you assess your proficiency in each of the following technical skills (1 being a beginner or having no experience and 5 being an expert)?
   a) Networking
   b) Database Management
   c) Object-Oriented Programming (e.g., Java, C++)
   d) Functional Programming (e.g., OCaml, Haskell)
   e) Procedural Programming (e.g., C, Go)
   f) Web Development
   g) Mobile Development
   h) Distributed/Parallel Computing
   i) System Administration
   j) Reverse Engineering
   k) Cryptanalysis
   l) Software Testing
   m) Test Automation
   n) Hardware/Firmware Development
   o) Other

Demographics.

1) Please specify the gender with which you most closely identify. (Choices: Male, Female, Other, Prefer not to answer)
2) Please specify your age. (Choices: 18-29, 30-39, 40-49, 50-59, 60-69, > 70, Prefer not to answer)
3) Please specify your ethnicity (Choices: White, Hispanic or Latino, Black or African American, American Indian or Alaska Native, Asian, Native Hawaiian, or Pacific Islander, Other)
4) Please specify which country/state/province you live in.
5) Please specify the highest degree or level of school you have completed (Choices: Some high school credit, no diploma or equivalent; High school graduate, diploma or the equivalent; Some college credit, no degree; Trade/technical/vocational training; Associate degree; Bachelor's degree; Master's degree; Professional degree; Doctorate degree)
6) If you are currently a student or have completed a college degree, please specify your field(s) of study (e.g., Biology, Computer Science, etc).
7) Please select the response option that best describes you current employment status. (Choices: Working for payment or profit, Unemployed, Looking after home/family, Student, Retired, Unable to work due to permanent sickness or disability, Other)
8) If you are working for payment, please specify your current job title.
9) If you are currently working for payment, please specify the business sector which best describes you job (Choices: Technology, Government or government contracting, Healthcare and social assistance, Retail, Construction, Educational services, Finance, Arts/Entertainment/Recreation, Other)
10) Please specify the range which most closely matches your total, pre-tax, household income in 2016. (Choices: < $29,999, $30,000-$49,999, $50,000-$74,999, $75,000-$99,999, $100,000-$124,999, $125,000-$149,999, $150,000-$199,999, > $200,000)
11) Please specify the range which most closely matches your total, pre-tax, household income specifically from vulnerability discovery and software testing in 2016. (Choices: < $999, $1,000-$4,999, $5,000-$14,999, $15,000-$29,999, $30,000-$49,999, $50,000-$74,999, $75,000-$99,999, $100,000-$124,999, $125,000-$145,999, $150,000-$199,999, > $200,000)

APPENDIX B
INTERVIEW QUESTIONS

A. General experience

1) What was the motivation/thought process behind performing vulnerability discovery in the different contexts you listed in the survey?
2) If most of the bugs are in a particular software or bug type:
   a) Why do you focus on a specific area? Why this area?
   b) Have you ever developed software in this area?
   c) Have you worked outside of this area of expertise in the past? What was the reason for the change?
3) If the types of bugs are generally split across software or bug type:

a) What do you see as the importance of staying general?
b) Are there any unique trends you've noticed that encourage you to stay general with your skills?

B. Task analysis

Program selection.

1) In general, how do you decide which software [for testers: part of the program] to investigate for vulnerabilities and which not? What factors do you consider when making this decision?
2) Which of these factors do you consider to be the most important? Why?
3) Are there any factors that you consider non-starters (i.e. reason not to try looking for bugs)?
4) Is there a specific process you use when determining where to look for vulnerabilities and which of these characteristics different software exhibit?
   a) Why did you choose this particular process?
   b) How did you develop/learn this process?
   c) Are there any tools that you use that assist you in this process?
      i) What were the benefits of these tools? Weaknesses?
      ii) Have you ever used anything else for this purpose? What led you to switch?

Vulnerability-discovery process.

1) Once you've selected a software target, what steps do you take when looking for vulnerabilities?
   a) For each step:
      i) What are your goals for this step? What information are you trying to collect?
      ii) What actions do you take to complete this step? Have you every tried anything else? What are the advantages/disadvantages of this set of actions?
      iii) Are there any tools that you use to complete this step? What were the benefits of these tools? Weaknesses?
      iv) What skills do you use to complete this task? Why do you think these skills are important? How did you develop/learn these skills?
      v) How do you know when you have successfully completed this step?
      vi) How do you decide when to take this step? Is it something you repeat multiple times? Do you always do this step?
      vii) How did you learn to take this step? Why did you find this source of information helpful?
2) Is there anything else you haven't mentioned that you've done to try to find vulnerabilities and stopped? What are the main differences between your current process and what you did in the past? What led you to switch?

Reporting.

1) What kinds of information do you include in the report? Do you always report the same information? What factors do you consider when deciding which information to include in the report?
2) What information do you think is the most important in vulnerability reports?
3) Have you ever included/excluded anything that you didn't feel was important, but just included/excluded because you felt it was traditional/expected?
4) What bug report information is the most difficult/time consuming to get?
5) Do you ever look at anyone else's bug reports to learn from them? Why do you think these are helpful?
6) Do you use any special tool for reporting? What were the benefits of these tools? Weaknesses?
7) Do the organizations you submit to reach out to you with questions about the bugs? If so, what do they ask about?
8) Can you give me an example of a bad/good experience you've had with reporting? In your opinion, what factors are the most important for a good reporting experience?

C. Skill development

Learning.

1) Imagine you were asked for advice by an enthusiastic young person interested in learning about vulnerability discovery. How would you recommend they get started? What steps would you suggest they take?

Community participation.

1) Do you have regular communication with other hackers or software testers?
2) How do you typically communicate with others?
3) How important is each community you participate in to your/others development?
4) What community that you belong to do you find most useful? Why?
5) What information do you typically share?
6) How often do you communicate (specifically regarding technical information)?
7) How close are the relationships you have with others? How many other hackers/testers do you communicate with?