"If you want, I can store the encrypted password." A Password-Storage Field Study with Freelance Developers

Alena Naiakshina
University of Bonn
naiakshi@cs.uni-bonn.de

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

Eva Gerlitz
University of Bonn
gerlitz@uni-bonn.de

Emanuel von Zezschwitz
University of Bonn, Fraunhofer FKIE
zezschwitz@cs.uni-bonn.de

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

ABSTRACT

In 2017 and 2018, Naiakshina et al. [21, 22] studied in a lab setting whether computer science students need to be told to write code that stores passwords securely. The authors' results showed that, without explicit prompting, none of the students implemented secure password storage. When asked about this oversight, a common answer was that they would have implemented secure storage - if they were creating code for a company.

To shed light on this possible confusion, we conducted a mixed-methods field study with developers. We hired freelance developers online and gave them a similar password storage task followed by a questionnaire to gain additional insights into their work. From our research, we offer two contributions. First of all, we reveal that, similar to the students, freelancers do not store passwords securely unless prompted, they have misconceptions about secure password storage, and they use outdated methods. Secondly, we discuss the methodological implications of using freelancers and students in developer studies.

CCS CONCEPTS

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

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05.
https://doi.org/10.1145/3290605.3300370

KEYWORDS

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

ACM Reference Format:
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 *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300370

1 INTRODUCTION

In recent years, many researchers have investigated end-user password creation, password use and knowledge [7, 10, 16, 27, 31, 33–35, 37], as well as password composition policies and their effects [19, 20, 25, 28]. Due to end-users' weak password habits (e.g., password reuse) [17, 29], attackers can focus on services with low security mechanisms to target services with high security standards [8]. Still, only a few studies have examined developers handling end-user password storage, although these are primarily responsible for end-user password security.

In 2017 and 2018, Naiakshina et al. conducted two studies in which 40 computer science students were asked to implement the registration process for a university social network [21, 22]. In a qualitative [21] and quantitative study [22], Naiakshina et al. tested the effect of development framework and task design. Half the participants used plain Java (JSF) and the other half used the Spring framework. Spring offers a support library which implements password storage with a high level of security. With JSF, the participants had to implement salting and hashing on their own. To test whether participants would realize the need for secure password storage without prompting, the authors gave half the participants a task description which did not mention security, while the

other half were explicitly tasked with implementing a secure solution.

There were several takeaways from these two studies. Firstly, none of the non-primed participants (i.e., those who did not explicitly get tasked with creating a secure solution) stored passwords securely. Two of them attempted to create a secure solution but gave up and handed in a non-secure solution. Of the 20 participants who were explicitly tasked to create a secure solution, 12 implemented some level of security. Secondly, when comparing the different frameworks (JSF vs. Spring), more of those participants who implemented some security achieved a high score for security through Spring's default implementation. In the Spring conditions, all but one participant achieved the highest score of 6 points out of a possible 7. In the JSF conditions the spread was much greater with values ranging from 2 to 6. Thirdly, all participants who achieved a secure solution used "copy and paste" to do so: i.e., none of the participants achieved secure solutions by writing their own code.

However, there are a couple of caveats in the above results. Firstly, all participants were computer science students. It is unknown whether these same results would have been observed for other types of developers. Secondly, the participants were aware that they were taking part in a study. This could have led them to believe that security was not of interest in the non-priming conditions. It could also have led the participants to take security less seriously in the priming conditions, since they knew no real data would be under threat if they made a mistake or created a sub-optimal solution. This would make the results a study artifact and would invalidate the study design. This concern is supported by the fact that 15 of the 28 participants who did not implement a secure solution stated that they would have done so if they had been doing the same task for a real employer. It is of course also possible that the students' statements were merely an excuse provided after the fact to explain their behavior in a face-saving way. This possibility is supported by the fact that there are many password-database compromises in the wild [12, 14, 18, 36] in which developers made mistakes just like the students did in the study.

To shed light on this problem, we repeated Naiakshina et al.'s study with a few necessary modifications with freelance developers whom we hired online through Freelancer.com. We posed as a start-up company that had just lost a developer and needed help completing our social networking site. We hired 43 freelancers and gave them a regular contract to implement the registration process for us. This would allow us to see the security properties of code created by real developers who, we hoped, believed our ruse that they were writing code for a real company which would use genuine user data. After they had completed their task, they were paid either €100 or €200 and then informed about the real purpose of our study. We then asked if they would be willing to take part in an exit survey in order to give us more insights into their development process and to test whether they believed they were creating code for a company. Our key findings are:

- **If You Want Security, Ask For It.** We found that security prompting had a statistically significant effect on whether the participants stored the passwords in a secure way. Similar to the student sample in [21, 22], a number of freelancers did not feel responsible for security. Especially in the lower paid group (€100), the majority of non-prompted freelancers did not think about security.
- **Payment.** We found no effect of payment (€100 vs. €200) on the final security solutions. However, this bears further examination.
- **Field vs. Lab Study.** Freelance developers are aware that clients will use their solution in the real world. However, the quality of solutions was comparable to the solutions of students.
- **Misconceptions.** We found a number of freelancers were reducing password storage security to a visual representation and thus using `Base64` as their preferred method to ensure security. Additionally, *encryption* and *hashing* were used as synonyms, which was often reflected by the freelancers' programming code.
- **Continuous Learning.** A number of freelancers used outdated methods to store user passwords securely. This phenomenon was also observed in the student sample [21] and shows that freelancers do not update their knowledge as well.
- **Copy and Paste.** Similar to students' solutions from the lab study, we identified freelancers' security code on the Internet.
- **Social Desirability.** A number of freelancers reported they would store user passwords securely even without a security instruction. However, these participants sent insecure solutions as their first submissions.

2 RELATED WORK

The related work section is divided into two parts. First, we discuss related work in the area of developer and password studies. Second, for the methodological discussion in section 6, we summarize those security developer studies that utilized a sample group of freelancers.

Developer and Password Studies

In recent years many studies have been published that investigated end-user password creation, password use and knowledge [7, 10, 16, 27, 31, 33–35, 37], as well as password composition policies and their effects [19, 20, 25, 28]. Even though

mistakes by developers and administrators have greater consequences, relatively little work has been done in examining these actors in the context of password storage. The two studies on which this paper is based were conducted by Naiakshina et al. [21, 22] and were described in the introduction.

In [39], Wijayarathna and Arachchilage recruited 10 developers over GitHub to test the usability of the `scrypt` hashing functionality of the Bouncy Castle API. Participants worked on their own computers and had to improve the password storage of a simple web application by using `scrypt`. The authors were able to identify 63 usability issues regarding secure password storage.

Acar et al. [3] conducted a study with 307 GitHub users. Participants were requested to implement 3 security related tasks (URL shortener, credential storage, string encryption). The authors also asked the developers to complete a questionnaire about their programming experience, security background, and occupation. The authors found a significant effect of programming experience on the resulting implementations with regard to functionality and security. Of the 307 participants, 162 securely stored user credentials in a database. By contrast, the majority of the insecure solutions left user passwords vulnerable to rainbow-table attacks or stored them in plain text.

Balebako et al. [4] interviewed 13 and surveyed 228 app developers to find out how they make decisions concerning privacy and security. According to the survey, half of these developers reported that they securely store data in a database. However, the work done by Naiakshina et al. and our study show that developers also misreport and/or misunderstand secure storage. Therefore, we caution against using self-reported data in this respect.

Bonneau and Preibusch [8] identified 150 websites which offered free user accounts relying on user-chosen passwords. They found that websites with few security features have the worst security practices and assumed they use passwords mainly for psychological reasons, such as building up a relationship with their customers. By contrast, websites storing payment details or personal user data made safer security choices.

In 2007 Prechelt [24] staged a contest where web developers competed using three different web development platforms (Java EE, Pearl, PHP) in a two-day event. The resulting code was analyzed for usability, functionality, reliability, security, and structure. While there were some large differences in platform characteristics, PHP was found to have the smallest within-platform variations. Finifter and Wagner [11] also evaluated the program code of [24]. They checked for correlations between the number of vulnerabilities and the programming language or the framework's support for security features. While no correlations between the language and the web-application's security were found, the authors did notice that developers almost never make use of the built-in support for password storage that was offered by many frameworks.

Acar et al. [2] examined the effect of documentation resources on the security of programmers' code. Developers were advised to complete four programming tasks: the storage of data, the use of HTTPS, the use of ICC, and the use of permissions. The authors found developers who used Stack Overflow created less secure code.

Acar et al [1] conducted an online developer study to investigate the usability of Python crypto-APIs. They found developers need accessible documentations with secure and easy-to-use code examples.

Gorski et al. [15] conducted a controlled online experiment with 53 GitHub users to test the effectiveness of API-integrated security advice. They found that 73% of those participants who received a security warning and advice improved their insecure code.

Developer Studies with Freelancers

Yamashita and Moonen [40, 41] invited 85 freelancers from Freelancer.com to answer a survey about developers' knowledge of code smells and their interest in them. 32% of the respondents admitted to not knowing anything about code smells. The majority of the participants mentioned being moderately concerned about the presence of smells in source code.

In a tech report Bau et al. [6] developed a metric for web application vulnerability scanners. They compared the code created by 19 start-ups with those created by 9 freelancers hired from freelance websites. The freelancers were asked to design a complete youth sports photo sharing site with a number of features. To highlight their interest in security, the authors asked the freelancers to follow *legal regulations* and to *secure* sensitive contact information. Three different languages (PHP, Java/JSP, ASP) and three price ranges (<$1000, $1000-$2500, and >$2500) were chosen for the task. For the analysis, Bau et al. searched for correlations between the vulnerability rate of a web application and the programming language, developer's occupation, and their security background knowledge. Web applications written in PHP and those implemented by freelancers showed higher weaknesses. Regarding password storage, a huge gap was found between freelancers' knowledge and the resulting implementation. We used the photo sharing scenario as the inspiration for our own company scenario. However, we substantially reduced the scope of the task to focus solely on the registration process.

3 LIMITATIONS

Our study has the following limitations which need to be taken into account:

Sample. We sampled developers from Freelancer.com. This sample is certainly not representative for all developers and might not even be representative for other freelancer hiring services.

Deception. The premise of our study hinged on the freelance developers believing that they were developing code for a real company and that any vulnerabilities would affect real users. In our exit survey we asked whether they suspected that the task they had been given was part of a study and asked for their reasoning. We also analyzed the developer chat for indications that a developer did not take the task seriously. We only found one participant ($N2_{100}$) who stated that he had taken part in scientific studies before and that he thought the task description was too detailed for a normal job and thus he had suspected that it was part of a study. We removed this participant from the statistical evaluation. One other freelancer ($P2_{100}$) indicated that he thought that the job might be a test job to evaluate him before being given bigger tasks. On the whole it seems that our freelancer sample took their jobs seriously.

Language. Many of our participants were non-native English speakers and in some cases answers were difficult to understand. This also created the risk that participants had difficulty understanding the task description. While this is a realistic problem for real jobs as well, it is sub-optimal for a study.

We also had one problem during the execution of our study. The project code sent to the first 17 freelancers already contained two security imports (java.security and javax.crypto). This could have inadvertently primed participants in the non-prompted conditions that security was important. Luckily for us,[1] none of these participants implemented a secure solution.

Ethics

The use of deception in research always needs to be critically appraised and should only be used when strictly necessary. Since our goal was to gather real world data, we opted to pose as a company and hire freelancers. We informed participants after completion and payment and gave them the opportunity to continue to the questionnaire or withdraw from the study. None withdrew from the study and only one did not take part in the questionnaire. The participants' feedback was positive throughout, both in the questionnaire as well as in the reviews they left on the Freelancer site. We initially offered some of the freelancers €100 and the others €200 to see if that affected security performance. For reasons of fairness, the €100-group was actually paid €200 at the end of the study. The Research Ethics Board of our university reviewed and approved our study.

[1]But unluckily for security in general.

4 METHODOLOGY

As a starting point we adopted the methodology and study design frame from [21, 22], which was described in the Introduction. However, instead of using students in a lab environment we hired freelancers from the web platform Freelancer.com for our study. We kept the priming/non-priming conditions in which participants were either prompted to store the passwords securely or not. However, we dropped the Spring condition since the security scores in [21, 22] were mostly homogeneous and we hoped to gain more insights by focusing on JSF.

First Pilot Study

In our first pilot study we used exactly the same task as [21, 22]. We did not state that it was research, but posted the task as a real job offer on Freelancer.com. We set the price range at €30 to €250. Eight freelancers responded with offers ranging from €100 to €177. The time ranged from 3 to 10 days. We arbitrarily chose one with an average expectation of compensation (€148) and 3 working days delivery time. We gave this participant the non-prompting task. After approximately 21 hours, we received the freelancer's solution which, just like the students in [21, 22], stored the passwords in plain text. As part of the regular approval process we requested the passwords to be stored securely. After a further 38 hours the participant submitted code using symmetric encryption.

Critically, the participant mentioned in the follow-up chat that he is familiar with university exercise sheets and is available for further tasks. Since the task from [21, 22] was framed as a university social networking website, he assumed the task to be an exercise for university credit.[2]

Updated Study Design

Since we did not want the freelancers to think the code was "only" needed for course credit but would actually be used in the real world, we decided to change the task framing. We changed the task from a university social networking platform to a sports photo sharing social network. To make it more believable we created a web presence for the company. Screen-shots from the different pages and the task description can be found in the supplemental material.

While hiring the freelancers we posed as employees of the company and stated that we had just lost our developer and wanted help in finishing the registration code. We provided the freelancers the platform code as a ZIP file. Participants had to store user data in a remote database provided by us via Amazon Web Services (AWS).

Second Pilot Study. In a second pilot study we tested the new task design. The task was posted as a project with a

[2]Troublingly, it seems that freelancers are often hired by students to work on university assignments.

price range from €30-€100. Java was specified as a required skill. Fifteen developers made an application for the project. Their compensation proposals ranged from €55 to €166 and the expected working time ranged from 1 to 15 days. We randomly chose two freelancers from the applicants, who did not ask for more than €110 and had at least 2 good reviews. One freelancer was prompted for secure password storage while the other was not. However, the freelancer website shows who is working on which project and this caused confusion for the second freelancer who thought the job had already been done. To avoid this misunderstanding we switched to recruiting via private messages, which will be detailed in the next section. Both freelancers provided us with very positive feedback with regard to our study.

Final Study

For the final study we recruited freelancers via direct messages. We searched for all freelancers and filtered for the skill "Java." Unfortunately, Freelancer.com's search function also returns JavaScript developers or developers where we saw no connection to Java, so we manually pruned out developers whose profile did not include Java skills.

Based on our experience in the pre-studies we added two payment levels to our study design (€100 and €200). We only accepted freelancers' submissions if they were functional.

The final component of the study was a play-book to control the interaction with the participants (see supplemental material). Unlike a study in which the experimenter usually does not interact with the participants, it is normal and expected for the hiring person to interact with the freelancers. For the communication with freelancers we used the chat functionality offered by Freelancer.com. We built a play-book based on the interactions in the pre-studies and extended this whenever needed. The play-book dictated how we would respond to the different queries to keep our interaction as homogeneous as possible. The three most important interactions were as follows: if a participant asked if he or she should store passwords securely or if a certain method was acceptable, we answered "*Yes, please!*" and "*Whatever you would recommend/use.*" If a participant delivered a solution where passwords were stored in plain text in the database we replied "*I saw that the password is stored in clear text. Could you also store it securely?*" These participants are marked as having received the *security request*. We deliberately set the bar for this extra request low, to emulate what a security-unaware requester could do; i.e., if it looked like something hashed we accepted it. After final code submission freelancers were informed about the real purpose of the project and were invited to answer a questionnaire. The questionnaire can be found in the supplemental material and is an extended version of the one used in [21, 22]. For filling out the survey, participants were compensated with an additional €20.

On Freelancer.com it is common to provide reviews for employers and employees. In order to make sure that other freelancers did not discover that we were conducting a study, we asked all freelancers to avoid mentioning our real purpose in their reviews.

Scoring Code Security

Some of the freelancers sent us videos that showed how data was stored in a database. Additionally, we tested each submission for functionality within our system. For scoring the security of the freelancers' code, we adopted the security scale of Naiakshina et al. [21, 22] (see supplemental material). The scoring system contains a binary variable *secure* indicating whether participants used any kind of security in their code and an ordinal variable *security* to score how well they did. The score is based on a range of factors such as what hash algorithm was used, the iteration count, and whether and how the salt was generated. The value of *security* could range from 0 to 7, although 6 was the highest score actually achieved in both the freelance and student group.

We had to extend the scale because the freelancers used two methods which did not appear in the student study. In our study we also saw the use of `Base64` encoding and symmetric encryption, as well as one occurrence of `HMAC`. We scored these as follows: both `Base64` and symmetric encryption received 0 points for security. `HMAC` was treated as a hash function with a non-random salt. All code was independently reviewed by three authors. Differences between the scores were resolved by discussion.

Quantitative Analysis

Due to the adjusted study design, we were able to test three of the seven main hypotheses from [22]. In particular, we tested whether prompting (H-P1), years of Java experience (H-G1), or password storage experience (H-G2) had an effect on security. We used the same statistical tests as used in [22]. We did not use multivariate statistics since we considered our sample size too small [30]. We used Bonferroni-Holm corrections for all tests concerning the same dependent variable. To ease identification, we labeled Bonferroni-Holm corrected tests with "family = N," where N is the family size, and reported both the initial and corrected p-values ($cor-p$). With roughly 10 participants per condition we could only find large effects; therefore, absence of statistical significance should not be interpreted as an absence of an effect. A summary of our statistical analysis results can be found in the supplementary material.

Qualitative Analysis

We used *inductive coding* [32] to analyze our qualitative data from the open questions in the survey as well as the chat

Gender	Male: 39	Female: 1	Prefer not to say: 1, NA: 2
Age*	min = 22, max = 68	mean = 30.34, median = 29	sd = 7.63, NA: 2
University Degree	Yes: 37	No: 4	NA: 2
Profession	Freelance developer: 29	Industrial developer: 7	Academic researcher: 1
	University collaborator and freelance developer: 1	Software engineering lead: 1	Undergraduate student: 1, Graduate student: 1
		NA: 2	
Country of Origin	Bangladesh: 1	India: 14	Vietnam: 2
	China: 8	United States: 3	Italy, Mexico: 2
	Mongolia: 1	Nigeria: 1	Pakistan: 4
	Sri Lanka: 2	Egypt: 3	NA: 2
General Development Experience [years]*	min = 2, max = 20	mean = 8, median = 8	sd = 3.61, NA: 2
Java Experience [years]*	min = 1, max = 15	mean = 6.39, median = 6	sd = 2.66, NA: 2

* = There were no significant demographic differences between the groups.

Table 1: Demographics of the 43 Participants

interactions during development. Two researchers independently searched for themes and categories emerging in the raw data. After coding open questions of a new participant, both researchers went back to their codes, analyzing them again for the prior participants as well. After the coding process was completed, the codes were compared and the inter-coder agreement by using the Cohen's kappa coefficient (κ) [9] was calculated. The agreement was 0.91. A value above 0.75 is considered a good level of coding agreement [13].

Participants

As described above, we used Freelancer.com to search for developers with Java skills and manually removed those who only specified JavaScript in their profile. In total we selected 340 developers of which we had to remove 80 due to the search issue. That left us with 260 remaining Java freelancers. We randomly assigned these 260 participants to one of our four conditions: **P**rompting-**100**Euro (P_{100}), **N**on-Prompting-**100**Euro (N_{100}), **P**rompting-**200**Euro (P_{200}), **N**on-Prompting-**200**Euro (N_{200}) and then contacted them with the job offer. We did this in batches to balance conditions in case of higher uneven rates. A total of 211 did not accept the offer. The most common reasons were as follows: 72 did not respond; 63 declined the job because they were not experienced enough with Java, JSF, Hibernate, or PostgreSQL, which were mentioned in the task description; and 22 declined due to lack of time. We hired the remaining 49 developers of whom 44 completed the task. One of these had technical trouble submitting his solution so we only have 43 participants in our final sample.

The 43 valid participants reported ages between 22 and 68 years (median: 29, mean: 30.34; sd: 7.63) and almost all of them reported being male (39/43). All but two of our participants had been programming in general for at least two years and in Java for at least a year. Most (29) named freelancing as their main profession. Seven indicated to be industrial developers and only two reported to be students. More information on the demographics can be found in Table 1.

We analyzed the effect of the two different payment levels on the acceptance rate of freelancers. In the prompted task, which asked for a secure solution, 11 of 31 accepted the €100 offer, and 14 out of 20 accepted the €200 offer. Fisher's exact test showed this difference to be statistically significant (p = 0.02*, confidence interval [CI] = [0.058, 0.90], odds ratio [OR] = 0.24). In the non-prompted conditions, where security was not mentioned, 12 of 18 accepted the €100 offer and 11 of 14 accepted the €200 offer. The differences point in the same direction, but were not statistically significant.

5 RESULTS OF FREELANCER PASSWORD STUDY

Table 2 and Table 3 show a summary of the participants' submission evaluation.[3] It took our participants a mean of 3.2 days (sd 2.1, median 3) to submit their solution, including the time to add security if we had to request that. Those participants, who delivered an insecure solution at first and were asked to store the passwords securely, needed a mean of 6.4 hours (sd 7.3h, median 3.17 h) to fulfill our request.

Security

Our freelancers used three different techniques to store user passwords: (1) hashing (+ salting); (2) symmetric encryption; and (3) `Base64` encoding.

Seventeen freelancers used a hash function in their first submissions. After receiving a security request, 29 of the freelancers overall stored user passwords securely by hashing. Participants who decided to use `bcrypt` benefited from the library's automatic salt generation. Also, participants who used `PBKDF2` came across a salt parameter and were thus forced to generate a salt value. By contrast, only 3 of 17 participants, who used other hash algorithms, implemented salting. One of them generated a random salt, one made use of the username, and one hard-coded a static salt.

Three freelancers used symmetric encryption in order to securely store user passwords in their first submissions. After

[3]Table 2 considers participant $N2_{100}$, who suspected the task was part of a research study. This participant was not included in Table 3 and was also excluded from the statistical tests in the following sections.

Participant	Prompting	Payment	Working Time	Include Security	Secure	Security Score	Function	Length in bits	Iteration	Salt	C&P	Study	SQ
P1$_{100}$	**1**	**100**	**1 Day**	**3min**	**0/0**	**0/0**	**Base64**				✗	✗	✗
P2$_{100}$	1	100	7 Days		0	0	Sym. encryption(3DES)				✗	✗	✗
P3$_{100}$	1	100	8 Days		1	1	MD5	128	1		✓	✗	✗
P4$_{100}$	**1**	**100**	**1 Day**	**1h 50min**	**0/1**	**0/6**	**BCrypt**	**184**	**2^{10}**	**SecureRandom**	✗	✗	✗
P5$_{100}$	1	100	3 Days		1	6	BCrypt	184	2^{10}	SecureRandom	✓	✗	✗
P6$_{100}$	1	100	3 Days		1	3	SHA-256	256	1	Username	✓	✗	✗
P7$_{100}$	1	100	3 Days		1	1	MD5	128	1		✗	✗	✗
P8$_{100}$	1	100	2 Days		1	6	BCrypt	184	2^{10}	SecureRandom	✓	✗	✗
P9$_{100}$	1	100	3 Days		0	0	Base64				✗	✗	✗
P1$_{200}$	1	200	4 Days		0	0	Sym. encryption(3DES)				✓	✗	✗
P2$_{200}$	1	200	3 Days		1	1	MD5	128	1		✓	✗	✗
P3$_{200}$	1	200	1 Day		1	1	MD5	128	1		✗	✗	✓
P4$_{200}$	1	200	5 Days		1	1	MD5	128	1		✓	✗	✗
P5$_{200}$	1	200	3 Days		1	3	HMAC, SHA-1	160	2	Static	✗	✗	✗
P6$_{200}$	**1**	**200**	**2 Days**	**4h 15min**	**0/0**	**0/0**	**Sym. encryption(AES)**				✗	✗	✓
P7$_{200}$	1	200	5 Days		0	0	Base64				✗	✗	✗
P8$_{200}$	1	200	3 Days		1	6	PBKDF2(SHA-1)	256	10 000	Random	✓	✗	✗
P9$_{200}$	1	200	6 Days		1	6	BCrypt	184	2^{11}	SecureRandom	✓	✗	✗
P10$_{200}$	1	200	1 Day		0	0	Base64				✗	✗	✗
P11$_{200}$	1	200	1 Day		1	6	BCrypt	184	2^{11}	SecureRandom	✗	✗	✗
P12$_{200}$	1	200	2 Days		1	5	PBKDF2	256	10 000	Static	✓	✗	✗
N1$_{100}$	**0**	**100**	**1 Day**	**4min**	**0/1**	**0/6**	**BCrypt**	**184**	**2^{12}**	**SecureRandom**	✗	✗	✗
N2$_{100}$	**0**	**100**	**1 Day**	**6h 20min**	**0/1**	**0/2**	**SHA-1**	**160**	**1**		✗	✓	✗
N3$_{100}$	0	100	2 Days		0	0	Base64				✗	✗	✗
N4$_{100}$	**0**	**100**	**5 Days**	**17h**	**0/1**	**0/1**	**MD5**	**128**	**1**		✓	✗	✗
N5$_{100}$	**0**	**100**	**1 Day**	**18h**	**0/1**	**0/2**	**SHA-256**	**256**	**1**		✗	✗	✓
N6$_{100}$	**0**	**100**	**5 Days**	**21h**	**0/0**	**0/0**	**Base64**				✗	✗	✓
N7$_{100}$	**0**	**100**	**3 Days**	**3h**	**0/0**	**0/0**	**Sym. encryption(AES)**				✗	✗	✗
N8$_{100}$	**0**	**100**	**2 Days**	**25min**	**0/1**	**0/1**	**MD5**	**128**	**1**		✓	✗	✓
N9$_{100}$	0	100	1 Day		1	4	MD5	128	1	SecureRandom	✓	✗	✗
N10$_{100}$	**0**	**100**	**3 Days**	**3h 20min**	**0/1**	**0/1**	**MD5**	**128**	**1**		✗	✗	✗
N11$_{100}$	**0**	**100**	**8 Days**	**19h**	**0/1**	**0/2**	**SHA-256**	**256**	**1**		✗	✗	✗
N12$_{100}$	0	100	4 Days		0	0	Sym. encryption(AES)				✓	✗	✗
N1$_{200}$	**0**	**200**	**1 Day**	**6min**	**0/0**	**0/0**	**Base64**				✗	✗	✗
N2$_{200}$	0	200	1 Day		1	6	PBKDF2(SHA-1)	256	10 000	SecureRandom	✓	✗	✗
N3$_{200}$	**0**	**200**	**5 Days**	**10min**	**0/1**	**0/1**	**MD5**	**128**	**1**		✗	✗	✗
N4$_{200}$	**0**	**200**	**3 Days**	**1h**	**0/1**	**0/2**	**SHA-256**	**256**	**1**		✗	✗	✗
N5$_{200}$	0	200	4 Days		1	2	SHA-256	256	1		✗	✗	✗
N6$_{200}$	0	200	2 Days		1	6	PBKDF2(SHA-1)	256	65 536	SecureRandom	✗	✗	✗
N7$_{200}$	0	200	4 Days		0	0	Base64				✗	✗	✗
N8$_{200}$	**0**	**200**	**2 Days**	**4h**	**0/1**	**0/6**	**PBKDF2(SHA-1)**	**1152**	64 000	**SecureRandom**	✓	✗	✗
N9$_{200}$	**0**	**200**	**3 Days**	**5h**	**0/0**	**0/0**	**Sym. encryption(3DES)**				✗	✗	✗
N10$_{200}$	**0**	**200**	**3 Days**	**3h**	**0/1**	**0/6**	**BCrypt**	**184**	**2^{10}**	**SecureRandom**	✓	✗	✗

Table 2: Evaluation of Participants' Submissions

Bold: Participants who at first delivered an insecure solution and received the additional security request. **Working time** participants took to submit their first solution. **Include Security:** Time participants needed to add security after the request. **C&P:** Security code was copied and pasted from the Internet. **Study:** Freelancers who stated that they were aware the project might be a study. **SQ:** Freelancers who asked which hashing function they should use.

receiving a security hint, 3 additional freelancers decided to use symmetric encryption. Interestingly, almost all of them used a secret key, which was a combination of our social networking website name "SportSnapShare" and some other String (e.g., registration). As noted in the methodology, we did not consider this as a secure solution.

Further, 5 of our participants decided to use `Base64` to encode the passwords in their first submission. After receiving a security instruction, 3 additional freelancers added `Base64` to their solution. Naturally, this is not a secure solution.

Since some participants received a security request, we checked to see if the security scores of these participants differed from those who did not (but who considered security by themselves). We did not find such a difference (Wilcoxon rank sum: W = 200, p-value = 0.83).

Next we wanted to see the effects of any of our two variables task description (prompting vs. non-prompting) and payment (€100 vs. €200). For this we first conducted a Fisher's exact omnibus test on all four conditions (FET: p = 0.02*). Further, we followed this up with three post-hoc tests (regarding prompting, payment and password storage experience) with Bonferroni-Holm correction to test the variables separately.

Task Design. For the first solution, we received 17 non-secure solutions from the non-prompted and 8 from the prompted participants (see Table 3). For the secure solutions, 4 came from non-prompted participants and 13 from prompted. Consequently, task design had a significant effect, with non-prompted participants delivering fewer initial secure solutions (FET: p = 0.01*, $cor - p$ = 0.03*, OR = 6.55, CI = [1.44, 37.04], family = 3).

Payment did not have a significant effect (FET: p = 0.22, $cor - p$ = 0.44, family = 3). However, more participants handed in secure initial solutions in the €200 conditions. Since we had a small sample size, we recommend not dismissing the

	Secure	Insecure
Prompted_{100}	5	4
Prompted_{200}	8	4
$\text{Non-Prompted}_{100}$	1	10
$\text{Non-Prompted}_{200}$	3	7

Table 3: Number of Secure Solutions per Condition

different payment levels yet and looking at this in future work.

Experience. Acar et al. [3] found a correlation between programming experience, whereas Naiakshina et al. [22] did not. Naiakshina postulated that the effect might not be visible due to the smaller range of experience seen in the student sample. Thus, we also tested whether more Java experience is correlated with higher security scores. We found no effect (Kruskal-Wallis: χ^2 = 13.31, df = 10, p-value = 0.21). Further, we investigated whether previous password experience might affect the security awareness. We found no significant effect between whether the participants stated that they had stored passwords before and whether their solutions were secure (FET: p-value = 0.52, $cor - p$ = 0.52, OR = 0, CI = [0, 8.91], family = 3).

Copy and Paste. Naiakshina et al. [22] found a significant positive effect of copy and paste. Due to the fact that we conducted a field study we could not gather the same level of information on this behavior. We did, however, analyze the code to see if we could find online sources. We found 16 obviously copied examples. Out of these 16 participants, 6 copied `MD5`, 2 symmetric encryption, 4 `bcrypt`, 3 `PBKDF2`, and 1 `SHA-1` as security methods. We cannot say with certainty that the other solutions were not copied, so we did not perform any comparisons on this data.

Qualitative Analysis

Next, we present a qualitative analysis of the data we gathered. Based on the inductive coding process we separate our findings into three phases: (1) the request phase; (2) the implementation phase; and (3) the reflection phase. The coding overview can be found in the supplemental material.

Phase One - Request

Our participants mentioned several aspects from which they decided whether to implement what they thought was a secure solution. While N4_{100} stated that security is dependent on data sensitivity, most of our participants (N6_{100}, N10_{100}, P9_{200}, P7_{100}, P5_{100}, N11_{100}, N7_{100}, N2_{100}, N9_{200}, N5_{200}) stated that security should be part of the task description from the client: "*I cannot find it in requirements, password encryption, can you tell me where is it written? I might have missed it*" (N11_{100}). Interestingly, this was also the case for participants in the prompted conditions, where a secure solution was explicitly required.

In the non-prompted group we had an interesting case: N10_{100} sent us a message asking whether he should store user passwords securely. However, before we could reply he had already handed in a plain text solution. This happened within three hours during the night in our time zone.

As mentioned above, we found a significant effect of prompting and non-prompting with our freelancers. Four out of 22 non-prompted developers did add security, which is more than the 0 out of 20 in the student lab study.[4] Yet the lesson remains the same: Even for a task which - for security experts - is obviously security-critical, like storing passwords, one should not expect developers to know this or be willing to spend time on it without explicit prompting:

> *"If you want, I can store the encrypted password."* (P2_{200})

So it is absolutely necessary to explicitly state that security is desired.

Phase Two - Implementation

Misconceptions. While the task description had a similar effect on the freelancers as on the students, we found some interesting differences concerning misconceptions about secure password storage. For example, some participants in the student sample confused password storage security with data transmission security. This confusion was not found in our freelance sample. Instead, we found another phenomenon, which was not observed in the student sample: the usage of the binary-to-text encoding scheme `Base64` to "securely" store user passwords in a database. Eight of our freelancers stored user passwords in the database by using `Base64`; 4 of them were in the prompted condition and 4 in the non-prompted condition. Of the 18 participants who received the additional security request, 3 (N1_{200}, N6_{100}, P1_{100}) decided to use `Base64` and argued, for example: "*[I] encrypted it so the clear password is not visible*" (N1_{200}) and "*It is very tough to decrypt*" (N6_{100}).

We also found misconceptions which were shared by students and freelancers. Many participants used *hashing* and *encryption* as synonyms (e.g., P5_{200}, N7_{100}, N11_{100}, N10_{200}, P2_{200}, N8_{100}, N10_{100}, N12_{100}). However, this did not lead any of the students to use symmetric encryption methods to store passwords. This misconception was strongly reflected by the freelancers' code submissions. For instance, when asked how he stored user passwords, N7_{100} stated he "*used [a] hashing algorithm.*" In actual fact, he used symmetric encryption. Six

[4]The difference is not statistically significant (FET p = 0.12). However, due to the small sample size this should not be over-interpreted.

freelancers used symmetric encryption in order to store user passwords and some were very convinced that this was a good solution ("*It will be almost impossible to break*" ($P1_{200}$)). Another freelancer ($P5_{200}$) used the hash-based message authentication code `HMAC` in order to secure user passwords. While being useful in some context, this method is not intended to be used for storing user passwords in a database.

Many freelancers also used outdated methods, but still claimed to have implemented a sufficient or even optimal secure solution (e.g., $N5_{100}$, $N11_{100}$, $P2_{200}$, $N8_{100}$, $N10_{100}$, $P3_{200}$). For instance, $N11_{100}$ declared, "*Passwords are stored after encryption, usually MD5. This makes it secure and almost impossible to compromise.*"

Functionality First, Security Costs Extra. Similar to end-users and participants from the student sample [21], freelancers concentrated on the functionality first. The complexity of the application ($N7_{100}$) and lack of time ($P9_{100}$) were named as reasons for this. Since we made no restrictions on the time, this probably refers to either a cost-benefit calculation or the overall workload of the freelancer. However, those participants who received the security request, took a median time of 3.17 hours to add security. For instance, after receiving the instruction to store user passwords securely, $N3_{200}$ said "*Sure, it's a piece of cake*" and added `MD5` without salt or iterations in 10 minutes. Interestingly, $N11_{100}$ even asked for an extra payment of €20: "*I can add, but I'll have to implement md5 encryption at client end. Its a couple hrs job. Can you please increase budget a little?*" Another indication that security impacts the budget can be seen in the rejection rate of the 100 euro job offers in which security was mentioned (see Section 4).

Library Usability. Similar to the students, our participants mentioned that they like APIs/libraries which provide automatic security ($N10_{200}$) and only need a few lines of code to work ($P8_{100}$, $P11_{200}$). A number of participants reported that they specifically chose an API/library because it is easy to use ($N1_{100}$, $N10_{100}$, $N3_{200}$, $N6_{200}$, $P4_{200}$).

Phase Three - Reflection

After submitting their solutions, the freelancers filled out our survey in which they reflected their knowledge and experience with regard to password storage security. Based on our coding we found some distinct character types in these reflections. The coding table in the supplemental material gives details and counts. Due to the small sample size, the relation between the groups should not be over-interpreted.

Cocky Developers. We found freelancers who believed that they had created an optimal or even great solution: "*Because I write the best code always*" ($N1_{100}$). Indeed, his final solution was one of the best since he used `bcrypt` and received 6 of 7 points for security. However, his first submission was insecure and he needed to be explicitly asked for security. $P11_{200}$ and $N4_{200}$ also claimed to write optimal code, because they had "*extensive experience*" and were "*skillful.*" While $P11_{200}$ also submitted a high scoring solution with 6 points, $N4_{200}$ received only 2 points for security.

Developers' Uncertainties. Several freelancers stated that they were unsure about the security of their solutions. However, this did not necessarily mean that they created bad solutions. $N8_{200}$ indicated that he was undecided about which algorithm or which parameters are most appropriate for safe password storage. Yet he received 6 out of 7 points for security. By contrast, $P3_{100}$ and $P7_{200}$ indicated not understanding hashing as a security concept at all. $P3_{100}$ used `MD5` and $P7_{200}$ `Base64` as techniques in their final applications. Although $P3_{100}$ was aware he used an outdated hash function, he did not change this in his code. $P9_{100}$ was also aware that using `Base64` is the wrong technique and yet, despite being in the prompting condition, he implemented this in his final submission.

Security Aware. We also had a couple of security-conscious developers who implemented security despite being in the non-prompting condition. However, we did observe different levels of knowledge concerning the actual implementation. $N6_{200}$ received 6 of 7 points without any prompting. He noted: "*I think this is an intuitive step since nobody wants to put his passwords under exposure.*" By contrast, $P4_{200}$ used `MD5` to store user passwords securely and argued "*I've used this technique many times to store passwords and have not faced any security issues.*"

Trust in Standards. A fair number of our freelancers argued that they trust standards and third party APIs to do the right thing and store passwords securely ($P8_{100}$, $P9_{200}$, $N11_{100}$, $P3_{200}$, $N2_{100}$, $P2_{100}$, $N4_{100}$, $P11_{200}$, $P12_{200}$). However, this trust is sometimes misplaced. While $P8_{100}$ and $P9_{200}$ indeed used industry standards for security and thus received 6 of 7 points, almost all of the other participants used `MD5` as a "standard" for password storage. Additionally, $P6_{100}$, $P9_{200}$ and $N11_{100}$ indicated that trust in organizations is an important aspect when choosing security features.

Testing. $N9_{100}$, $N9_{200}$, and $P7_{200}$ claimed to have conducted tests in order to ensure security even though, by using `MD5` symmetric encryption and `Base64`, they did not ensure best practices.

Social Desirability. We also had some instances of what is likely to be a manifestation of the social desirability bias while answering survey questions. Several freelancers stated that they store passwords securely even if not explicitly

instructed to. However, 14 of these 30 sent insecure solutions as their first submission.

6 METHODOLOGICAL DISCUSSION

While the usable security and privacy community has ample experience in recruiting participants for lab and online end-user studies, researchers have limited resources and lack knowledge of how and where to recruit professionals for security developer studies. Amazon Mechanical Turk (MTurk) is one of the most famous examples where participants are recruited for online end-user studies. A comparable service for software developer recruitment is not yet known. Therefore, previous studies used convenience samples such as computer science students [2, 5, 21, 23] or GitHub developers [3, 15, 26, 38, 39].

The response rate for GitHub users is rather low [1–3, 15, 26, 39]. For instance, of the 50,000 invited developers in [2], 302 took part in the study. In [15] 38,533 emails were sent to GitHub users, from which 272 agreed to take part in the study - finally resulting in 53 valid participants.

Apart from the low response rate, the iterated sending of study invitation emails can result in bothersome spam after a while. In [3], of the 23,661 invited GitHub users, 315 participants completed the study, 3,890 requests were bounced, and another 447 invitees requested to be removed from the list. Also in [1], 52,448 emails were sent to GitHub users from which 256 participants completed the study, 5,918 emails bounced, and 698 users requested to be removed from the list. Consequently, we assume that this recruitment method will fail for future studies.

Apart from gathering results from a field study with freelance developers, we wanted to see how students and online freelancers differ when conducting a similar task. Due to the necessary change in the task description this is not a direct comparison. Nevertheless, we believe we can offer some valuable insights into the use of these two groups for developer studies.

Freelance vs. Student Sample

Requirements. Similar to the students from Naiakshina et al. [21] (e.g., JN11: "*I was aware that the good practice is to store them securely, but the task didn't mention anything about that*"), our participants relied on client requirements when deciding whether they wanted to store the passwords securely. Therefore, task description is a main motivator when deciding to deal with security.

Misconceptions. Both students and freelancers have security misconceptions. Interestingly, our freelancers had a wider range of them. A number of freelancers used `Base64` to store user passwords securely. This misconception shows that participants confuse encoding functions with hashing functions by reducing them to visual representations, a phenomenon also shared by end-users. Additionally, *encryption* and *hashing* were used as synonyms, also often reflected by the freelancers' programming code. It seems that due to misconceptions, developers are searching for familiar terms like *encryption* and do not take the time to check whether this method is useful in the related use-cases.

Continuous Learning. We found freelancers often use outdated methods to store passwords securely. This phenomenon, which was also observed in the student sample [21], shows that freelancers do not update their knowledge as well. Yet as Naiakshina et al. observed in [22], using a web framework that provides secure password storage as a default that gains widespread use by tutorials could increase the likelihood of using the latest security standards due to copying and pasting. In fact, similar to the students, we found freelancers pasting in security code they had obtained from the Internet. In some cases we were able to identify this by spotting the same comments in the code snippets which had originally appeared in tutorials and blog posts, and which our freelancers had not bothered to change.

Field vs. Lab Study. Even though we had freelancers create the code under the impression that it would be used in the real world, the results are very similar to the lab study. While this single study cannot decide this on its own, it is an indication that lab studies can deliver valuable insights.

Experience With Freelancer.com

Response Rate. In comparison to the rather low response rate in developer studies conducted with GitHub users, we found Freelancer.com to be a suitable source for recruiting enough willing professional developers to work on (study) projects. Especially for longer studies such as those conducted by Naiakshina et al. [21, 22], Freelancer.com can be an appropriate choice for a recruitment website. This convenience, of course, comes at a cost of hiring the freelancers. Most of the GitHub studies were conducted with volunteers.

Experience With Freelancers. While Bau et al. in [6] reported freelancers to be generally unreliable, Yamashita and Moonen [40, 41] emphasized the flexibility, the access to a wide population, and the low costs of Freelancer.com while also acknowledging the uncertainty of freelancers' backgrounds and skills. Unlike Bau et al., we found our participants to be very dependable. All of our hired freelancers delivered a solution in a reasonable amount of time, with only a handful of participants needing longer to implement the registration functionality than they initially promised. On the other hand, communication was a challenge in some cases, as most of the participants were not fluent in English.

7 CONCLUSION

In this paper we present the results of a field study with 43 freelance developers recruited from Freelancer.com. Broadly speaking, we found similar results as Naiakshina et al. [21, 22] did in their lab study. We confirmed that task framing has a large effect. Importantly, we shed light on the statement made by many of the students in the Naiakshina et al. studies, who claimed that they would have created secure code if they had been doing this for a real client. Our sample shows that freelancers who believe they are creating code for a real company also seldom store passwords securely without prompting. We also highlighted differences between the misconceptions and behaviors of student and freelance developers.

In addition, we found a significant effect in the freelancers' acceptance rate between the €100 and €200 conditions for the prompted task and examined the effect of different payment levels on secure coding behavior. We saw more secure solutions in the €200 conditions, although the difference was not statistically significant. However, this result might be due to the small sample size and we believe this is worth following up in future work.

8 ACKNOWLEDGMENTS

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

REFERENCES

[1] 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.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. 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, San Jose, CA, USA, 289–305. https://doi.org/10.1109/SP.2016.25

[3] 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

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

[5] 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. https://doi.org/10.1109/ICSE.2017.59

[6] Jason Bau, Frank Wang, Elie Bursztein, Patrick Mutchler, and John C Mitchell. 2012. Vulnerability Factors in New Web Applications: Audit Tools, Developer Selection & Languages. *Stanford, Tech. Rep* (2012).

[7] J. Bonneau. 2012. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*. IEEE, San Francisco, CA, USA, 538–552. https://doi.org/10.1109/SP.2012.49

[8] Joseph Bonneau and Sören Preibusch. 2010. The Password Thicket: Technical and Market Failures in Human Authentication on the Web.. In *WEIS*.

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

[10] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 2013. Does my password go up to eleven?: The Impact of Password Meters on Password Selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, ACM, New York, NY, USA, 2379–2388.

[11] Matthew Finifter and David Wagner. 2011. Exploring the Relationship Betweenweb Application Development Tools and Security. In *Proceedings of the 2Nd USENIX Conference on Web Application Development (WebApps'11)*. USENIX Association, Berkeley, CA, USA, 9–9. http://dl.acm.org/citation.cfm?id=2002168.2002177

[12] Jim Finkle and Jennifer Saba. 2012. LinkedIn suffers data breach-security experts. Retrieved May 18, 2017 from http://in.reuters.com/article/linkedin-breach-idINDEE8550EN20120606

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

[14] Dinei Florêncio, Cormac Herley, and Paul C Van Oorschot. 2014. An administrator's guide to internet password research. In *28th Large Installation System Administration Conference (LISA14)*. 44–61.

[15] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic (API) Misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 265–281. https://www.usenix.org/conference/soups2018/presentation/gorski

[16] Ameya Hanamsagar, Simon S Woo, Chris Kanich, and Jelena Mirkovic. 2018. Leveraging Semantic Transformation to Investigate Password Habits and Their Causes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, ACM, New York, NY, USA, 570.

[17] Philip G. Inglesant and M. Angela Sasse. 2010. The True Cost of Unusable Password Policies: Password Use in the Wild. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 383–392. https://doi.org/10.1145/1753326.1753384

[18] Poul-Henning Kamp, P Godefroid, M Levin, D Molnar, P McKenzie, R Stapleton-Gray, B Woodcock, and G Neville-Neil. 2012. LinkedIn Password Leak: Salt Their Hide. *ACM Queue* 10, 6 (2012), 20.

[19] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of Passwords and People: Measuring the Effect of Password-Composition Policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, ACM, New York, NY, USA, 2595–2604.

[20] Peter Mayer, Jan Kirchner, and Melanie Volkamer. 2017. A Second Look at Password Composition Policies in the Wild: Comparing Samples from 2010 and 2016. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 13–28. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/mayer

[21] 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.

https://doi.org/10.1145/3133956.3134082

[22] 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

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

[24] 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. https://doi.org/10.1109/TSE.2010.22

[25] Sean M Segreti, William Melicher, Saranga Komanduri, Darya Melicher, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L Mazurek. 2017. Diversify to Survive: Making Passwords Stronger with Adaptive Policies. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association, Santa Clara, CA, USA, 1–12. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/segreti

[26] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. 2018. Understanding Software Developers' Approach towards Implementing Data Minimization. *arXiv preprint arXiv:1808.01479* (2018).

[27] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2014. Can Long Passwords Be Secure and Usable?. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2927–2936. https://doi.org/10.1145/2556288.2557377

[28] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Designing Password Policies for Strength and Usability. *ACM Transactions on Information and System Security (TISSEC)* 18, 4, Article 13 (May 2016), 34 pages. https://doi.org/10.1145/2891411

[29] Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2010. Encountering Stronger Password Requirements: User Attitudes and Behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security (SOUPS '10)*. ACM, New York, NY, USA, Article 2, 20 pages. https://doi.org/10.1145/1837110.1837113

[30] Kamran Siddiqui. 2013. Heuristics for Sample Size Determination in Multivariate Statistical Techniques. *World Applied Sciences Journal* 27 (01 2013), 285–287. https://doi.org/10.5829/idosi.wasj.2013.27.02.889

[31] Elizabeth Stobert and Robert Biddle. 2014. The Password Life Cycle: User Behaviour in Managing Passwords. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*. USENIX Association, Menlo Park, CA, USA, 243–255. https://www.usenix.org/conference/soups2014/proceedings/presentation/stobert

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

[33] Blase Ur, Jonathan Bees, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Do Users' Perceptions of Password Security Match Reality?. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3748–3760. https://doi.org/10.1145/2858036.2858546

[34] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Julio López. 2012. Helping Users Create Better Passwords. *USENIX* 37, 6 (2012), 51–57. http://www.ece.cmu.edu/~lbauer/papers/2012/login2012-passwords.pdf

[35] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M. Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2015. I Added '!' at the End to Make It Secure: Observing Password Creation in the Lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. USENIX Association, Ottawa, 123–140. https://www.usenix.org/conference/soups2015/proceedings/presentation/ur

[36] Ashlee Vance. 2010. If your password is 123456, just make it hackme. *The New York Times* 20 (2010).

[37] Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. 2016. Understanding Password Choices: How Frequently Entered Passwords are Re-used across Websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association, Denver, CO, 175–188. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash

[38] Chamila Wijayarathna and Nalin AG Arachchilage. [n. d.]. Am I Responsible for End-User's Security? USENIX Association, Baltimore, MD. https://wsiw2018.l3s.uni-hannover.de/

[39] 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. https://doi.org/10.1145/3210459.3210483

[40] Aiko Yamashita and Leon Moonen. 2013. Do Developers Care about Code Smells? An Exploratory Survey. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, IEEE, 242–251.

[41] Aiko Yamashita and Leon Moonen. 2013. Surveying Developer Knowledge and Interest in Code Smells through Online Freelance Marketplaces. In *User Evaluations for Software Engineering Researchers (USER), 2013 2nd International Workshop on*. IEEE, IEEE, San Francisco, CA, USA, 5–8.