On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers

Alena Naiakshina[1], **Anastasia Danilova**[1], **Eva Gerlitz**[2], **Matthew Smith**[1,2]
[1]University of Bonn, [2]Fraunhofer FKIE, Germany
{naiakshi, danilova, gerlitz, smith}@cs.uni-bonn.de

ABSTRACT

Ecological validity is a major concern in usable security studies with developers. Many studies are conducted with computer science (CS) students out of convenience, since recruiting professional software developers in sufficient numbers is very challenging. In a password-storage study, Naiakshina et al. [28] showed that CS students behave similarly to freelance developers recruited online. While this is a promising result for conducting developer studies with students, an open question remains: Do professional developers employed in companies behave similarly as well? To provide more insight into the ecological validity of recruiting students for security developer studies, we replicated the study of Naiakshina et al. with developers from diverse companies in Germany. We found that developers employed in companies performed better than students and freelancers in a direct comparison. However, treatment effects were found to be significant in all groups; the treatment effects on CS students also held for company developers.

Author Keywords

Security Developer Study; Developer Password Study; Usable Security and Privacy; Student Developer

CCS Concepts

•**Human-centered computing** → **Empirical studies in HCI;**
•**Security and privacy** → *Usability in security and privacy;*

INTRODUCTION

Usable security and privacy research that focuses on the end user has been conducted for over 20 years. However, there is only limited knowledge on the human factor in "software development" within the security ecosystem at present [18]. In their research agenda, Acar et al. [5] argued that *ecological validity* issues are a major concern of usable security studies with developers. Similar to end-user goals [7, 40, 39, 48], security is considered as a secondary task by software developers [4, 29, 30, 28]. They are rarely security experts [18], and therefore task design in security developer studies can have a large influence on developers' programming practices [29, 30, 28]. Furthermore, the recruitment of professional software developers for research studies is challenging due to lack of time, spread out geographical locations, and high cost [5, 24, 42, 4, 6, 23, 51]. Consequently, previous conducted security software engineering research recruited computer science (CS) students, who are often studied out of convenience [29, 30, 4, 10, 36, 23, 51]. While there is evidence that students behave similarly to professionals in software engineering studies [20, 11, 25, 44, 38], only limited knowledge exists on whether this holds true for security developer studies. In 2019, Naiakshina et al. [28] conducted a security developer study and found that CS students and freelancers showed similar behavior with regard to secure password storage. One open question remains: Do professional software developers employed in organizations and companies behave similarly to CS students and freelance developers in the security software engineering context as well?

In order to offer more insights into the ecological validity of security developer studies, we replicated the study of Naiakshina et al. [28] with employed professional software developers. We henceforth refer to these people as "company developers." With [28] being a follow-up study of [29, 30], we present in Table 1 an overview of the previous studies and this study. In 2017 and 2018, Naiakshina et al. [29, 30] asked computer science students to program the registration functionality for a university social platform in a laboratory setting. Half of the participants were asked to consider password-storage security (security prompting), while the other half were told the study is about Application Programming Interfaces (API) usability. Additionally, half the participants were advised to use the Java based application framework Spring [2], which offers supporting libraries for secure password storage, while the other half used JavaServer Faces (JSF) [1] without any supporting libraries. Both the variables prompting and framework had a significant effect on secure solutions.[1] Participants using Spring achieved higher security scores than participants using JSF. Further, none of the CS students stored user passwords securely, unless they were prompted. However, some students claimed they would have considered security if they had been given the task in a company. In order to find out whether these results were a study artifact, the authors replicated the study

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6708-0/20/04.
http://dx.doi.org/10.1145/3313831.3376791

[1]After the Bonferroni-Holm correction, the difference between the two groups was not flagged as significant for the framework variable.

	CS Students [29, 30]	Freelancers [28]	Company Developers
Independent Variables (IV)	IV1: Security prompting (yes/no) IV2: Framework (JSF/Spring)	IV1: Security prompting (yes/no) IV2: Payment (100/200 euros)	IV1: Security prompting (yes/no) IV2: Framework (JSF/Spring)
Dependent Variables (DV)	DV1: Secure (yes/no) DV2: Security score (0-7) DV3: API Usability scale from [3] DV4: Functionality (yes/no)	DV1: Secure (yes/no) DV2: Security score (0-7)	DV1: Secure (yes/no) DV2: Security score (0-7) DV3: API Usability scale from [3]
Study Setting	Laboratory study	Field online study	Online study
Study Context and Task	University researchers University social networking platform	Start-up Sports photo sharing social networking platform	University researchers Sports photo sharing social networking platform
Study Announcement	Yes	No	Yes
Recruitment	University	Freelancer.com	Company, Xing
Country of Participants' Residence	Germany	International Primary from India, China or Pakistan	Germany
Compensation	100 euros for programming task and survey	100/200 euros for programming task and 20 euros for survey	400 euros for programming task and survey
Security Request After Initial Submission	No	SecRequest-P if plain text submission	SecRequest-P if plain text submission SecRequest-G if security score < 6

Table 1: Comparison of password-storage studies conducted with CS students [29, 30], freelancers [28], and company developers.

in 2019 with freelance developers recruited online on Freelancer.com [28]. The framework variable was not investigated since all participants were asked to use the JSF framework. Within their pilot studies, the authors recognized that the university context of their study led participants to think they were solving a homework task for university credits. Therefore, the authors changed the study context. This time, the authors claimed to be a start-up who lost a developer and needed help with the registration functionality of their sports photo sharing social network platform. Although the study purpose was not revealed to the freelancers, they behaved similarly to the CS students. The authors concluded that both samples – the CS students in the lab and the freelance developers in the field – behaved the same with regard to security prompting and password-storage practices.

In this paper, we extended Naiakshina et al.'s work by conducting an online study with 36 regularly employed professional software developers. Like Naiakshina et al. [28], we told half the participants to store end-user passwords securely, while the other half were told the study focused on API usability. In order to ensure that our study appeared real and meaningful to the subjects (experimental realism) [42], we refrained from the university context and opted for the start-up scenario. We adopted the study design of the freelancer study [28] with some necessary modifications. First, in the context of a freelance service, it was possible to hire developers for the programming task without revealing the study purpose. However, it was not realistic to maintain this pretense for regular professional software developers. Thus, we announced the project as a study. In the Methodology section we discuss the reasons for this in detail. Second, like Naiakshina et al. [30], we examined the variable framework with the new sample of professional developers.

With this paper, we contribute on a primary level by comparing password-storage practices across different developer samples: students, freelancers, and company developers. In addition, we discuss methodological implications for security developer studies on a meta-level. As suggested by Sjoberg et al. [42], for the primary level analysis, we compared the results of students, freelancers, and professionals in *absolute* terms. For the meta-level analysis, we compared the results in *relative* terms. Based on the primary level analysis, we found that company developers overall performed better than students with regard to security measures. However, the meta-level analysis showed that the treatment effects of task design and framework were found to be significant in all groups. We refer to treatments as "prompting" vs. the baseline "non-prompting" and the Spring framework vs. the baseline plain Java (JSF).

RELATED WORK

While observations of studies conducted with students are assumed to apply to professional developers in the domain of software engineering [20, 11, 44, 38], it is not known for certain yet on whether this holds true for security developer studies [51, 4, 6, 23, 28]. To place our study in context, the following section is divided into three parts. First, we outline security studies with students. Second, we discuss related work in the area of security studies with professional developers. Finally, we summarize those works that compared behavior of students and professional developers in security studies. Most important for our work are the studies conducted by Naiakshina et al. with freelance developers [28] and CS students [29, 30], which were described in the Introduction and with this study being a replication of [28] with minor changes as described in the Methodology section.

Security Studies with Students

As the recruitment of professional software developers is often very challenging due to high costs or lack of time, many studies have instead been conducted with CS students. For example, Thomas et al. [46] studied if interactive annotation was useful for developers in indicating access control logic and understanding the ensuing reported security vulnerabilities. They recruited 28 CS students and found that participants struggled to use the researchers' tools to trace the cause of vulnerabilities.

Jain et al. [21] explored developers' behavior regarding location privacy in mobile applications when provided with different location APIs. They recruited 25 CS students to work on three programming tasks and found that participants preferred the more privacy-friendly API when given a choice. Furthermore, to understand how developers use error-messages, Barik

et al. [10] conducted an eye-tracker-study with 56 students who were asked to resolve defects in Java code. The authors found that reading error messages was as difficult as reading source code. Both Jain et al. [21] and Barik et al. [10] argued to specifically have recruited students to avoid the unrealized bias of professionals being familiar with APIs.
Layman et al. [26] recruited 18 CS students to investigate factors influencing developers when deciding whether or not to fix a fault after being notified by an automated fault detection. Based on their finding, the authors provided recommendations regarding automated fault detection tools. Furthermore, Scandariato et al. [41] conducted a controlled experiment with 9 graduate students and found static analysis to uncover more vulnerabilities in a shorter time in comparison to penetration testing. In order to approve the external validity of their findings, Layman et al. [26] and Scandariato et al. [41] advised to recruit professional developers for future work.

Security Studies with Developers

Myers and Stylos [27] summarized related work on evaluating API usability and highlighted that usability issues with APIs can affect the security of software. Thus, they encouraged to conduct user studies. One related usability study on password storage was conducted by Wijayarathna and Arachchilage [49] with 10 GitHub developers. The authors evaluated the usability of Bouncycastle API's functionality of the password hashing function `SCrypt`. On observing 63 usability issues, the authors found that most of the difficulties arose from the complexity of the method's parameters. In accordance with our findings, they concluded that developers usually are not security experts and thus they should be provided appropriate guidelines, especially in organizational scenarios.
Another study analyzing the security, reliability, and functionality of web applications was conducted by Prechelt [37] with 27 experienced developers. Three groups of participants had to implement similar tasks using different platforms. PHP showed fewer noticeable variations than Perl or Java EE. Trying to find out more reasons why programmers make security errors, Jing et al. [50] conducted 15 semi-structured interviews with developers who were experienced in Java, C++, C, Python, PHP, and JavaScript. The authors implied that developers need more interactive tools to assist them in implementing security measures.
Johnson et al. [22] conducted 20 interviews with professional developers and found that false positives and poor warning visualization are reasons for not using static analysis tools. Furthermore, Thomas et al. [47] conducted a study with 13 participants with professional development expertise and suggested a tool for effectively communicating security vulnerabilities.
Balebako et al. [9] conducted an online survey with 228 app developers and found that smaller companies (<=30 employees) were less likely to demonstrate positive privacy and security behaviors compared to larger companies with 31-100 or 100+ employees. By contrast, Assal et al. [8] conducted an online survey with 123 software developers and found no evidence that company size has an influence on participants' behaviors and attitudes towards software security. However, a workplace environment that nurtures security was more motivating to consider security for participants in larger companies, compared to those in smaller companies. We, therefore, also examined whether the size or security focus of a company affected participants' decision to store user passwords securely, but found no significant effect.

Student and Developer Comparison in Security Studies

Acar et al. [4] conducted a lab study with 54 Android developers to explore the impact of information sources on code security and found that professionals produced more functional code than students. However, with regard to security, no significant effect was found. In a further study of Acar et al. [6], 307 GitHub users were given short security related tasks, inter alia, to store user passwords in a database. The authors found that neither student nor professional status (self-reported) was a significant factor for functionality, security, security background or security perception.
Krombholz et al. [23] attempted to find reasons for weak TLS configurations. Their expert interviews with security auditors underlined the ecological validity of their results from a lab study conducted with students.
What is more, Nguyen et al. [36] recruited skilled students and professional developers to examine whether an IDE plug-in could help developers write more secure Android applications. Except for the professional developers' smaller motivation in taking part in their study, the authors did not find any significant differences between the samples.
Yakdan et al. [51] measured the quality of decompilers for malware analysts by asking 21 students and 9 professional malware analysts to work on reverse engineering tasks using three decompilers. Unlike in the previously-mentioned papers, they noticed significant differences in the performance of students and professionals. However, the overall assumption of which decompiler performed best remained the same for both samples.
In contrast to the previous studies and in accordance with Yakdan et al.'s [51] findings, our study results showed that professionals' and students' security behavior was different in absolute terms. However, we found similarities in relative terms.

METHODOLOGY

We conducted an online study of password storage with 36 professional software developers employed by different companies and organizations. We adopted the methodology and study design frame from a previous study of password storage with freelancers [28]; that study replicated yet a lab study with computer science students [30]. These studies were described in more detail in the introduction.

The recruitment of professional developers posed new challenges not faced in Naiakshina et al.'s [28] study of freelancers, but it presented new opportunities as well. Some adjustments were made to the study frame for this specialized sample. In order to ensure that our study design decisions were reasonable, we consulted an IT-company with 250 employees from Germany. To preserve confidentiality, this company is here referred to as "ITXcompany." We opted for the same online study design used in the freelancers study [28]. In that study, freelancers were hired to work on the study and could do so

during their normal working hours. In our study, however, company developers were not supposed to take part in studies during their working hours; they had to participate during free time after work or on weekends. The resulting changes to the study design will be described in detail in the next section. Overview of the study context of the previous studies and the present one are provided in Table 1.

Study Design Changes

Study Announcement. In our study, it would have not been realistic to claim that we are a start-up searching for a regularly employed developer to do a small programming job of about one working day for several reasons. First, in the frame of a freelance service, it is authentic to hire developers for solving a rather small programming task without study announcement. By contrast, a start-up would rather search with an advertisement for a permanent employee rather than contacting employees at a company. Second, we consulted ITXcompany and agreed that, even if the programming task would be assigned by the management board to several employees, the likelihood that those employees would exchange information about the study was rather high. This would have consequently invalidated our study results. It would be particularly suspicious if multiple developers working for the same organization were asked to do the same job. Therefore, we had to declare that our project was a study. To keep the study as similar as possible to the model study [28], we used the same task as that in Naiakshina et al.'s role-playing scenario (see Table 1). Participants were also provided with the startup's web presence. The task is described in detail in the supplementary materials.

Study Compensation. In the original study [30], students needed about 6-8 hours to finish the project. Thus, Naiakshina et al. [28] announced the project as a freelance job that would take 8 hours. Based on freelancers' varied compensation proposals, Naiakshina et al. explored the effect of two payment levels: Participants were paid either 100 or 200 euros for the project (see Table 1). Freelancers took a mean of 3.2 days to submit their solutions, though their precise hours of total work on the task were not clear. The authors found that the payment variable had no significant effect. Participants in the freelancer study came primarily from India, China, or Pakistan. In our study, we recruited regular software developers from Germany and paid them 400 euros. Since the salary of German developers is assumed to be higher, determining appropriate study compensation was especially challenging. We again consulted ITXcompany about a fair pay. We wanted the compensation to be high enough to motivate well-paid company software developers to take part in a rather large-scale programming study – which differs from the more common type of study consisting of a short task and survey – in their free time. Still, the compensation also needed to be converged by ordinary research funding. We finally opted for one payment level and agreed with ITXcompany on a compensation of 400 euros. Following the approach of the freelancer study [28], we did not require the task to be finished by a deadline. Rather, we wanted to know how long regular developers would need to complete the project outside their usual working hours. Following the previous studies [30, 28], our participants were informed that the project was estimated to take 6-8 hours but that they could work on the task whenever convenient.

In the final survey, we asked our participants to rate the compensation for the study; 86% (31/36) thought the payment was just right, and 8.3% (3/36) felt the payment was too much. One participant indicated the payment was too low, and another indicated that it was way too low[2] (2.8% each).

Framework Security Support. In the original study with students, Naiakshina et al. [30] found that participants received a higher security score when they used Spring rather than JSF. The authors concluded that this was because Spring offers a security library import with secure parameter defaults and automatic salt generation. Furthermore, students copied and pasted solutions from the Internet; these solutions were often provided by tutorials using the default library support. In contrast, participants who used JSF needed to manually implement secure password storage, so they had to determine secure parameters on their own. While all participants in the freelancer study [28] used JSF, we wondered whether the different levels of framework security support would affect the company developers' programming solutions in a similar way as that observed in the student group. Thus, like the student studies in [30, 29], we asked our participants to use either Spring or JSF to solve the task (see IV in Table 1).

Final Study

We assumed that the results with students would differ from those with professionals in absolute terms (primary level analysis). However, we were interested in whether the *relative* difference between student and professional results would be comparable (meta-level analysis). We randomly assigned participants to one of four conditions: **P**rompting-**J**SF (PJ), **N**on-Prompting-**J**SF (NJ), **P**rompting-**S**pring (PS), **N**on-Prompting-**S**pring (NS). We accepted only functional submissions and tested all solutions in our system. To score the security of the developers' code, we adopted the extended version of the security scale used by Naiakshina et al. [28]. As displayed in Table 1 (see DV1 and DV2), the scoring system contains a binary variable *secure* indicating whether participants used any kind of security in their code and an ordinal variable *score*[3] to score how well they did. The score could range from 0 to 7; the following factors were considered: whether and what hash algorithm was used (0-3 points), the iteration count for key stretching (0-1 points), and whether and how the salt was generated (0-3 points). The detailed security scale can be found in the supplemental materials.

Security Request. As shown in Table 1, freelancers who submitted plain text solutions were requested to revise their submissions and to store user passwords securely (*SecRequest-P[laintext]*). The security requirements (prompts in the task description and *SecRequest-P*) used in the previous studies did not provide further information on password-storage security. Acar et al. [4] demonstrated, however, that information

[2]This participant, however, indicated to have actively worked 20 hours on the task, more than any other participant.

[3]Previously called *security* [29, 28]. In order to avoid confusion between *secure* and *security*, the name of this variable security was changed to *score*.

General Information:			
Gender	Male: 33	Female: 2	Prefer not to answer: 1
Age	min: 25, max: 54	mean: 36.64, median: 36	SD: 7.7
University Degree	Yes: 29	No: 7	
Profession	Software developer: 30	Other: 6	
Nationality	German: 28	German + other: 4	Spanish, French: 1 each, NA: 2
Professional Experience:			
General Development Experience [years]	min: 1, max: 30	mean: 12.92, median: 14	SD: 7.9
Java Experience [years]	min: 1, max: 21	mean: 9.42, median: 10	SD: 5.28
Organization Information:			
Organization Age [years]	min: 2, max: 200	mean: 49.7, median: 29	SD: 49.6
Organization Size	1-9: 1	10-249: 11	250-499: 3
	500-999: 2	1 000 or more: 19	
Organization with Security Focus	Yes: 12	No: 24	

Table 2: Demographics of the 36 participants

sources can affect developers' security related programming practices. We wondered whether providing more precise task requirements, such as suggesting a popular password-storage security related information source would help developers to follow security best practices. Participants whose solutions scored lower than 6,[4] received another security request (*SecRequest-G[uideline]*) asking them to follow the recommendations of the National Institute of Standards and Technology (NIST) [17] or the Open Web Application Security Project (OWASP) [32]. While NIST's official documents are rather long and difficult to access, OWASP provides an overview of state-of-the-art recommended hashing functions and an example of an implementation of `Argon2` using Java code. Therefore, we expected that the NIST documentation would be secure but difficult to use and that OWASP documentation would be secure and relatively ease to use.

After participants submitted their final solutions, they were invited to take part in an online survey, an extended version of the survey used in the previous studies [30, 28]. The survey focused on participants' experience with the task, their knowledge of security and IT, their company background, and how well the study task represented their field of work. The security requests, the security request procedure, and the final survey can be found in the supplementary materials.

Participants

We recruited 75 software developers through several channels, of whom a total of 36 employed developers completed the study. The recruitment of a reasonable number of professional software developers for quantitative research studies is challenging [5, 24, 28, 8, 6, 23], especially for a study as long as ours. The participants we invited to join our study, are only those who had at least one year of programming experience with Java and were regularly working in companies or organizations. To establish a comparison between studies with full-time working developers and with those who are freelancers, and students, we filtered out freelancers, full-time students, and unemployed software developers from our sample. We started recruitment in ITXcompany. Some developers of this company also recommended our study to their colleagues and friends. Of the 45 Java software developers from the company, 15 showed interest in participating in our study. Three of them were sorted out because of their student status. Twelve developers were then invited to the study, and they signed the consent form. However, 7 developers dropped out because of a lack of time and an additional one also dropped out because he did not manage to solve the task in a functional way. Finally, only a total of 4 developers of ITXcompany completed the study. Among the developers' colleagues and friends, 8 showed interest, 6 of whom were invited to the study; the other 2 participants were sorted out because of their student status. One submission was discarded for non-functional reasons, leaving us with 5 valid participants' sumbissions from the referrals.

We continued our recruitment through XING [35], a German career-oriented social networking site similar to the American platform LinkedIn [31]; we posted the project in forums for Java, Java development, Java user groups, and the job market. Forty-nine participants recruited via XING showed interest in our study, 11 of whom were sorted out for requirement reasons. Thirty-eight participants were therefore invited to the study, and 34 signed the consent form. Finally, a total of 25 participants recruited via XING completed the study. Additionally, we contacted further professional and industry contacts, of whom three registered to the study. One was sorted out for participant requirement reasons, so two signed the consent form and completed the study. The data reported herein is those for the remaining 36 participants.

The participants' demographic information is shown in Table 2. The 36 participants reported ages between 25 and 54 (mean: 36.64, median: 36, SD: 7.7), and almost all of them were male (33 males, 2 females). On average our participants had been programming for 12.92 years (min: 1, max: 30, median: 14, SD: 7.9) and in Java for 9.42 years (min: 1, max: 21, median: 10, SD: 5.28).[5] Nineteen participants worked in companies with 1 000 or more employees. Participants' team size were on average 13.1 (median: 8, min: 2, max: 50, SD:

[4]Six of possible 7 points was the highest score actually achieved in both the freelance and student group. Neither group became aware of the state-of-the-art recommendations, indicating that memory-hard hashing functions are necessary for security best practices [12, 19, 32, 17]. However, solutions which earned 6 points followed industry best practices. In accordance with previous studies, we did not want participants to be penalized for using the framework's standards, so the Spring `BCrypt` default parameters were judged to be secure.

[5]No significant demographic differences were found between the 4 condition groups in terms of general and Java programming experience.

Factor	Description	Baseline
Prompting	Whether the participant is asked to store the password securely	False
Sample	Student, freelancer or company developer	Company developer
Framework	Spring or JSF	JSF
Java experience	Years of Java experience	n/a

Table 3: Factors used in our regression models. To select the final models we chose the minimum AIC.

11.8). Twelve participants reported that their organization has a security focus, of whom 6 also explicitly indicated to work in a team with a security focus. One additional participant worked in a company without, but in a team with a security focus. Further information on the participants' demographic information is available in the supplementary materials.

Evaluation

Code Analysis

To ensure data reliability [16, 43], three coders independently reviewed all programming code and evaluated them for security. In three cases, disagreement occurred with respect to the parameter specifications of algorithms– the iteration defaults of algorithms, whether an algorithm used the classes Random [33] or SecureRandom [34] for salt generation, and how static salts are rated. Disagreement was resolved by consulting a security expert and discussing algorithm specifications. Because of the rigid rules of the scoring system and the strict algorithm specifications, full agreement was achieved.

Statistical Analysis

Because of the adjusted study design, we were able to test 4 of the 7 main hypotheses from [30]. In particular, we tested whether prompting (H-P1), framework (H-F1), years of Java experience (H-G1), and password storage experience (H-G2) had an effect on security.[6] A detailed description of the hypotheses can be found in the supplementary materials. For the data analysis, we used the same tests as those utilized in [30, 28]. All tests referring to the same dependent variable were corrected using the Bonferroni-Holm correction. Bonferroni-Holm corrected tests are labelled with "family = N", where N is referring to the family size. Thus, we report both the initial and corrected p-values (cor-p).

Furthermore, we obtained different iterations of the code submissions from our participants because some received security requests to improve their code (see Section Final Study). To compare the results of this developer study with those of the student and freelancer sample, we used the code submission in the first iteration before any additional security requests were made. We utilized regression models to conduct an overall analysis of all studies, including those of Naiakshina et al. [30, 28] and this replication study. For the binary outcome (secure), we used logistic regression, and for the continuous outcome (score), we used linear models. To find the best combination of factors, we selected the model with the lowest Akaike information criterion (AIC) [13]. All factors for our regression analysis are summarized in Table 3. For all regressions, we selected as final the model with the lowest AIC.

[6]While it was possible to track security attempts of the student sample in a lab setting, this information was not accessible in an online study. We, therefore, considered the subset *secure = 1* (achieving security) for our analysis.

Qualitative Analysis

We analyzed the qualitative data from the open-ended questions in the survey through *inductive coding* [45]. Two researchers independently searched for codes and categories emerging from the raw data. After the coding process was completed, the codes were compared and inter-coder agreement by using Cohen's kappa coefficient (κ) [14] was calculated. The agreement was 0.83. A value above 0.75 suggests a high level of coding agreement [15].

Limitations

Our study has limitations that need to be considered when interpreting the results. The study was conducted online and we thus had less control over the study compared to the lab study by Naiakshina et al. [29]. We opted for the online study to reduce the difficulties in recruiting company developers, since it gives the participants the option to work at a time of their own choosing. None the less, we found recruiting a high number of employed developers for a one-working-day study (approximately 8-hours) outside their regular job time extremely difficult and thus our sample size is not as large as we would have wished.

Moreover, when comparing the different studies several caveats must be taken into account. The students sample had a time limit of 8 hours to complete the task in the lab, while freelancers and our company developers had no time restrictions to complete the task. We did not set deadlines for task completion since we hoped to motivate more employed developers to take part in the study. However, the developers reported to have spent an average of 7 hours actively working on the task. Also the study task was framed differently in the three studies. The study with university students used a university context while the freelancer and our study used a company context. Furthermore, our sample consisted of developers working in Germany, which means that similar studies in other countries could lead to different results. Because this is a replication of a study conducted with freelancers covering already a wide range of cultural backgrounds, we did not find significant differences in the results between the freelancers and our company developers in this context.

Finally, our study task is only one example case in one programming language and thus further studies are needed to see if our results replicate in other cases as well, e.g., by examining other security tasks, characteristics of developer groups, frameworks, and programming languages.

	Non-secure	Secure	Score	Total
NJ	6	2	$\mu = 1$ ($\sigma = 1.85$) min = 0, max = 4	8
NS	5	2	$\mu = 1.71$ ($\sigma = 2.93$) min = 0, max = 6	7
PJ	1	9	$\mu = 4.8$ ($\sigma = 1.75$) min = 0, max = 6	10
PS	0	11	$\mu = 5.86$ $\sigma = 0.45$) min = 4.5, max = 6	11
Total	12	24	$\mu = 3.68$ ($\sigma = 2.69$)	**36**

(a) **Initial Solution, n = 36**

	Non-secure	Secure	Score	Total
NJ	2	4	$\mu = 1.92$ ($\sigma = 2.25$) min = 0, max = 5	6
NS	0	5	$\mu = 5.4$ ($\sigma = 1.34$) min = 3, max = 6	5
PJ	-	-	- -	-
PS	-	-	- -	-
Total	2	9	$\mu = 3.5$ ($\sigma = 2.56$)	**11**

(b) **SecRequest-P, n = 11**

	Non-secure	Secure	Score	Total
NJ	0	8	$\mu = 6.13$ ($\sigma = 1.73$) min = 2, max = 7	8
NS	0	1	$\mu = 7$ ($\sigma = 0$) min = 7, max = 7	1
PJ	0	7	$\mu = 6.5$ ($\sigma = 0.87$) min = 5, max = 7	7
PS	0	1	$\mu = 7$ ($\sigma = 0$) min = 7, max = 7	1
Total	0	17	$\mu = 6.38$ ($\sigma = 1.29$)	**17**

(c) **SecRequest-G, n = 17**

Table 4: Number of (non-)secure solutions and security score per condition and per security request
NJ = Non-prompting JSF; **NS** = Non-prompting Spring; **PJ** = Prompting-JSF; **PS** = Prompting-Spring

Ethics

Our project was approved by the institutional review board of our university. At the beginning of our study the participants were asked to download the consent form and provide their consent, complying with the General Data Protection Regulations. Participants were informed about the practices used to process and store their data and that they could withdraw their data during or after the study without any consequences. We ensured all participants that the information about their performance would be kept confidential both within their company and outside, and that only anonymized data would be published. Additionally, we ensured all our subjects that they would be informed about the results of our study.

One variable of our study included deception by not prompting participants for security. The feedback of our participants regarding our study and survey, though, was positive overall. None of the non-prompted participants reported to feel deceived or expressed any negative feelings. Three of them, however, wished to have been informed about security requests in the initial task description. After assessing the security score, one prompted participant stated that the security request for industry standards should be included in the task requirements because it took more time for him to complete the task than expected. Indeed, the estimated time calculation of 6 to 8 hours to complete the project was based on the previous studies with students and freelancers [30, 28]. Due to our new security request SecRequest-G on state-of-the-art security, participants might have needed more time to read the information sources and apply security measures than we expected. However, similar to students in the lab, company developers reported to have actively worked an average 7 hours on the task (min: 2, max: 20, median: 6, SD: 4.72).

RESULTS

Table 4 shows an overview of participants' submissions. On average, initial solutions were submitted after 8 days (min: 16.5 hours, max: 26 days, median: 8 days, SD: 6.5 days). To submit a solution after SecRequest-P, participants needed, on average, 1.5 days (min: 3.5 hours, max: 9 days, median: 12 hours, SD: 2.5 days) and after SecRequest-G, they needed an average of 4 days (min: 2 hours, max: 14.5 days, median: 2 days, SD: 4 days).

In total, we received 36 submissions of which 12 were non-secure and 24 were secure (used at least a hashing function). Eleven of the 12 non-secure solutions were submitted in plain text and thus these participants received SecRequest-P (see Table 4b). Only participants from the non-prompted group (NJ and NS) received SecRequest-P and yet 2 of the 6 participants who used JSF subsequently submitted a non-secure solution. Seventeen participants (8 prompted and 9 non-prompted) in total did not receive at least 6 points for their initial or revised submission after SecRequest-P and thus received SecRequest-G (see Table 4c). Two of these participants were from the Spring group (NS and PS) and 15 were from the JSF group (NJ and PJ). In comparison to participants using JSF, participants using Spring were more likely to submit solutions with at least 6 points. After receiving SecRequest-G, all participants delivered a secure solution with an average score of 6.38 points (SD: 1.29).

Within participants' initial or successive submission after SecRequest-P, the highest security score achieved in all the groups was 6 (of 7) with no participants using a memory-hard hashing function. While participants using Spring tended to utilize its opt-in functionality with `BCrypt` as the preferred hashing function for storing user passwords securely and thus received 6 points for security (16/18), participants using JSF used a variety of methods. Two participants used symmetric encryption and received 0 points. Another two participants used `MD5` without salt and received only 1 of 7 points for security. Four participants used `SHA-2/3` with salt and received 4-5 of 7 points. Seven of the 18 JSF users employed `PBKDF2` as a hashing function, but only one received 6 points for her/his parameter choice. Five of these participants used 128 bits as the output length for the hashing algorithm, which was an unusual choice for password storage. These results suggested that those participants had copied and pasted code from the Internet where `PBKDF2` was used for its initial purpose, a password-based encryption of a string. The final two participants using JSF employed `BCrypt` as their preferred hashing function and received 6 points for security.

Of the 17 participants who achieved a lower score than 6 and thus received SecRequest-G, 12 used `Argon2` and thus received the full 7 points for security. Of these participants, 7 indicated having used OWASP as an information source, suggesting that they adopted the available Java example of an `Argon2` implementation. Indeed, we found 6 `Argon2` submissions obviously copied and pasted from the OWASP source, which we were able to identify based on the same comments present on the website and participants' code. In addition, we observed 2 participants (PJ3 and PJ5) to have stored an

additional salt in the database. It seems these participants were not aware that the `Argon2` implementation generated a salt by default. Finally, by using `BCrypt`, two other solutions scored 6 points for security. Only three participants used `PBKDF2` or `SHA-3` and received at most 5.5 of 7 points for their parameter choices. These results indicate that security requests providing a secure-but-usable information source can lead to higher software security. A more detailed evaluation of individual submissions is available in the supplementary materials.

Prompting (H-P1)

We explored the effect of our task description variable (prompting vs. non-prompting) on whether the participants decided to store the passwords securely. Table 4a shows that in their initial solutions, the majority of the non-prompted developers (NJ and NS) did not store user passwords securely (11 out of 15). Only 4 participants stored the passwords securely without being prompted. For the prompting conditions (PJ and PS), only one participant stored user passwords insecurely, whereas 20 participants stored the passwords securely. Thus, the prompting task showed a significant effect (FET: p < 0.001*, $cor - p$ < 0.001*, OR = 46.33, CI = [4.74, 2434.13], family = 2).

Experience (H-G1, H-G2)

We tested whether Java experience had an effect on the security scores of our participants. We found no significant effect of Java experience on the initial security scores (Kruskal-Wallis: χ^2 = 11.9, df = 15, p = 0.69, $cor - p$ = 0.69, family = 2). We also found that Java experience had no effect on the scores considering only secure solutions (group: secure = 1; Kruskal-Wallis: χ^2 = 10.6, df = 13, p = 0.64). We further investigated whether experience with password storage had an effect on whether the passwords were stored securely. Of the 26 participants who indicated to have stored user passwords in a database before, 10 initially submitted an insecure solution. Of the 10 participants who indicated they had never stored passwords before, 8 submitted a secure solution as their initial submission. Thus, we found no significant effect of previous experience with password storage (FET: p = 0.44, odds ratio = 0.41, CI = [0.04, 2.69], family = 2, $cor - p$ = 0.44).

Framework (H-F1)

In the initial submissions, we found that the security scores achieved in the JSF and Spring groups differed significantly (group: secure = 1; Wilcoxon Rank sum: W = 27, p = 0.003*, family = 2, $cor - p$ = 0.006*). The mean score for the JSF group was 5.09 (group: secure = 1; min: 4, max: 6, median: 5, SD: 0.7) and 5.89 for the Spring group (group: secure = 1; min: 4.5, max: 6, median: 6, SD: 0.42), indicating that participants from the Spring group achieved higher scores than the participants from the JSF group.

Similar to Naiakshina et al. [30], we calculated the API usability scores as suggested by Acar et al. [3] for both groups and found that the Spring group achieved higher usability scores (mean: 68.06, median: 67.5, SD: 12.44) than the JSF group (mean: 58.61, median: 56.25, SD: 18.71). However, the difference was not significant (Wilcoxon Rank sum: W = 106, p = 0.08). Furthermore, we found no correlation between the usability score and the security score achieved by our participants (Pearson, r = -0.11, p = 0.54).

Company Size and Security Focus

Similar to Assal et al. [8], we classified our participants' organization in two size categories: Small and Medium Enterprises (SME: less than 500 employees), and Large Enterprises (LE, more than 500 employees). Of our participants, 58% (21/36) indicated that they work for a LE, whereas 42% (15/36) reported to work for a SME. We found that company size had no effect on whether or not participants decided to store user passwords securely (FET: p = 1, odds ratio = 1, CI = [0.19, 4.99]). We also tested whether a focus on security by the participants' team or company had effect on security behavior. We found no effect for either team security focus (FET: p = 1, odds ratio = 1.31, CI = [0.17, 16.06] or company security focus (FET: p = 0.16, odds ratio = 0.34, CI = [0.06, 1.83]. Finally, we analyzed the participants' responses to the question of whether they solve security tasks during their working routine. Twenty-one participants reported to solve security tasks regularly, while 15 indicated that they did not solve any security tasks during their working routine. We found that having to solve security tasks as part of their work routine had no effect on the participants' security behavior (FET: p = 0.72, odds ratio = 0.60, CI = [0.10, 3.03]).

Qualitative Analysis

We evaluated the responses to the open-ended question of the survey to get more insight into participants' rationale behind their decisions. The coding overview can be found in the supplemental materials. Supporting our previous results, we found that besides standards and experience, developers mostly rely on requirements when implementing security. In particular, participants argued that there is a trade-off between requirements and effort (NJ4 - NJ7); if security is not explicitly requested, there is no personal benefit in dealing with it. Participants felt particularly insecure whether the used security methods are sufficiently secure for several reasons. First, they stressed that security is not part of their everyday development work (PJ5, PS8) and that they lack knowledge in this field (PJ7, NJ8). Some further suggested that there is a lot of outdated information on the Internet with respect to security practices (PJ6, PS8), making Internet research rather challenging. Participants preferred to consult more experienced colleagues (PS6, NS1) or other security entities (PJ10, NJ2, NJ7, PS5, PS8) to sufficiently fulfill security requirements.

Additionally, our assumption that OWASP was perceived as a more usable information source than NIST was confirmed. Of the 17 participants who received SecRequest-G, 7 reported to have used OWASP, 3 NIST (PJ4, PJ9, NJ2) and another 3 used both information sources (PJ10, NJ4, NJ45). The reasons provided by participants for their preferred use of OWASP were the practical code example of Argon2 (NS4, PJ5, PJ8, NJ3, NJ6), the familiarity with the organization (PS7, NJ3, NJ7) and the fact that it is open source (PJ3). Furthermore, PJ5 and PS3 reported to mistrust NIST because of its US origin and possibly providing back door access for the National Security Agency (NSA). NJ2 and DJP4, however, reported that the NIST source was easy to use and provided clear requirements. By contrast, PJ5 found both sources to be inconsistent with the requirements, too complicated, and difficult to follow.

	Students n = 40		Freelancers n = 42		Company Developers n = 36	
	Non-secure	Secure	Non-secure	Secure	Non-secure	Secure
Non-Prompting	20	0	17	4	11	4
Prompting	8	12	8	13	1	20

Table 5: Number of secure solutions per condition

Factor	O.R.	C.I.	p-value
Prompting	29.89	[9.08, 98.44]	<0.001*
Students	0.08	[0.01, 0.48]	0.006*
Freelancers	0.23	[0.05, 1.05]	0.06
Spring	1.75	[0.5, 6.19]	0.38
Java Experience	0.96	[0.83, 1.11]	0.58

Table 6
Logistic regression whether the initial solution is secure. Odds ratios (O.R.) estimate relative likelihood of succeeding. Baseline factors: company developer sample, JSF, and non-prompting. Nagelkerke R^2= 0.55.

Factor	Estimates	C.I.	p-value
Prompting	2.92	[2.15, 3.69]	<0.001*
Students	-1.98	[-3.2, -0.75]	0.002*
Freelancers	-1.54	[-2.66, -0.43]	0.008*
Spring	0.94	[-0.01, 1.9]	0.06
Java Experience	-0.03	[-0.13, 0.08]	0.65

Table 7
Linear regression on the score of developers' results using the factors prompting, sample, framework, and experience. Baseline factors: company developer sample, JSF, and non-prompting. R^2= 0.44.

Primary Level Analysis Across Samples

For our primary analysis, we compared the password-storage results of students [30], freelancers [28] and company developers. For this comparison we only considered the initial submissions of participants. We were able to compare the security results based on two variables: prompting and framework. By contrast, the API usability was measured in all groups on different stages of the study (students: after initial submission; freelancers: e.g., after SecRequest-P; company developers: e.g., after SecRequest-G). Therefore, no comparison of the API usability across samples was conducted.

Table 5 summarizes how many participants from the prompted and non-prompted conditions submitted a secure solution. Similar to freelancers and students, most company developers who were not prompted did not submit a secure solution for password storage, while the majority of those who were prompted did. However, unlike the students and freelancers, more company developers in total were able to solve the task securely. Table 6 shows the results of the logistic regression on whether participants submitted secure solutions. The model includes the following factors: prompting, the freelance or student status, Spring, and Java experience. Prompting and students were significant factors in the regression model, demonstrating an effect on whether a participant implemented security. Prompting was associated with approximately 30× higher odds that the participants' solution would include secure storage of user passwords. In comparing samples, student participants were only 0.08× as likely to store user passwords securely as company developers. Java experience, framework and freelance status did not show a significant effect in our model.

With regard to the security scale, a variety of security scores were observed. Figure 1 shows the security scores of initial submissions across different frameworks and samples. In the JSF conditions, company developers achieved higher scores than freelancers and students. In the Spring conditions, the developers' and students' scores were almost the same. Freelancers were not tested for the Spring conditions, although they showed a greater variety of scores for JSF compared to company developers and students. Table 7 shows the results of the regression model (F(5,107) = 16.83) for the security scores and includes the factors of freelance or student status, prompting, Spring, and Java experience. Prompting and sample were significant while Java experience and framework had no significant effect on the score. Participants that were prompted for secure password storage achieved a higher score (on average 2.92 more points) than participants who were not prompted. On average, students received 1.98 and freelancers 1.54 less points than company developers.

Meta-level Analysis Across Samples

Table 8 summarizes our findings of the meta-level analysis. The conditions of being prompted in regards to security showed an effect on CS students, freelancers, and company developers. Java or password storage experience had no effect on security in the student, freelancer, and company developer group. However, a treatment effect of framework was found to be significant in the developer as well as in the student sample (because they only used JSF, freelancers were not considered for the treatment framework). As Figure 1 demonstrates, students in the Spring group achieved higher security scores than students using JSF. Similarly, company developers in the Spring group achieved higher security scores than company developers in the JSF conditions. Moreover, 2 of the company developer submissions had to be discarded for non-functional reasons. Although all other company developers were able to submit functional solutions, some reported to struggle with functionality issues similar to the students.

DISCUSSION

Ecological validity issues are a major concern of usable security studies with developers [5]. While there is evidence that students behave similar to professionals in software engineering studies [20, 11, 25, 44, 38], limited knowledge exist whether this holds true for security developer studies. Our

IV	DV	St.Test	Students [30]	Freelancers [28]	Company Developers
Prompting	Secure	FET	$p <$ **0.001*** O.R. = ∞, C.I = [5.06, ∞]	$p =$ **0.01*** O.R. = 6.55, C.I. = [1.44, 37.04]	$p <$ **0.001*** O.R. = 46.33, C.I. = [4.74, 2434.13]
Java Experience	Score	Kruskal-Wallis	$p = 0.249$	$p = 0.21$	$p = 0.69$
Stored Passwords Before	Secure	FET	$p = 0.297$ O.R. = 2.54, C.I = [0.49, 17.72]	$p = 0.52$ O.R. = 0, C.I. = [0,8.91]	$p = 0.44$ O.R. = 0.41, C.I. = [0.04, 2.69]
Framework	Score	Wilcoxon Rank sum	$p =$ **0.03*** group: secure = 1	- -	$p =$ **0.003*** group: secure = 1

Table 8: Summary of all tests across different samples
The IV framework was not examined for freelancers. In the student sample, the originally examined group was attempted security = 1 (for IV prompting and framework).

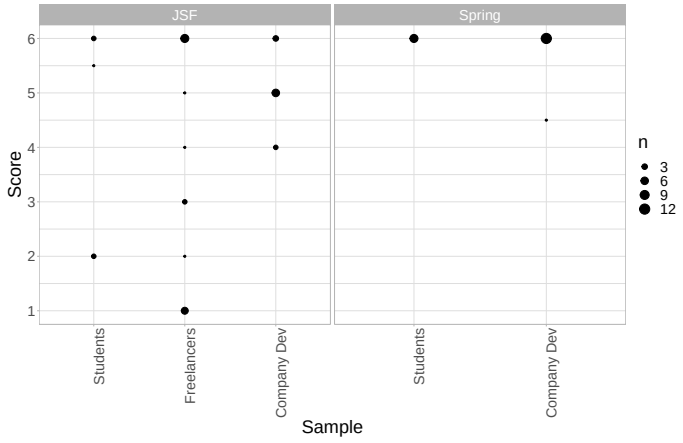

Figure 1: Scores across samples (secure = 1)
Company dev = company developer. The size of the points symbolizes how many participants from a sample gained the corresponding scores considering initial and secure solutions only. In the freelancer sample only the JSF framework was tested.

regression analysis on password-storage security showed that company developer and student results differed in absolute terms. This means that company developers produced more secure solutions than students. Since years of Java experience was not a significant predictor for a higher security score, other factors need to be taken into account for an exploration of the question: Why did company developers perform better than students? Our results suggest that the company context of developers could be the key. In our survey, 17 of 36 participants reported company project experience, company training, and exchange with colleagues to be their main IT security source of knowledge. Only 8 referred to university, which is probably the main source of knowledge for most students. Additionally, 13 company developers indicated to work in a company/team with security focus and 21 to be involved in security relevant projects. However, more research will be needed to answer this question in more depth.

As suggested by Sjoberg et al. [42], we considered the comparison of students' and professionals' behavior not only in absolute terms, but also in relative terms. In terms of relativity, we observed the treatment effects hold for all groups: prompting for students, freelancers and company developers, and the effect of framework for students and company developers. We believe that relative behavior is more relevant to the usable security and privacy community. Absolute values (such as the security score for password storage) are very dependent on the study sample and since it is extremely difficult to recruit a representative sample of developers, absolute values do not carry much weight beyond the study sample. However, relative values, such as "security scores for library A were better than scores for library B," look more robust and thus are more likely to be useful for researchers who want to test if e.g., a new system they designed improves the state of the art.

Our findings offer an indication for the ecological validity of security developer studies with CS students examining relative behavior. Since the recruitment of students for academic studies is considered as rather convenient by researchers, these are promising results for future studies. However, it should be taken into consideration that we examined a security example case in one programming language and thus further studies are needed to see if our results replicate in other cases as well.

CONCLUSION

The main goal of this work was to compare findings of studies conducted with students and freelancers recruited out of convenience to findings of studies conducted with professionally employed developers. Therefore, we replicated the password-storage study of Naiakshina et al. [30, 28] with 36 software developers employed by diverse companies. Our analysis showed that the behavior of company developers and students differed in absolute terms with regard to security measures. However, we found that the effect of the presented treatment of security prompting hold for all samples (students, freelancers, and company developers). Furthermore, the treatment effect of achieving more secure solutions by using an API with a higher level of password storage support existed for both, students and company developers. Since the results of students and company developers were similar in relative terms, we argue that security developer studies conducted with CS students can offer valuable insights if the effects of different treatments are explored. However, our study is based on an example case in one programming language and therefore, future work should examine other security tasks, developer characteristics, frameworks, and programming languages.

ACKNOWLEDGMENTS

The authors would like to thank Johanna Deuter for her help during the paper writing. This work was partially funded by the ERC Grant 678341: Frontiers of Usable Security.

REFERENCES

[1] [n.d.]. JavaServer Faces (JSF). ([n.d.]). Retrieved August 31, 2019 from https://javaee.github.io/javaserverfaces-spec/

[2] [n.d.]. Spring. ([n.d.]). Retrieved August 31, 2019 from https://spring.io/projects/spring-framework

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, IEEE, San Jose, CA, USA, 154–171.

[4] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016a. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE Press, Piscataway, NJ, USA, 289–305. DOI: http://dx.doi.org/10.1109/SP.2016.25

[5] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. 2016b. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In *Cybersecurity Development (SecDev), IEEE*. IEEE, IEEE Press, Piscataway, NJ, USA, 3–8.

[6] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. 2017. Security Developer Studies with GitHub Users: Exploring a Convenience Sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 81–95. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar

[7] Anne Adams and Martina Angela Sasse. 1999. Users are not the enemy. *Commun. ACM* 42, 12 (1999), 40–46.

[8] Hala Assal and Sonia Chiasson. 2019. 'Think Secure from the Beginning': A Survey with Software Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 289, 13 pages. DOI: http://dx.doi.org/10.1145/3290605.3300519

[9] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. (2014).

[10] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages?. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 575–585. DOI: http://dx.doi.org/10.1109/ICSE.2017.59

[11] Patrik Berander. 2004. Using Students as Subjects in Requirements Prioritization. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*. IEEE, IEEE, Redondo Beach, CA, USA, 167–176.

[12] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. 2015. *Argon2: the memory-hard function for password hashing and other applications*. Technical Report. Tech. rep., Password Hashing Competition (PHC).

[13] Kenneth P Burnham and David R Anderson. 2004. Multimodel inference: understanding AIC and BIC in model selection. *Sociological methods & research* 33, 2 (2004), 261–304.

[14] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[15] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. John Wiley & Sons.

[16] Nahid Golafshani. 2003. Understanding reliability and validity in qualitative research. *The qualitative report* 8, 4 (2003), 597–606.

[17] Paul A Grassi, James L Fenton, EM Newton, RA Perlner, AR Regenscheid, WE Burr, JP Richer, NB Lefkovitz, JM Danker, Yee-Yin Choong, and others. 2017. NIST Special Publication 800-63b: Digital Identity Guidelines. *Enrollment and Identity Proofing Requirements. url:https://pages.nist.gov/800-63-3/sp800-63b.html* (2017).

[18] Matthew Green and Matthew Smith. 2016. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security & Privacy* 14, 5 (2016), 40–46.

[19] George Hatzivasilis, Ioannis Papaefstathiou, and Charalampos Manifavas. 2015. Password Hashing Competition-Survey and Benchmark. *IACR Cryptology ePrint Archive* 2015 (2015), 265.

[20] Martin Höst, Björn Regnell, and Claes Wohlin. 2000. Using Students as Subjects- A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering* 5, 3 (2000), 201–214.

[21] Shubham Jain and Janne Lindqvist. 2014. Should I protect you? Understanding developers' behavior to privacy-preserving APIs. In *Workshop on Usable Security*, Vol. 2014.

[22] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 672–681. http://dl.acm.org/citation.cfm?id=2486788.2486877

[23] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I Have No Idea What I'm Doing" - On the Usability of Deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1339–1356. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/krombholz

[24] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006a. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 492–501. DOI: http://dx.doi.org/10.1145/1134285.1134355

[25] Thomas D LaToza, Gina Venolia, and Robert DeLine. 2006b. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. ACM, 492–501.

[26] L. Layman, L. Williams, and R. S. Amant. 2007. Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 176–185. DOI:http://dx.doi.org/10.1109/ESEM.2007.11

[27] Brad A Myers and Jeffrey Stylos. 2016. Improving API Usability. *Commun. ACM* 59, 6 (2016), 62–69.

[28] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 140, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300370

[29] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 311–328. DOI: http://dx.doi.org/10.1145/3133956.3134082

[30] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. 2018. Deception Task Design in Developer Password Studies: Exploring a Student Sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, USA, 297–313. https://www.usenix.org/conference/soups2018/presentation/naiakshina

[31] [n.d.]. [n.d.]a. LinkedIn. ([n.d.]). https://www.linkedin.com Accessed: September 2019.

[32] [n.d.]. [n.d.]b. Open Web Application Security Project (OWASP). ([n.d.]). https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md Accessed: September 2019.

[33] [n.d.]. [n.d.]c. Random (Java SE 11 & JDK 11) - Oracle Docs. ([n.d.]). Retrieved September 05, 2019 from https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html

[34] [n.d.]. [n.d.]d. SecureRandom (Java SE 11 & JDK 11) - Oracle Docs. ([n.d.]). Retrieved September 05, 2019 from https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/SecureRandom.html

[35] [n.d.]. [n.d.]e. XING. ([n.d.]). Retrieved August 31, 2019 from https://www.xing.com/

[36] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time: Supporting Android Developers in WritingSecure Code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1065–1077. DOI: http://dx.doi.org/10.1145/3133956.3133977

[37] Lutz Prechelt. 2011. Plat_Forms: A Web Development Platform Comparison by an Exploratory Experiment Searching for Emergent Platform Properties. *IEEE Transactions on Software Engineering* 37, 1 (Jan 2011), 95–108. DOI:http://dx.doi.org/10.1109/TSE.2010.22

[38] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. 2015. Are students representatives of professionals in software engineering experiments?. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, IEEE, Florence, Italy, 666–676.

[39] M Angela Sasse. 2003. Computer security: Anatomy of a usability disaster, and a plan for recovery. (2003).

[40] Martina Angela Sasse, Sacha Brostoff, and Dirk Weirich. 2001. Transforming the 'weakest link' - a human/computer interaction approach to usable and effective security. *BT technology journal* 19, 3 (2001), 122–131.

[41] R. Scandariato, J. Walden, and W. Joosen. 2013. Static analysis versus penetration testing: A controlled experiment. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Press, Piscataway, NJ, USA, 451–460. DOI: http://dx.doi.org/10.1109/ISSRE.2013.6698898

[42] Dag IK Sjoberg, Bente Anda, Erik Arisholm, Tore Dyba, Magne Jorgensen, Amela Karahasanovic, Espen Frimann Koren, and Marek Vokác. 2002. Conducting realistic experiments in software engineering. In *Proceedings international symposium on empirical software engineering*. IEEE, IEEE Press, Piscataway, NJ, USA, 17–26.

[43] Malcolm Smith. 2017. *Research methods in accounting*. Sage.

[44] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. 2008. Using Students As Subjects - an Empirical Evaluation. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08)*. ACM, New York, NY, USA, 288–290. DOI: http://dx.doi.org/10.1145/1414004.1414055

[45] David R Thomas. 2006. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation* 27, 2 (2006), 237–246.

[46] T. Thomas, B. Chu, H. Lipford, J. Smith, and E. Murphy-Hill. 2015. A study of interactive code annotation for access control vulnerabilities. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Press, Piscataway, NJ, USA, 73–77. DOI: http://dx.doi.org/10.1109/VLHCC.2015.7357200

[47] Tyler W. Thomas, Heather Lipford, Bill Chu, Justin Smith, and Emerson Murphy-Hill. 2016. What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/thomas

[48] Alma Whitten and J Doug Tygar. 1999. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0.. In *Usenix Security*, Vol. 1999.

[49] Chamila Wijayarathna and Nalin A. G. Arachchilage. 2018. Why Johnny Can't Store Passwords Securely?: A Usability Evaluation of Bouncycastle Password Hashing. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018 (EASE'18)*. ACM, New York, NY, USA, 205–210. DOI:http://dx.doi.org/10.1145/3210459.3210483

[50] J. Xie, H. R. Lipford, and B. Chu. 2011. Why do programmers make security errors?. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Press, Piscataway, NJ, USA, 161–164. DOI: http://dx.doi.org/10.1109/VLHCC.2011.6070393

[51] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. 2016. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, IEEE, San Jose, CA, USA, 158–177.