Lessons Learned in Five Years of Conducting Security Studies With Software Developers

Alena Naiakshina
Ruhr University Bochum
alena.naiakshina@rub.de

Anastasia Danilova
University of Bonn
danilova@cs.uni-bonn.de

Matthew Smith
University of Bonn, Fraunhofer FKIE
smith@cs.uni-bonn.de

ABSTRACT
End-user-based research was conducted for over 20 years aiming at supporting end-users with software security issues. Thus ample knowledge exists on how to conduct security studies with end-users. However, critical security incidents in the last decade showed that, like end-users, software developers are usually not security experts and need support with security-critical tasks as well. While this motivated researchers to conduct more security studies with software developers, it also highlighted a lack of methodological knowledge concerning security studies with software developers. We derived six recommendations concerning security developer studies’ ecological validity and recruitment challenges based on our lessons learned from the last five years of research with software developers.

ACM Reference Format:

1 INTRODUCTION
Ecological validity issues constitute a significant concern of usable security studies with software developers [5]. It is unclear how knowing they participate in a study or being explicitly instructed to consider security aspects in a task description affects software developers’ security behavior. Like end-users, developers might consider security a secondary task and thus might behave differently in real-world circumstances [5, 16]. Further, it is challenging to recruit enough professionals to obtain a significant amount of relevant study data due to time, spread out geographical locations, and financial constraints [4–6, 19, 27, 31]. A common approach in software engineering studies is the recruitment of convenience samples, such as computer science (CS) students (e.g., [4, 17, 20, 25, 26, 29]). While recent work indicates that CS students can be acceptable proxies for professionals in research studies [4, 6, 18, 31], there are still some caveats [31]. To widen the recruitment pool and include non-student participants, it is common for researchers to resort to online studies and recruit participants online (e.g., [6–8, 15, 21, 22, 30, 32]). Diverse recruitment strategies have been used, such as cold-calling programmers on platforms such as Stack Overflow, GitHub, Meet-up groups, etc. or posting open invitations on social media, in forums, newsletters, and events, with the expectation being that participants without programming knowledge will not sign up for the studies [7, 9, 28]. However, since researchers often offer significantly higher compensation than for end-user studies [22–24], there can be an incentive for participants to take part in studies despite having no programming skill. We conducted lab, online, and field studies with students, freelancers, and company developers investigating their security behavior and different study design variables. In the following, we contribute our lessons learned from the last five years of research with software developers.

2 RESEARCH IMPLICATIONS

2.1 Security deception
To investigate software developers’ security behavior with user-password storage, we conducted a qualitative study with 20 CS students in a laboratory setting [23]. Participants were asked to develop a user registration function for a university social networking platform. To examine whether software developers think about security on their own, half the participants were explicitly prompted to store user passwords securely, while the other half were told that the study was about API usability (non-prompted). The study also investigated whether an application programming interface (API) offering built-in functionalities for secure password storage would help developers produce more secure code than an API requiring developers to choose secure password storage mechanisms manually. Based on their popularity, Spring [2] and JavaServer Faces (JSF) [1] were selected as frameworks for the study. Spring was chosen to represent a more supportive framework with inbuilt functionalities for secure password storage. In contrast, with JSF, developers had to implement secure password storage on their own. None of the participants stored user passwords securely without being prompted in this study. If researchers are testing the usability of security APIs but do not instruct participants to think about security, participants might not use the security features. Our results indicated that security deception might simulate a more realistic environment, as employers or end clients without a background in technology or security rarely ask for secure implementation but rather for software functionality. Researchers might consider including functionality requirements in security tasks to increase the ecological validity of security studies with developers.

Security deception might increase the ecological validity of security studies with developers.

2.2 Qualitative vs. quantitative studies
To further explore methodological requirements, the previous student study was extended to include an additional 20 CS students,
and quantitative analysis was performed [24]. The findings of the qualitative study with students described in Section 2.1, provided already good indications for the results of this follow-up quantitative study. We concluded that qualitative studies might offer valuable insights without recruiting many professionals.

2.3 Study deception
Since participants from the previous student studies reported that they might store user-passwords securely in a real company, we conducted a follow-up study with freelancers. In the hope of increasing the ecological validity, we used a deception study design wherein freelance developers were hired on Freelancer.com for a regular job using a company front created for the study, instead of openly telling them that they were taking part in a study [22]. After completing the programming task, participants were informed about the research context. In this study, participants’ behavior was similar to that of CS students concerning secure password storage practices. While deception in this study was used to ensure that the results would reflect the real work of online freelancers, it can entail additional study design work and negotiations with ethical oversight bodies. Therefore, we replicated the study but announced and ran it as a study on Freelancer.com [11]. Our findings suggested that study deception did not significantly affect this password storage study, and the open recruitment without deception was a viable recruitment method. While the results certainly do not generalize to all developer security studies, it is an essential first indication that freelance developers recruited as part of a study behave similarly to when they are hired for a regular job. We found Freelancer.com to be a suitable source for recruiting enough willing professional developers to work on (study) projects.

2.4 Sample comparison
To investigate whether professional developers employed by companies behave the same as CS students and freelance developers, an additional online study was conducted with professional developers from different companies [21]. However, it was not realistic to hire professionals from companies for a small task without revealing the research context of the study. Therefore, participants were informed that the programming task was requested for research purposes. In this study, professional developers employed in companies chose higher security implementations for password storage than CS students or freelancers. However, similar to CS students and freelancers their security behavior varied when they were not prompted for security. In addition, like the students, professional developers working in companies made rather better security choices with Spring than JSF. In summary, it was demonstrated that the findings regarding relative behavior applied to all developer samples. Since the usable security and privacy community is concerned with increasing secure development rather than with which samples perform better, these are promising results for the ecological validity of developer studies conducted with CS students.

2.5 Programming vs. code reviewing
The previous programming studies were time consuming and the sample size was not as large as we would have wished. Therefore, researchers often tend to design programming tasks in such a way that software developers only need to solve small and short tasks (e.g., [3, 4, 6]). To provide deeper insights into this research field, we tested code reviewing as a promising methodology for security studies with developers. Instead of asking developers to program a piece of code, we showed them functional code snippets and asked them to write code reviews about the snippets. We conducted an online code reviewing study with 44 freelance developers showing each of them an insecure password storage code snippet [13]. Participants needed less time to complete the code review study compared to the programming studies in [11, 21–24] and we still found similar results concerning security awareness and security prompting. While we do not argue to replace programming tasks with code-reviewing tasks in security developer studies, funding is often limited within academia, and smaller tasks yielding similar effects could enable more future research with developers.

2.6 Screener questions
In previous online studies with programmers, researchers often relied on participants’ claims to have programming skills or used programming tasks or knowledge questions to verify these [7, 14, 28]. However, our work showed that designing programming screener questions is not trivial, and we would not recommend using questions without testing them before. We surveyed a total of 249 people to find questions that can be used to filter participants with programming skills. We designed 16 questions and tested them with programmers, non-programmers and under adversarial conditions. We recommend six screener questions for use in online studies based on our evaluation (see [12]). Since the most reliable screeners were also those that took the most time, we extended the pool of screeners and made recommendations on improving the process and introduced time limits allowing us to create more efficient (i.e., quicker but still reliable) screeners (see [10]). Future researchers can use our questions to improve their data quality by screening out participants without any programming skill.

ACKNOWLEDGMENTS
This work was partially funded by the ERC Grant 678341: Frontiers of Usable Security.